# Neural Tensor Factorization for Temporal Interaction Learning

Xian Wu*
University of Notre Dame
xwu9@nd.edu

Baoxu Shi*
University of Notre Dame
bshi@nd.edu

Yuxiao Dong
Microsoft Research
yuxdong@microsoft.com

Chao Huang
University of Notre Dame
chuang7@nd.edu

Nitesh V. Chawla
University of Notre Dame
nchawla@nd.edu

## ABSTRACT

Neural collaborative filtering (NCF) [13] and recurrent recommender systems (RRN) [37] have been successful in modeling relational data (user-item interactions). However, they are also limited in their assumption of static or sequential modeling of relational data as they do not account for evolving users' preference over time as well as changes in the underlying factors that drive the change in user-item relationship over time. We address these limitations by proposing a Neural network based Tensor Factorization (NTF) model for predictive tasks on dynamic relational data. The NTF model generalizes conventional tensor factorization from two perspectives: First, it leverages the long short-term memory architecture to characterize the multi-dimensional temporal interactions on relational data. Second, it incorporates the multi-layer perceptron structure for learning the non-linearities between different latent factors. Our extensive experiments demonstrate the significant improvement in both the rating prediction and link prediction tasks on various dynamic relational data by our NTF model over both neural network based factorization models and other traditional methods.

## CCS CONCEPTS

• **Information systems → Temporal data**;

## KEYWORDS

Tensor Factorization; Temporal Interaction Learning; Deep Learning

**ACM Reference Format:**
Xian Wu, Baoxu Shi, Yuxiao Dong, Chao Huang, and Nitesh V. Chawla. 2019. Neural Tensor Factorization for Temporal Interaction Learning. In *The Twelfth ACM International Conference on Web Search and Data Mining (WSDM'19), February 11–15, 2019, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3289600.3290998

*These authors contributed equally.


*WSDM '19, February 11–15, 2019, Melbourne, VIC, Australia*

ACM ISBN 978-1-4503-5940-5/19/02…$15.00
https://doi.org/10.1145/3289600.3290998

## 1 INTRODUCTION

Learning from relational data (*e.g.*, the user-item interactions on Netflix) has benefited many real-world services and applications, such as rating prediction and item recommendation on online platforms. A significant line of research has shown that latent factor models, in particular factorization based techniques, offer state-of-the-art results for learning tasks on relational data [20, 24, 25]. In addition, as the main framework of the winning solution of the Netflix Prize, matrix factorization (MF) has demonstrated its power on industrial-grade applications [21], further attracting much effort in generalizing its predictive abilities.

One direction of these efforts has been devoted to extending a two-dimensional matrix, representative of interactions between users and items, into a three-dimensional tensor for incorporating the time information. Subsequently, the tensor factorization (TF) technique can be employed to project users and items into a latent space with the encoding of time [3, 19]. However, conventional TF assumes the independence between two consecutive time slots, leaving it infeasible to make predictions for the next time slot. Further, it is also incapable of capturing the temporal patterns that are themselves time-evolving, such as (i) the fast-changing item perception, for example, individuals' impression to a movie may be dynamically affected by its winning of some movie awards. and (ii) the evolution of users' preferences, i.e., user's tastes may change over time.

Recently, another attempt—recurrent recommender networks (RRN)—was made to integrate a recurrent neural network with factorization models for modeling the sequence dependencies between users' behavioral trajectories [37]. However, RRN achieves this by setting a fixed length of items' historical ratings, ignoring the time interval between two consecutive ratings. Consequently, these fixed-length (*e.g.*, $k$) rating sequences may cover various timeframes for different items. This is a limitation of RRN. For example, a popular movie may only take one hour to receive $k$ ratings, while a cult movie may need days to collect the same count of ratings. Additionally, both RRN and conventional TF models utilize dot product to make rating predictions, missing the potential to model the nonlinearities between latent factors.

To address these challenges and limitations, we develop a Neural network based Tensor Factorization model (NTF) for temporal interaction learning. In general, the NTF takes a three-way tensor (i.e., user-item-time) as input and learns the latent embeddings (commonly referred to as factors in TF) for each dimension of the tensor. Specifically, NTF integrates the long short-term memory (LSTM) network with tensor factorization. The LSTM module is used to

adaptively capture the dependencies among multi-dimensional interactions based on the learned representations for each time slot. Furthermore, instead of using dot product between learned representations to make rating predictions, NTF concatenates the inherent factors together and feeds them into a Multilayer Perceptron (MLP) architecture. As such, the learned representations encode the non-linear interactions between different dimensions.

In addition to the aforementioned differences with RRN, our NTF also differs from it with respect to the input to the LSTM module. To predict the next rating, the input to RRN's LSTM is the previous rating sequence with a fixed length, while the LSTM module of NTF takes the representation vectors from previous time slots. Furthermore, NTF is different from the recent work on neural collaborative filtering (NCF) [13] and collaborative deep learning (CDL) [36], whereas they infers the users' and items' latent embeddings under a static scenario. Additionally, different from CDL, NTF does not need auxiliary information and domain knowledge to determine the effectiveness of features. The advantage of the NTF model lies in its complete utilization of each dimension of the relational data.

To sum up, the main contributions of this paper include:

- To the best of our knowledge, we present the first model to generalize tensor factorization for temporal interaction data with deep neural networks, empowering it to model time-evolving multi-dimensional data. We call it NTF.
- We incorporate the multiple-layer perceptron architecture in NTF for modeling the non-linearities in relational data, eliminating the linear limitation of dot product used in conventional tensor factorization.
- We perform extensive experiments for the problems of rating prediction and rating inference in both Netflix and MovieLens dataset, and link prediction in the Github dataset, demonstrating the significant improvements over state-of-the-art baselines, such as RRN and NCF.

## 2 PROBLEM FORMULATION AND TENSOR FACTORIZATION (TF)

In this section, we present the notations and problem formulation. We also provide a quick review of the conventional Tensor Factorization (TF) model and discuss its limitations in modeling the temporal dynamics of relational data.

### 2.1 Problem Formulation

We consider a dynamic scenario wherein there exists evolving pair-wise relations between multiple types of entities (e.g., user, item, and time), such as Netflix users' ratings to various movies on different months, and Github users' repository forks during a week. To model the relationships among entities over time, we use a tensor to represent their time-evolving interactions.

In this work, we focus on the three-way tensor with a temporal dimension. Formally, we construct a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ denoting the first-order tensor of each dimension with a size of $I$, $J$ and $K$, respectively. We denote the entry in the tensor $\mathcal{X}$ as $x_{i,j,k}$ to represent the interactions among different dimensions which are indexed by $i$, $j$ and $k$, respectively. For example, in relational data, $x_{i,j,k}$ can represent: (i) the quantitative rating score of $i$-th user on $j$-th item in $k$-th time slot, and (ii) the binary interactions (links) between $i$-th and $j$-th nodes at time slot $k$.

**Problem Formulation.** Based on the above definitions, we use $x_{i,j,k}$ to represent the interactions among three dimensions in tensor $\mathcal{X}$ which are observed from relational data. The objective of this work is to learn a predictive model that can effectively infer the unknown values in $\mathcal{X}$ with the observed ones.

### 2.2 Conventional Tensor Factorization

The key idea of tensor factorization is learning connections among the observed values in a tensor in order to infer the missing ones. The most common mechanism of tensor factorization is CANDECOMP/PARAFAC (CP) [6], which decomposes a tensor into multiple low-rank latent factor matrices representing each tensor-dimension. For example, movie rating dataset can be viewed as a tensor with 3-dimensions: user, movie, and time. The latent factor matrices in this case, measure the latent factors of each dimension. The latent factors on user-dimension may be users' genre or rating preferences, the factor matrix on movie-dimension may model the movie plot, starring information, and many other features, the time-dimension factors could be specific time related information such as holiday season or special events. With these three matrices, the ratings can be obtained by a simple dot product across the matrices. Following the convention in Representation Learning (RL) literature, in this work we will refer inherent factor vectors as lower-dimensional embedding or representation vectors interchangeably [2].

Formally, TF factorizes a tensor into three different matrices $U \in \mathbb{R}^{I \times L}$, $I \in \mathbb{R}^{J \times L}$ and $T \in \mathbb{R}^{K \times L}$, where $L$ is the number of latent factors (indexed by $l$). We define the factorization of tensor $\mathcal{X}$ as:

$$\mathcal{X} \approx \sum_{l=1}^{L} U_{:,l} \circ I_{:,l} \circ T_{:,l}, \tag{1}$$

where $U_{:,l}$, $I_{:,l}$ and $T_{:,l}$ represents the $l$-th column of matrices $U$, $I$ and $T$ respectively. $\circ$ denotes the vector outer product. Each entry $x_{i,j,k} \in \mathcal{X}$ can be computed by the inner-product of three $L$-dimensional vectors as follows:

$$\hat{x}_{i,j,k} \approx < U_i, I_j, T_k > \equiv \sum_{l=1}^{L} U_{i,l} I_{j,l} T_{k,l}. \tag{2}$$

The objective of tensor factorization is to learn $U$, $I$ and $T$ using maximum likelihood estimation. We further define $U_i$, $I_j$ and $T_k$ to index the row of $U \in \mathbb{R}^{I \times L}$, $I \in \mathbb{R}^{J \times L}$ and $T \in \mathbb{R}^{K \times L}$, respectively. After the model learns $U$, $I$, and $T$ from the observed $\mathcal{X}$, one can easily fill in the missing values in $\mathcal{X}$ using Eq. 2.

However, two significant limitations exist in conventional tensor factorization model: (i) it fails to capture temporal dynamics because the interaction prediction $\hat{x}_{i,j,k}$ only depends on current timeslot $T_{k,:}$; (ii) it is a linear model which cannot deal with complex non-linear interactions that exist in real-world relational data. In the present work we aim to explicitly incorporate temporal dynamics into tensor factorization frameworks and model the non-linear interactions across different latent factors.

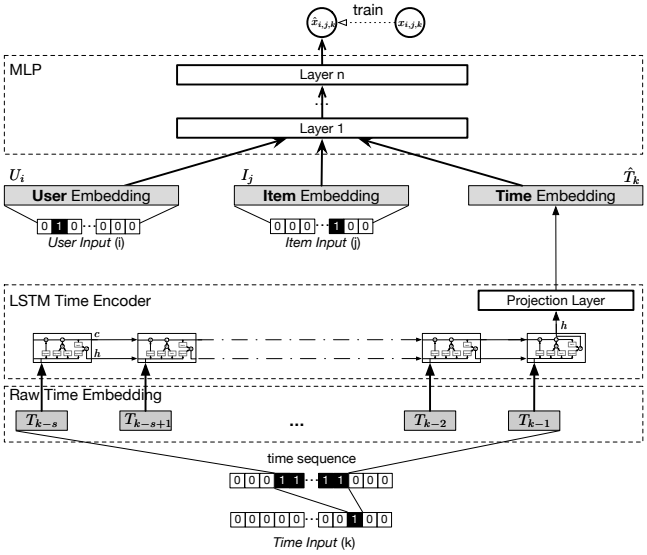



**Figure 1: The NTF Framework.**

## 3 NEURAL NETWORK BASED TENSOR FACTORIZATION FRAMEWORK

In this section, we present the Neural network based Tensor Factorization (NTF) framework, which is capable of learning implicit time-evolving interactions in relational data. We first introduce the general framework of NTF to elaborate the motivation of the model and then present details of NTF in the following subsections.

### 3.1 General Framework

Our NTF is a multi-layer representation learning model which pursues a full neural treatment of tensor factorization to explicitly model the time-evolving interactions between different dimensions. We include the model architecture in Figure 1 and present the pseudo code of NTF in Algorithm 1. The motivations of designing our model are listed as follows:

- To capture the complex temporal dynamics, with the generated embedding vectors as input, we utilize LSTM to encode the evolving interactions addressing the issues of long-term dependencies and vanishing gradients in recurrent neural networks [16]. As a general tensor factorization framework, NTF is also flexible to integrate other variants of recurrent neural networks.
- To model the non-linearity of multi-dimensional interactions, we decide to use a Multi-layer Perceptron (MLP) on top of the first component. Let us consider the generated tensor with user-item-time dimensions as an example. The projected temporal embedding vectors will be fed into a multi-layer neural architecture together with the embedding vectors of users and items. This enables NTF to incorporate the learned complex dependencies in the temporal dimension into the factorization process as constrains. By doing so, we can detect the implicit patterns of user-item-time interactions through each layer in the MLP framework and model the non-linear combinations of latent factors to make better predictions. Finally, the latent vectors will be mapped to quantitative values (*i.e.*, $\hat{x}_{i,j,k}$) which represents future interactions across different dimensions.

**Algorithm 1:** Training the NTF model.

**Input:** Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, observed interaction set $\mathbf{X}$, sequence length $s$, and batch size $b_{\text{size}}$.
**Paras:** Embedding matrices $\mathbf{U} \in \mathbb{R}^{I \times L}$, $\mathbf{I} \in \mathbb{R}^{J \times L}$, $\mathbf{T} \in \mathbb{R}^{K \times L}$, and other hidden parameters $\theta$.

*Initialize all parameters*;
// Sample a minibatch of size $b_{\text{size}}$.
**foreach** $\mathsf{T}_{\text{batch}} = \mathsf{sample}(\mathbf{X}, b_{\text{size}})$ **do**
  **foreach** $\langle i, j, k \rangle \in \mathsf{T}_{\text{batch}}$ **do**
    /* Gather embeddings for all dimensions. */
    $\mathsf{U}_i = \mathbf{U}[i, :]$, $\mathsf{I}_j = \mathbf{I}[j, :]$;
    $c = c_0$, $h = h_0$; // init. hidden states
    **for** t $\leftarrow (k - s)$ **to** $(k - 1)$ **do**
      $c, h = \mathsf{LSTM}(\mathbf{T}[\text{t}, :], c, h)$; // according to Eq.3
    **end**
    $\hat{\mathbf{T}}_k = \mathsf{Projection}(h)$; // encoded time embedding
    $\hat{x}_{i,j,k} = \mathsf{MLP}([\mathsf{U}_i; \mathsf{I}_j; \hat{\mathbf{T}}_k])$; // according to Eq.6
    $x_{i,j,k} = \mathcal{X}[i, j, k]$;
    update loss $\mathcal{L}$ w.r.t $\frac{1}{2}(x_{i,j,k} - \hat{x}_{i,j,k})^2$;
  **end**
  *update all parameters w.r.t* $\mathcal{L}$;
**end**

### 3.2 Modeling Temporal Dynamics via LSTM

To obviate the gradient vanishing and exploding problems and model long-distance dependencies in sequence data, LSTM is introduced as a special kind of RNN to model long-term dependencies and addresses the vanishing gradient problem by developing a more complicated hidden unit. In particular, LSTM proposes to derive the vector representations of hidden states $h_t$ and $c_t$ for each time step $t$ as follows:

$$\begin{aligned} i_t &= \sigma(W_i h_{t-1} + V_i x_t + b_i) \\ o_t &= \sigma(W_o h_{t-1} + V_o x_t + b_o) \\ f_t &= \sigma(W_f h_{t-1} + V_f x_t + b_f) \\ \widetilde{c_t} &= \phi(W_c h_{t-1} + V_c x_t + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \widetilde{c_t} \\ h_t &= o_t \odot \phi(c_t) \end{aligned} \tag{3}$$

where $W_* \in \mathbb{R}^{d_s \times d_s}$ represents the transformation matrix from the previous state (*i.e.*, $c_{t-1}$ and $h_{t-1}$) to LSTM cell and $V_* \in \mathbb{R}^{d_s \times d_x}$ are the transformation matrices from input to LSTM cell, where $d_s$ and $d_x$ denotes the dimension of hidden states and input vectors, respectively. Furthermore, $b_* \in \mathbb{R}^{d_s}$ is defined as a vector of bias term. $\sigma(\cdot)$ and $\phi(\cdot)$ represents the *sigmoid* and $tanh$ function, respectively. The $\odot$ operator denotes the element-wise product. In Eq. 3, $i_t$, $o_t$, and $f_t$ represents input gate, output gate and forget gate, respectively. For simplicity, we denote Eq. 3 as $[c_t, h_t] = \text{LSTM}(*, c_{t-1}, h_{t-1})$ in the following subsections.

### 3.3 Fusion of LSTM and TF

In this subsection, we present how we fuse LSTM and TF under the NTF framework to model the time-evolving interactions across

three dimensions $\mathbb{R}^I$, $\mathbb{R}^J$ and $\mathbb{R}^K$. In relational data, dynamic characteristics are observed across different time slots. In this work, we consider temporal factors affecting the interactions over time based on a global trend by assuming that the interactions between multi-dimensions evolve in a smooth way. In particular, to capture the temporal smoothness, we further assume that the embedding vectors of the temporal dimension depends on embedding vectors from previous $s$ time slots. In NTF, we predict the embedding vector in current time slot based on the embedding vectors from past $s$ time slots using LSTM.

In our framework, our LSTM encoder generates embedding to encode the evolving temporal hidden factors. We formally define the hidden states $c_t$ and $h_t$ in encoding the contextual sequence as:

$$\begin{aligned}[c_1, h_1] =& \text{LSTM}(T_{k-s}, c_0, h_0)\\ &\cdots\\ [c_{s-1}, h_{s-1}] =& \text{LSTM}(T_{k-1}, c_{s-2}, h_{s-2})\end{aligned} \tag{4}$$

Using the last hidden state vector $h_{s-1}$, we can define the embedding vector $\hat{T}_k$ through Projection Layer as:

$$\hat{T}_k = \sigma(W_T h_{s-1} + b_T) \tag{5}$$

where $W_T \in \mathbb{R}^{s \times L}$ is the projection matrix, $b \in \mathbb{R}^L$ is the projection bias. $\sigma$ denotes the *sigmoid* funtion.

Finally, the predictive value $\hat{x}_{i,j,k}$ could be derived by the dot product of $U_i$, $I_j$, $\hat{T}_k$ according to the conventional tensor factorization. With the increasing utilization of deep neural networks to handle complicated non-linearity in image and text data [8, 26], it is intuitive to explore the non-linear interactions in relational data. To address this issue, we concatenate embedding vectors of $U_i$, $I_j$, $\hat{T}_k$ together and consider them as input to the Multi-Layer Perceptron (MLP) and output $\hat{x}_{i,j,k}$. In this way, we can address the limitation caused by the dot product in tensor factorization (as introduced in Section 2) with a neural network architecture to capture non-linear interactions by concatenating latent factors from the previous embedding layer. Formally, we present MLP as:

$$\begin{aligned}Z_1 =& \phi_1(W_1[U_i; I_j; \hat{T}_k] + b_1)\\ &\cdots\\ Z_n =& \phi_n(W_n Z_{n-1} + b_n)\\ \hat{x}_{i,j,k} =& W_o Z_n + b_o\end{aligned} \tag{6}$$

where $n$ represents the number of hidden layers which is indexed by $l$ and ; represents the concatenate operation. For the $Z_l$ layer, $\phi_l$, $W_l$ and $b_l$ represents the activation function (e.g., *ReLU* or $tanh$ function) of MLP layers, weight matrix and bias vector, respectively. In the experiments, we investigate the effect of activation function and number of layers in MLP.

## 3.4 Learning Process

In this subsection, we first describe the learning process of our NTF framework. Then, we further utilize the advanced technique, *i.e.*, *batch normalization*, to optimize the NTF.

*3.4.1 The Objective Function.* As we introduced in the Section 2, our objective is to derive the value of $\hat{x}_{i,j,k}$ which denotes the interactions between $i$-th, $j$-th and $k$-th elements of first-order tensor $\mathbb{R}^I$, $\mathbb{R}^J$ and $\mathbb{R}^K$ in $\mathcal{X}$, respectively. We formally define our objective function in factorization procedure as follows:

$$\arg\min_{U,I,T} \frac{1}{2} \sum_{(i,j,k)\in \mathbf{X}} (x_{i,j,k} - \hat{x}_{i,j,k})^2 \tag{7}$$

where $\mathbf{X}$ denotes the set of observed interactions in tensor $\mathcal{X}$. The NTF can be learned by minimizing the above loss function between the observed interaction data and the factorization representation.

*3.4.2 Batch Normalization.* In the training process of neuron network models, their performances could be degraded by covariance shift [32]. To tackle this challenge, *Batch Normalization (BN)* has been proposed to normalize the input data from previous layer before sending it to the next layer as input [15]. In NTF framework, we apply *BN* to reduce the internal covariance shift by transforming the input to zero mean/unit variance distributions in each mini-batch training. In NTF, we apply the *BN* to LSTM to avoid the deceleration in training process as:

$$\widetilde{c_t} = \phi(BN(W_c h_{t-1} + V_c x_t + b_c)) \tag{8}$$

where $BN(\cdot)$ stands for the batch normalization operation. *BN* is also applied to Projection Layer.

# 4 EVALUATION

We demonstrate the effectiveness of NTF with three real-world applications on dynamic relational data, *i.e.*, regression-*rating prediction and inference*, and classification-*link prediction* corresponding to predicting and inferring quantitative (rating scores), and binary (existence of links) interactions in a dynamic scenario, respectively. In particular, we aim to answer the following questions:

- **Q1**: How does our NTF framework perform as compared to the state-of-the-art techniques in rating prediction task?
- **Q2**: Can NTF outperform the competitive baselines in rating inference task?
- **Q3**: How does our NTF framework work for link prediction task when competing with baselines?
- **Q4:** Does NTF consistently outperform other baselines in terms of prediction accuracy with respect to different time windows with different training and testing time period?
- **Q5:** Does NTF consistently achieve the best performance compared with competitive baselines in terms of inference accuracy with respect to different training and test ratio?
- **Q6**: How is the performance of NTF variants with different combinations of key components in the proposed framework?
- **Q7**: How different hyperparameter settings affect the performance of NTF?

## 4.1 Experimental Setup

*4.1.1* **Data.** In our evaluation, we perform experiments on three types of dynamic relational datasets and corresponding tasks. Table 1 summarizes the statistics of the above three datasets.

**Netflix Rating Data**[1]. This movie rating dataset has been widely used in rating prediction evaluation [31, 40]. In the Netflix dataset,

[1] https://www.netflix.com/

**Table 1: The Statistics of Datasets**

| Dataset | # of Users | # of Items | # of Ratings | Time Span |
|---|---|---|---|---|
| Netflix | 68,079 | 2,328 | 12,326,319 | 36 months |
| MovieLens | 6,040 | 3,900 | 1,000,209 | 34 months |
| Dataset | # of Users | # of Projects | # of Fork | Time Span |
| Github | 81,001 | 72,420 | 1,396,115 | 21 weeks |

users rate a movie using a 1 (worst) to 5 (best) scale, the given score is also associated with a rating date to denote when the rating was reported. We generate tensor $\mathcal{X}$ by associating each movie with the users who rated this movie on different months, which was collected between Jan 2002 and Dec 2005. In particular, if $i$-th user rated $j$-th movie on $k$-th month (the time slots we used for evaluation correspond to calendar months) in the dataset, the element $x_{i,j,k}$ represents the multi-dimensional interaction in $\mathcal{X}$.

**MovieLens Rating Data**[2]. This movie rating dataset has been widely used to evaluate rating prediction and inference approaches [10, 13]. In this dataset, each user has at least 20 ratings. The rating scale is consistent with that in Netflix data. The dataset is collected from Apr 2000 to Feb 2003. We generate the tensor $\mathcal{X}$ in a similar way as Netflix data.

**Github Archive Data**[3]. We collected this dataset from Github to record the *fork* actions of users on repositories from Jan 2017 to May 2017. In this dataset, an edge $(i, j, k)$ is generated when $i$-th user forks the $j$-th repository at $k$-th time slot (the time slots we used for evaluation correspond to calendar weeks). We set the element $x_{i,j,k}$ in tensor $\mathcal{X}$ to 1 if edge $(i, j, k)$ exists in the dataset and 0 otherwise.

**Table 2: Parameter Settings**

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| Hidden state dimension | 32 | Embedding size | 32 |
| # of Time steps | 5 | # of Hidden layers | 6 |
| BN scale parameter | 0.99 | BN shift parameter | 0.001 |
| Batch size | 256 | Learning rate | 0.001 |

*4.1.2* **Parameter Settings.** We summarize the parameter settings of NTF and experiments in Table 2. In addition, we vary each of key parameters in NTF and fix others to examine the parameter sensitivity. We implement our framework based on TensorFlow and chose Adam [18] as our optimizer to learn the model parameters[4].

*4.1.3* **Baselines.** Because rating prediction, rating inference and link prediction are three different tasks and have different representative baselines. Here, we summarize the compared baselines of these tasks separately. In addition, the reason to compare NTF with matrix factorization methods rather than tensor factorization schemes is mainly twofold: (1) It is difficult to apply tensor factorization with temporal dimension to make predictions due to the ignorance of temporal dependencies between time slots. (2) There is no duplicate interactions (*i.e.*, ratings and links) existed between users and items in different time slots in all datasets.

[2]https://grouplens.org/datasets/movielens/1m/
[3]https://www.githubarchive.org/
[4]The data and code can be found at https://bitbucket.org/xianwu9/ntf/src/master/

**Table 3: Baseline Summary**

| Method | PMF | BPMF | AA | AP | TDSSM | RRN | NCF | BPTF | NTF |
|---|---|---|---|---|---|---|---|---|---|
| Rating Prediction | ✓ | ✓ | | | ✓ | ✓ | ✓ | | ✓ |
| Rating Inference | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Link Prediction | ✓ | ✓ | ✓ | ✓ | | | ✓ | | ✓ |

**Rating Prediction and Inference**. For rating prediction and inference tasks, we consider three types of baselines: (i) representative matrix factorization techniques for recommendation systems (*i.e.*, PMF, BPMF and BPTF), neural network based collaborative filtering methods for recommendations or predictive analytics (*i.e.*, NCF), and variants of Recurrent Neural Network models for time series prediction (*i.e.*, RRN and TDSSM).

- **Probabilistic Matrix Factorization (PMF)** [25]: it is a probabilistic method for matrix factorization, which assigns a D-dimensional latent feature vector (following Gaussian distributions) for each user and item. The ratings are derived from the inner-product of corresponding latent features.
- **Bayesian Probabilistic Matrix Factorization (BPMF)** [28]: extended from the PMF, BPMF learns the latent feature vector for each user and item by using Monte Carlo Markov Chain method, which is able to address the overfitting issue.
- **Bayesian Probabilistic Tensor Factorization (BPTF)** [40]: it is a bayesian probabilistic tensor factorization method for modeling evolving relational data.
- **Temporal Deep Semantic Structured Model (TDSSM)** [33]: this method is a temporal recommendation model which combines traditional feedforward networks (DSSM) with LSTM, to capture temporal dynamics of users' interests.
- **Recurrent Recommendation Networks (RRN)** [37]: it aims to predict future interactions between users and items by specifying two embedding vectors (stationary and dynamic) for both user and item. The dynamic embedding vectors are inferred with LSTM model based on historical ratings.
- **Neural Collaborative Filtering (NCF)** [13]: it proposes a framework for collaborative filtering based on neural network architecture to model the interactions between users and items.

**Link Prediction**. In addition to the above NCF, BPMF and PMF algorithms which have been applied to solve the link prediction problem, we consider other two traditional link prediction baselines to compare the performance of NTF in predicting future binary interactions (*i.e.*, links) between users and items.

- **Preferential Attachment (PA)** [23]: PA assumes new connections are more likely to form between well-connected nodes.
- **Adamic Adar (AA)** [1]: AA smoothes the common neighbor method using neighbors' node degree.
- **PMF**, **BPMF**, and **NCF**: as introduced above.
- **TDSSM** and **RRN**: both methods require the historical sequences of items and users. However, the sequences cannot be generated from the Github dataset due to the following two reasons: First, the unobserved interactions may be either negative or positive cases; Second, if we take all unobserved cases as negative, it is also difficult to locate them to specific time slots.

For all neural network based methods, we use the same parameters as NTF which are listed in Table 2.

**Table 4: Performance of rating prediction for all compared algorithms *w.r.t.* different time windows on Netflix data.**

| Month - 2004 | Jan | | Mar | | May | | Jul | | Sep | | Nov | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Metrics | RMSE | MAE | RMSE | MAE | RMSE | MAE | RMSE | MAE | RMSE | MAE | RMSE | MAE |
| PMF | 0.9385 | 0.7331 | 0.9274 | 0.7263 | 0.9243 | 0.7171 | 0.8968 | 0.6972 | 0.8978 | 0.6923 | 0.8885 | 0.6856 |
| BPMF | 0.9879 | 0.7686 | 0.9829 | 0.7659 | 0.9741 | 0.7541 | 0.9449 | 0.7319 | 0.9415 | 0.7264 | 0.9312 | 0.7165 |
| TDSSM | 1.0031 | 0.8001 | 1.0386 | 0.8488 | 0.9897 | 0.7886 | 0.9365 | 0.7442 | 0.9259 | 0.7306 | 0.9401 | 0.7413 |
| RRN | 1.0062 | 0.7936 | 0.9901 | 0.7798 | 0.9721 | 0.7584 | 0.9328 | 0.7276 | 0.9574 | 0.7529 | 0.9227 | 0.7146 |
| NCF | 0.9498 | 0.7517 | 0.9364 | 0.7357 | 0.9421 | 0.7408 | 0.9069 | 0.7148 | 0.9080 | 0.7071 | 0.9089 | 0.7068 |
| NTF*dot* | 0.9869 | 0.7763 | 0.9736 | 0.7702 | 0.9600 | 0.7523 | 0.9231 | 0.7221 | 0.9237 | 0.7186 | 0.9123 | 0.7089 |
| NTF(*ReLU*) | 0.9192 | 0.7204 | 0.9111 | 0.7169 | 0.9127 | 0.7131 | 0.8823 | 0.6869 | 0.8871 | 0.6923 | 0.8753 | 0.6787 |
| NTF(*sigmoid*) | 0.9158 | 0.7178 | 0.9113 | 0.7148 | 0.9141 | 0.7110 | 0.8802 | 0.6889 | 0.8882 | 0.6875 | 0.8751 | 0.6793 |
| NTF($tanh$) | 0.9178 | 0.7187 | 0.9128 | 0.7185 | 0.9135 | 0.7111 | 0.8834 | 0.6896 | 0.8865 | 0.6941 | 0.8779 | 0.6783 |

**Table 5: Performance of rating prediction for all compared algorithms *w.r.t.* different time windows on MovieLens data.**

| Month - 2001 | May | | Jun | | Jul | | Aug | | Sep | | Nov | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Metrics | RMSE | MAE | RMSE | MAE | RMSE | MAE | RMSE | MAE | RMSE | MAE | RMSE | MAE |
| PMF | 1.1625 | 0.8848 | 1.0905 | 0.8377 | 1.1968 | 0.8909 | 1.2752 | 0.9712 | 1.4659 | 1.1247 | 1.2463 | 0.9545 |
| BPMF | 0.9345 | 0.7260 | 0.8810 | 0.6932 | 0.8925 | 0.7015 | 0.9249 | 0.7286 | 0.8895 | 0.6955 | 0.9141 | 0.7136 |
| TDSSM | 1.0482 | 0.8510 | 0.9648 | 0.7866 | 1.014 | 0.8123 | 1.1043 | 0.9097 | 1.0274 | 0.8371 | 1.0096 | 0.8114 |
| RRN | 0.9703 | 0.7492 | 0.9458 | 0.7362 | 0.9709 | 0.7501 | 1.0094 | 0.7899 | 1.007 | 0.7958 | 1.0122 | 0.7968 |
| NCF | 0.9205 | 0.7173 | 0.8942 | 0.7125 | 0.9015 | 0.7055 | 0.9332 | 0.7392 | 0.8861 | 0.6911 | 0.9269 | 0.7203 |
| NTF*dot* | 0.9381 | 0.7337 | 0.9248 | 0.7268 | 0.9376 | 0.7339 | 0.9674 | 0.7647 | 0.9298 | 0.7320 | 0.9706 | 0.7692 |
| NTF(*ReLU*) | 0.9091 | 0.7038 | 0.8683 | 0.6835 | 0.8767 | 0.6828 | 0.9125 | 0.7201 | 0.8563 | 0.6655 | 0.8985 | 0.6996 |
| NTF(*sigmoid*) | 0.9073 | 0.7017 | 0.8637 | 0.6840 | 0.8761 | 0.6869 | 0.9068 | 0.7134 | 0.8527 | 0.6688 | 0.8960 | 0.7033 |
| NTF($tanh$) | 0.9051 | 0.7055 | 0.8670 | 0.6819 | 0.8773 | 0.6838 | 0.9059 | 0.7138 | 0.8536 | 0.6649 | 0.9042 | 0.7099 |

*4.1.4* **Variants of NTF.** In addition to comparing NTF with existing approaches, we are also interested in discovering the best way to model non-linear multi-dimensional interactions among different embeddings in the proposed NTF framework. Namely, we aim to answer the following two questions: (1) does the selection of activation functions affect the performance of NTF? and (2) is multiple-layer perceptron, helpful for learning non-linear interactions from multi-dimensional relational data? Hence, in the evaluation of NTF framework, we consider four variants of NTF: *NTFdot*: a simplified version of NTF which does not use MLP to explore the non-linear interactions in relational data. Instead, it uses dot product to predict value $\hat{x}_{i,j,k}$, which is also applied in compared baselines. *NTF(ReLU)*, *NTF(sigmoid)*, and *NTF(tanh)*: the full version of NTF that uses different activation functions.

*4.1.5* **Evaluation Protocols. Rating Prediction and Inference**. To evaluate the performance of all compared algorithms in inferring quantitative rating scores, we use *Root Mean Square Error (RMSE)* and *Mean Absolute Error (MAE)* which have been widely adopted in quantitative prediction tasks [12]. Note that a lower RMSE and MAE score indicates better performance. The mathematical definitions of those metrics are presented as follows: $RMSE = \sqrt{\frac{1}{N}\sum_{(i,j,k)\in\mathcal{X}}(x_{i,j,k} - \hat{x}_{i,j,k})^2}$ and $MAE = \frac{1}{N}\sum_{(i,j,k)\in\mathcal{X}}|x_{i,j,k} - \hat{x}_{i,j,k}|$ where $N$ denotes the number of observed elements in tensor $\mathcal{X}$, $x_{i,j,k}$ and $\hat{x}_{i,j,k}$ represents the actual rating score and estimated rating score, respectively.

**Link Prediction**. To validate the performance of each method in predicting the existences of links, *Precision*, *Recall*, *F1-score* and *AUC* are used as evaluation metrics [29].

## 4.2 Rating Prediction (Q1, Q4 and Q6)

We now compare NTF with state-of-the-art techniques as we introduced above. In this experiments, we split the datasets into training (12 months) and testing (1 month) sets. In the training dataset, we select 10% as the validation dataset to tune hyperparameters and use test datasets to evaluate the final performance of all compared algorithms. All prediction and inference experiments are conducted across 30 consecutive days in test time frames and the average performance is reported. To investigate the performance of all compared algorithms on different targeted time frames, we show the results from different months on Netflix and MovieLens in Table 4 and Table 5, respectively.

**Training/test time period**. We observe that NTF consistently achieves the best performance over different time frames from Table 4. For example, for Netflix data, NTF achieves on average 0.075 and 0.076 (relatively 8.3% and 10.7%) improvements over TDSSM, and 0.075 and 0.075 (relatively 7.4% and 7.8%) improvements over RRN in terms of RMSE and MAE. The evaluation results across different time frames demonstrate the effectiveness of our NTF framework in modeling time-evolving interactions between multiple dimensions in a dynamic scenario. Furthermore, the Netflix data becomes more dense as time window slides, *i.e.*, density degree: Jan-3.67%, Mar-3.86%, May-4.11%, Jul-4.34%, Sep-4.45% and Nov-4.57%. We can observe that the performance gain between NTF and other baselines become larger as data becomes sparser, suggesting our NTF is capable of handing sparse relational data.

**Evaluations on variants of NTF**˙ We can notice that our NTF is not sensitive to the activation function and can reach high performance with different function choice. In addition, our results also indicate that the necessity of our multi-layer perceptron component in NTF for projecting inherent factors into a prediction output by capturing the non-linear relations between them.

**Performance gain analysis**. We can observe that NTF shows improvement over conventional collaborative filtering algorithms (*i.e.*, PMF, BPMF and BPTF), deep collaborative filtering models (*i.e.*, NCF) and recurrent neural networks based schemes (*i.e.*, RRN and TDSSM). In particular, *Firstly*, this sheds light on the limitations of collaborative filtering based algorithms which ignore the temporal dynamics among the multi-dimensional interactions in the rating data. *Secondly*, the large performance gap between NTF and recurrent neural network based schemes indicates the limitation of those approaches which only model the sequential pattern of the tensor's temporal dimension and fail to consider the dependencies between the implicit interactions across dimensions.

## 4.3 Rating Inference (Q2, Q5 and Q6)

We now compare NTF with state-of-the-art baselines in rating inference task on both Netflix and MovieLens datasets. In this experiments, we vary the ratio of training data and fix the validation data ratio as 10%, and consider the remaining data as testing.

**Training/test data ratio**. Table 6 and Table 7 show the prediction results on Netflix and MovieLens, respectively, when varying the percentage of data in the training set. In this experiment, we fix the percentage of validation data as 10% of the entire dataset. We can observe that obvious improvements can be obtained by our NTF with different sizes of training data, demonstrating that NTF is robust to the data sparsity issue. In addition, we can observe rising trends as the training size increases, which indicates the positive effects of training data size on predicting ratings.

**Evaluations on variants of NTF**. From Table 6 and Table 7, we can observe that the proper incorporation of multiple-layer perceptron shows great superiority against the one which uses dot product, which suggests the effectiveness of our NTF in capturing non-linear interactions from multi-dimensional relational data. Additionally, the selection of activation function has low effect on the model performance in rating inference.

**Table 6: Performance comparison of rating inference *w.r.t.* different percentages of training data on Netflix data.**

| Training ratio | 30% | | 50% | | 70% | |
|---|---|---|---|---|---|---|
| Metrics | RMSE | MAE | RMSE | MAE | RMSE | MAE |
| PMF | 0.9241 | 0.7154 | 0.8852 | 0.6813 | 0.8562 | 0.6588 |
| BPMF | 0.9328 | 0.7204 | 0.8920 | 0.6904 | 0.8673 | 0.6698 |
| BPTF | 0.9167 | 0.7384 | 0.8896 | 0.6971 | 0.8625 | 0.6750 |
| TDSSM | 0.9591 | 0.7631 | 0.9443 | 0.7613 | 0.8592 | 0.6665 |
| RRN | 0.9223 | 0.7210 | 0.8842 | 0.6891 | 0.8754 | 0.6805 |
| NCF | 0.9124 | 0.7211 | 0.8810 | 0.6924 | 0.8629 | 0.6692 |
| NTF*dot* | 0.9161 | 0.7184 | 0.8779 | 0.6846 | 0.8547 | 0.6639 |
| NTF(*ReLU*) | 0.8759 | 0.6829 | 0.8632 | 0.6708 | 0.8494 | 0.6586 |
| NTF(*sigmoid*) | 0.8747 | 0.6830 | 0.8637 | 0.6729 | 0.8476 | 0.6538 |
| NTF($tanh$) | 0.8781 | 0.6852 | 0.8661 | 0.6743 | 0.8560 | 0.6649 |

**Table 7: Performance comparison of rating inference *w.r.t.* different percentages of training data on MovieLens data.**

| Training ratio | 30% | | 50% | | 70% | |
|---|---|---|---|---|---|---|
| Metrics | RMSE | MAE | RMSE | MAE | RMSE | MAE |
| PMF | 1.8005 | 1.4570 | 1.2530 | 0.9635 | 1.0836 | 0.8383 |
| BPMF | 0.9201 | 0.7275 | 0.9164 | 0.7243 | 0.9140 | 0.7229 |
| BPTF | 0.9184 | 0.7269 | 0.9144 | 0.7230 | 0.9112 | 0.7208 |
| TDSSM | 1.1696 | 0.9620 | 1.0775 | 0.8863 | 1.0169 | 0.8242 |
| RRN | 1.1769 | 0.9352 | 1.0575 | 0.8354 | 0.9898 | 0.7818 |
| NCF | 0.9362 | 0.7432 | 0.9260 | 0.7368 | 0.9170 | 0.7239 |
| NTF*dot* | 1.0782 | 0.8592 | 0.9861 | 0.7795 | 0.9436 | 0.7447 |
| NTF(*ReLU*) | 0.9104 | 0.7177 | 0.9021 | 0.7111 | 0.8940 | 0.7068 |
| NTF(*sigmoid*) | 0.9077 | 0.7152 | 0.8909 | 0.7001 | 0.8851 | 0.7006 |
| NTF($tanh$) | 0.9072 | 0.7175 | 0.8933 | 0.7068 | 0.8829 | 0.6991 |

**Performance gain analysis**. the performance gain between NTF and other baselines becomes larger as the training size decreases, which validates the ability of NTF model in predicting interactions with sparse tensor. We can notice that neural network based models (*i.e.*, NCF and RRN) achieve better performance compared with conventional matrix factorization algorithms (*i.e.*, PMF, BPTF and BPMF) with less training data. This observation suggests that neural network based models are more suitable in sparse relational data. An interesting observation is that BPTF achieves better performance than TDSSM and RRN (neural methods), which indicates the effectiveness to consider the interactions between user, item and time embeddings.

## 4.4 Link Prediction (Q3 and Q6)

Table 8 lists the evaluation results of the link prediction task on Github dataset. In this evaluation, we evaluate all compared models using 90% and 10% of the Github archive data from first twenty weeks as training and validation data to predict links in the twenty-first week. To construct the testing set, we use the observed links in the twenty-first week as the positive cases and randomly enumerate node pairs and choose unobserved edges as negative cases. Overall,

**Table 8: Performance of link prediction on Github data.**

| Metrics | AUC | F1-score | Precision | Recall |
|---|---|---|---|---|
| AA | 0.5159 | 0.1334 | 0.4641 | 0.0768 |
| PA | 0.6443 | 0.3760 | 0.5565 | 0.2842 |
| PMF | 0.7278 | 0.5233 | 0.7507 | 0.4013 |
| BPMF | 0.7165 | 0.5291 | 0.6978 | 0.4250 |
| NCF | 0.7346 | 0.5639 | 0.6421 | 0.5029 |
| NTF*dot* | 0.7133 | 0.5328 | 0.6681 | 0.4437 |
| NTF(*ReLU*) | 0.7841 | 0.6231 | 0.6985 | 0.5612 |
| NTF(*sigmoid*) | 0.7787 | 0.6278 | 0.6836 | 0.5803 |
| NTF(*tanh*) | 0.7739 | 0.6186 | 0.6765 | 0.5689 |

the proposed NTF significantly outperforms other baselines in terms of F1-score and AUC. Specifically, the relative improvement of our NTF over NCF is 10.2% and 6.8% in terms of F1-score and AUC, respectively. This link prediction task further demonstrates that the NTF framework works well by capturing time-evolving interactions between different dimensions in a non-linear manner.

### 4.5 Parameter Sensitivity Studies (Q7)

To investigate the robustness of the NTF framework, we examine how the different choices of parameters affect the performance of NTF in both rating and link prediction tasks. Except for the parameter being tested, we set other parameters at the default values (see Table 2).

Figure 2 lists the rating prediction results (measured by RMSE and MAE) as a function of one selected parameter when fixing others. Note that we have two $y$-axes corresponding to RMSE (left-black) and MAE (right-blue) respectively due to their different value ranges. From Figure 3, overall, we can observe that NTF is not strictly sensitive to these parameters, except for the # of hidden layers, and can achieve high performance with cost-effective parameters, *i.e.*, the smaller the parameters are, the more efficient the training process will be. Figure 2(b) indicates that model performance becomes stable as long as the number of hidden layers is above 2. Furthermore, from Figure 3(a), we can observe that embedding size is positively correlated with the prediction accuracy and low impact of other two parameters (*i.e.*, # of time steps and state size in LSTM) on model performance. The above observations suggest the robustness of our NTF in modeling the temporal dynamics of multi-dimensional interactions. Similar results can be observed for the link prediction task.

## 5 RELATED WORK

There is a good amount of work on the applications of conventional matrix and tensor factorization [4, 11, 25, 38, 40, 41], such as recommendation systems. In particular, recommendation techniques can represent the interactions between users and items as a matrix and utilize the matrix factorization approach to model the latent factors of user-item interaction structures. To model the temporal effects in user-item interactions, the matrix factorization could be extended to handle the generated tensor data [40].

With the advent of deep learning techniques, significant effort has been made to develop neural network-based matrix factorization models [9, 17, 27, 30]. To address the sparsity problem in recommendation techniques, Kim *et al.* designed a model which integrates convolutional neural network (CNN) into probabilistic matrix factorization (PMF) [17]. However, the limitation of the above approaches is that they only consider static data instead of dynamic data in which temporal dimension need to be explored.

Our work is also related to the works that focus on relational data modeling [5, 7, 13, 14, 22, 30, 34–36, 39]. In particular, He *et al.* aimed to develop a neural network relational data modeling framework by modeling latent features of users and items [13]. Furthermore, Hsieh *et al.* studied the connection between metric learning and interaction modeling. Sedhain *et al.* proposed an autoencoder framework for interaction learning [30]. Wang *et al.* proposed a hierarchical Bayesian model which integrated content information by performing deep representation learning [36]. However, these approaches are static models and are lacking when they comes to dynamic scenarios. The proposed NTF addresses this problem by modeling temporal evolution of latent factors in modeling dynamic relational data. Our work furthers the investigation on this direction by developing the NTF framework to capture the time-evolving temporal dynamics exhibited from relational data with multiple types of entity dependencies, which cannot be handled by previous models.

## 6 CONCLUSION

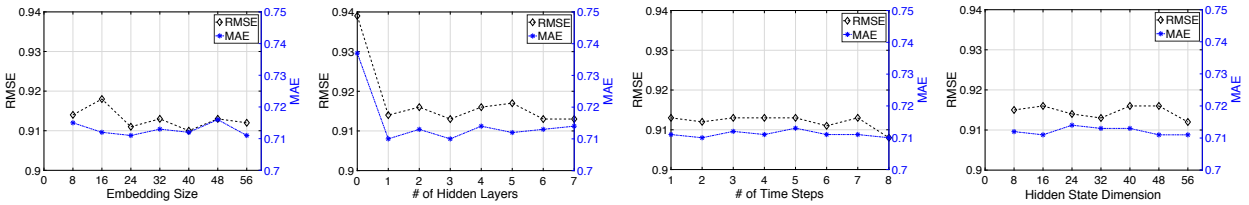
We developed a novel Neural network based Tensor Factorization (NTF) for modeling temporal interaction data that addresses the critical challenge of evolving user-item relational data. By modeling the time-evolving inherent factors and incorporating temporal smoothness constraints on those factors, NTF is capable of capturing both the time-varying interactions across dimensions and the non-linear relations between them. Extensive experiments on three real-world datasets in rating prediction and link prediction tasks show that NTF significantly outperforms baseline methods.

## ACKNOWLEDGMENTS

This work is supported by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, and NSF Grants IIS-1447795 and CNS-1622914.

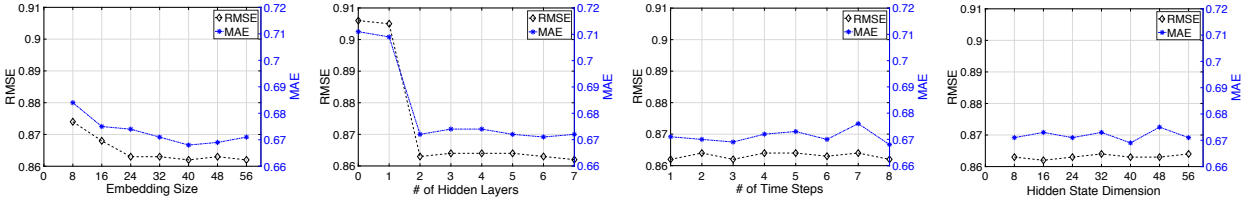
## REFERENCES

[1] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.
[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.
[3] Preeti Bhargava, Thomas Phan, Jiayu Zhou, and Juhan Lee. 2015. Who, what, when, and where: Multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data. In *WWW*. ACM, 130–140.
[4] Jiajun Bu, Xin Shen, Bin Xu, Chun Chen, Xiaofei He, and Deng Cai. 2016. Improving collaborative recommendation via user-item subgroups. *TKDE* 28, 9 (2016), 2363–2375.
[5] Iván Cantador, Alejandro Bellogín, and David Vallet. 2010. Content-based recommendation in social tagging systems. In *Recsys*. ACM, 237–240.




(a) Embedding Size (b) # of Hidden Layers (c) # of Time Steps (d) Hidden State Dimension

**Figure 2: Parameter sensitivity of NTF in rating prediction on Netflix data (data from May 2003 to Apr 2004 is used for training and validation, data from May 2004 is used for testing).**




(a) Embedding Size (b) # of Hidden Layers (c) # of Time Steps (d) Hidden State Dimension

**Figure 3: Parameter sensitivity of NTF in rating inference on Netflix data (50% as training data, 10% as validation data and the remaining as testing data).**


[6] J Douglas Carroll and Jih-Jie Chang. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika* 35, 3 (1970), 283–319.

[7] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *SIGIR*. ACM, 335–344.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR* 12, Aug (2011), 2493–2537.

[9] Gintare Karolina Dziugaite and Daniel M Roy. 2015. Neural network matrix factorization. *ICLR* (2015).

[10] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *TIIS* 5, 4 (2016), 19.

[11] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. ACM, 507–517.

[12] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *SIGIR*. ACM.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. ACM, 173–182.

[14] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *WWW*. ACM, 193–201.

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*. 448–456.

[16] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *ICML*. 2342–2350.

[17] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. 2016. Convolutional matrix factorization for document context-aware recommendation. In *Recsys*. ACM, 233–240.

[18] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[19] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[20] Yehuda Koren. 2008. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *KDD '08*. ACM, New York, NY, USA, 426–434.

[21] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37.

[22] Xiaopeng Li and James She. 2017. Collaborative Variational Autoencoder for Recommender Systems. In *KDD*. ACM, 305–314.

[23] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *journal of the Association for Information Science and Technology* 58, 7 (2007), 1019–1031.

[24] Hao Ma, Haixuan Yang, Michael R. Lyu, and Irwin King. 2008. SoRec: Social Recommendation Using Probabilistic Matrix Factorization. In *CIKM*. ACM, New York, NY, USA, 931–940.

[25] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *NIPS*. 1257–1264.

[26] Roberto Rigamonti, Matthew A Brown, et al. 2011. Are sparse representations really relevant for image classification?. In *CVPR*. IEEE, 1545–1552.

[27] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*. IEEE, 6655–6659.

[28] Ruslan Salakhutdinov and Andriy Mnih. 2008. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *ICML*. ACM, 880–887.

[29] Salvatore Scellato, Anastasios Noulas, et al. 2011. Exploiting place features in link prediction on location-based social networks. In *KDD*. ACM, 1046–1054.

[30] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *WWW*. ACM, 111–112.

[31] Ting-Yi Shih, Ting-Chang Hou, Jian-De Jiang, Yen-Chieh Lien, Chia-Rui Lin, and Pu-Jen Cheng. 2016. Dynamically Integrating Item Exposure with Rating Prediction in Collaborative Filtering. In *SIGIR*. ACM, 813–816.

[32] Hidetoshi Shimodaira. 2000. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference* 90, 2 (2000), 227–244.

[33] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. 2016. Multi-rate deep learning for temporal recommendation. In *SIGIR*. ACM, 909–912.

[34] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *NIPS*. 2643–2651.

[35] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. 2017. Relational Deep Learning: A Deep Latent Variable Model for Link Prediction.. In *AAAI*. 2688–2694.

[36] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *KDD*. ACM, 1235–1244.

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

[38] Xian Wu, Yuxiao Dong, Chao Huang, Jian Xu, Dong Wang, and Nitesh V Chawla. 2017. UAPD: Predicting Urban Anomalies from Spatial-Temporal Data. In *ECML/PKDD*. Springer, 622–638.

[39] Yao Wu, Christopher DuBois, Alice X Zheng, et al. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*. ACM, 153–162.

[40] Liang Xiong, Xi Chen, et al. 2010. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM*. SIAM, 211–222.

[41] Xuchao Zhang, Liang Zhao, Arnold P Boedihardjo, Chang-Tien Lu, and Naren Ramakrishnan. 2017. Spatiotemporal Event Forecasting from Incomplete Hyper-local Price Data. In *CIKM*. ACM, 507–516.