All You Need is Tensor Decomposition:
A Better Understanding of Sparse Tensors

by

Bo Hui

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 6, 2023

Keywords: Tensor decomposition, hyperbolic space, ODE

Approved by

Wei-Shinn Ku, Chair, Professor of Computer Science and Software Engineering
Cheryl Seals, Professor of Computer Science and Software Engineering
Anh Nguyen, Assistant Professor of Computer Science and Software Engineering
Shubhra Kanti Karmaker, Assistant Professor of Computer Science and Software Engineering
Yin Sun, Assistant Professor of Electrical and Computer Engineering

Abstract

Tensor is widely employed in data sciences to represent multi-dimensional information. Due to the low-rank nature of many tensors in the real world, predicting the unobserved entries of a partially observed tensor has gained much interest in many applications such as knowledge base completion and recommendation. Tensor decomposition is a widely-used method to solve these problems. However, existing works decompose the tensor into Euclidean vectors and assume a multilinearity relationship between tensor entries. In real applications, the input tensors tend to exhibit complex factor interactions. We propose to model these complex interactions for a better understanding of the sparse tensors.

We first study if the side information can be used to improve the performance of tensor decomposition. Specifically, we design a neural tensor model to estimate the values in a user-item-time tensor. A regularization loss head based on a novel social Hausdorff distance function is designed to optimize the reconstructed tensor. Despite the success of neural tensor models which is defined in Euclidean space, recent works shows that hyperbolic space is roomier than Euclidean space. To leverage the power of hyperbolic vector, we propose to decompose tensor in hyperbolic space instead of Euclidean space. Considering that the most popular optimization tools such as SGD (Stochastic Gradient Descent) have not been generalized in hyperbolic space, we have designed an adaptive optimization algorithm according to the distinctive property of hyperbolic manifold. In addition, we raise a new question: can we model the interaction between latent factors with neural ODEs (Ordinary Differential Equations)? Studying this research problem is particularly interesting since we can parameterize the derivative of the latent factors using a neural network. We design a neural ODEs tensor model to optimize the decomposed factors. Concretely, we aggregate the decomposed factors and feed it into a Neural ODEs model to reconstruct the input tensor. An ODE solver is introduced to minimize the difference between the original tensor and reconstructed tensor. We experiment with multiple real world datasets spanning diverse domains to demonstrate the effectiveness of the proposed methods.

Acknowledgments

This research has been funded in part by the U.S. National Science Foundation grants IIS-1618669 (III) and ACI-1642133 (CICI).

I would like to thank Dr. Wei-Shinn Ku for his invaluable supervision over the last five years. At the beginning of my Ph.D. program, I barely had any experience in the domain of data mining. Dr. Ku had a weekly meeting with me where I learned how to read research papers efficiently and think about the research problem at a high level. In addition, Dr. Ku showed me a path to becoming a qualified faculty. His support and encouragement have made everything I have undertaken in research possible. Dr. Ku has set up a model for my future faculty career.

I would also like to thank the committee members: Dr. Cheryl Seals, Dr. Anh Nguyen and Dr. Shubhra Kanti Karmaker for their valuable comments on my projects and dissertation. I really appreciate Dr. Yin Sun for reviewing my dissertation and for detailed feedback.

I am grateful to everyone I have collaborated with. At last, I would like to thank my lab mates, my friends and my family for their unconditional support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tensor is a natural higher-order generalization of matrix and is widely employed in data sciences to represent multi-dimensional objects or information. For example, knowledge graphs can be treated as partially observed third-order binary tensors where the value of each entry represents the existence of a relationship between two entities; a three-dimensional user-item-time tensor can represent the shopping events in the recommender systems. Due to the low-rank nature of many tensors in real applications, predicting the unobserved entries of a partially observed tensor has gained much interest in many machine learning applications such as knowledge base completion [78, 45], recommendation [22, 16], spatiotemporal analysis [77], health data analysis [33, 30], to name just a few.

Tensor Decomposition is a fundamental framework to solve these problems by decomposing the input tensor into a few low-rank embedding matrices. Then the value of an unknown entry can be estimated by a sequence of elementary operations on the decomposed matrices. CP (CANDECOMP/PARAFAC) [28] and Tucker decomposition [69] are the two conventional tensor factorization methods. Both CP and Tucker decompose the partially observed tensor into latent factors through multilinear multiplication. Based on CP and Tucker, many tensor decomposition algorithms have been developed, such as PARAFAC2 [27] and P-TUCKER [55].

However, in real applications, the input tensors tend to exhibit complex non-linear factor interactions and many recent works [31, 31] have shown that nonlinear factorization models have superior performance over linear models. With the prosperity of deep learning, many neural tensor models have been developed to leverage neural networks to solve the problem. For example, CoSTCo [47] utilizes the expressive power of convolutional neural network to

model the complex interactions inside tensors and NTM [16] combines neural networks and tensor algebra to capture nonlinear interactions among multi-aspect factors.

Despite the success of these neural tensor models, these methods have not utilized the side information which can further improve the overall performance. For example, in recommender systems, two users tend to interact with the same items according to the social homophily theory. Therefore, the social network can be used as side information of a user-item-time tensor. Also, existing works have not considered the latent structure of entries. Recent studies have found that many real-world tensor can be well described by a framework with an underlying non-Euclidean hyperbolic geometry. It is intuitive to investigate tensor decomposition in hyperbolic space. A challenge is that all operations in Euclidean space such as vector addition, matrix-vector multiplication, and vector inner product are defined in Euclidean space. Therefore, existing tensor decomposition methods including neural network models are not applicable anymore for hyperbolic vectors. Lastly, while neural ordinary differential equations (ODE) have been used to parameterize the derivative of the hidden state in various research problems, the gap between neural ODE and tensor decomposition has not been filled.

To this end, we have finished three projects, each with the aforementioned motivations regarding tensor decomposition. (1) In the first work, we investigate if the side information can be used for tensor completion in POI recommendation. Specifically, we propose a regularization loss head based on a novel *social Hausdorff distance* function to optimize the reconstructed tensor. To address the sensitivity of negative sampling, we train the model on the whole data by treating all unlabeled entries in the observed tensor as negative, and rewriting the loss function in a smart way to reduce the computational cost. Empirical results demonstrate the superiority of our model over state-of-the-art tensor completion methods. The details are discussed in Chapter 3. (2) Our second work studies tensor decomposition in hyperbolic space. With the same dimension, a hyperbolic vector can represent richer information (e.g., hierarchical structure) than a Euclidean vector. Considering that the most popular optimization tools (e.g, SGD) have not been generalized in hyperbolic space, we design an adaptive optimization algorithm according to the distinctive property of hyperbolic manifold. To address the non-convex

property of the problem, we adopt gradient ascent in our optimization algorithm to avoid getting trapped in local optimal landscapes. Our extensive experiments validate the superiority of our method over these baselines that solve the problem in Euclidean space. (3) In our third work, we propose to parameterize the derivative of the latent factors using a neural network. By learning a differential equation solver based on continuous-depth residual networks, we can model the deeply hidden relations between latent factors. With this intuition, we have filled the gap between neural ordinary differential equations and tensor decomposition. Concretely, we aggregate the latent factors and feed it into a Neural ODEs model to reconstruct the input tensor. A ODE solver is introduced to minimize the difference between the original tensor and reconstructed tensor. We empirically demonstrate the superiority of the Neural ODE over these methods without modeling the derivatives in tensor decomposition tasks.

Our key contributions are:

1. We propose a regularization loss head based on a novel social Hausdorff distance function to leverage side information in tensor decomposition.

2. To the best of the author's knowledge, our paper is the first work that studies the tensor decomposition problem in hyperbolic space.

3. We fill the gap between neural ODEs and tensor decomposition.

4. We experiment with multiple real-world datasets spanning diverse domains to demonstrate the effectiveness of the proposed methods

Chapter 2

Related works

In this section, we review the formulations of tensor completion, and recent related works on neural tensor completion models.

Tensor Factorization and Completion. Tensor as a data structure has been widely employed in the database and data mining community to represent multi-dimensional objects. In many application scenarios (e.g., recommender systems, knowledge base completion, and temporal health data analysis), only a sample of tensor entries are revealed to users, and the goal is to infer the values of all missing entries. A common prior knowledge leveraged for tensor completion is the low-rank tensor structure [7], which assumes that the unknown ground-truth tensor can be factorized into a few low-rank embedding matrices. CP (CANDECOMP/PARAFAC) [28] and Tucker decomposition [69] are two classical tensor factorization models, which can be considered as higher order generalizations of the matrix decomposition (e.g., singular value decomposition, or SVD). Without loss of generality, we consider 3D tensors. Formally, a rank-r CP model factorizes an order-3 tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ into three factor matrices: $\mathbf{U}^1 \in \mathbb{R}^{r \times I}$, $\mathbf{U}^2 \in \mathbb{R}^{r \times J}$ and $\mathbf{U}^3 \in \mathbb{R}^{r \times K}$, such that a tensor entry can be predicted as:

$$\hat{\mathcal{X}}_{i,j,k} = \sum_{t=1}^{r} {\mathbf{U}^1}_{t,i} {\mathbf{U}^2}_{t,j} {\mathbf{U}^3}_{t,k}. \tag{2.1}$$

In contrast, Tucker decomposition factorizes a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ into a core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ and three factor matrices such that:

$$\hat{\mathcal{X}}_{i,j,k} = \sum_{t_1=1}^{r_1} \sum_{t_2=1}^{r_2} \sum_{t_3=1}^{r_3} \mathcal{G}_{t_1,t_2,t_3} {\mathbf{U}^1}_{t_1,i} {\mathbf{U}^2}_{t_2,j} {\mathbf{U}^3}_{t_3,k}. \tag{2.2}$$

Based on CP and Tucker decomposition, many low-rank tensor factorization algorithms were developed. INDSCAL [9] is a special case of CP for three-way tensors that are symmetric in two modes. PARAFAC2 [27] is a variant of CP that can be applied to a collection of matrices. CANDELINC [10] is a CP variant with linear constraints. PARATUCK2 [29] groups the mode-1 objects and the mode-2 group into latent components.

Scalable Tensor Factorization and Completion. Among recent works, P-TUCKER [55] is a scalable Tucker factorization method for sparse tensors. SPALS [19] is a method to sample intermediate steps of alternating minimization algorithms for computing low-rank tensor CP decompositions. MAST [65] tracks the subspace of general incremental tensors for completion. GigaTensor [40] designs a scalable distributed algorithm for large-scale tensor decomposition. D-Tucker [39] compresses the tensor by computing randomized SVD.

To reconstruct a ground-truth tensor $\mathcal{X}$ based on a set, Ω, of observed entries at hand, the strategy of most existing works is to resort to the following least-squares formulation:

$$\text{minimize } f(\mathbf{U}^1, \mathbf{U}^2, \mathbf{U}^3) := \sum_{(i,j,k)\in\Omega} (\hat{\mathcal{X}}_{i,j,k} - \mathcal{X}_{i,j,k})^2. \tag{2.3}$$

Due to its highly nonconvex nature, however, solving this optimization problem exactly is computationally intractable. A line of polynomial-time approximation algorithms have been proposed, such as convex relaxation [63], sum of squares hierarchy [58], alternating minimization [38], vanilla gradient descent [7], and structure-aware proximal iterations [75].

Nonlinear and Neural Tensor Completion. While the above-mentioned works assume a multilinearity relationship between latent factors, recent works favor nonlinear factorization models to consider complex interactions in the real world. For example, in [31], the author introduces a kernel-based CP model to capture nonlinear relationships, and in [22], a Gaussian radial basis function is used for nonlinear tensor completion.

More recently, some works have proposed replacing the multilinear operations in tensor completion with multi-layer perceptrons (MLP), to utilize the non-linear activation layer of neural network models. CoSTCo [47] leverages the power of convolutional neural networks to

model the interactions inside tensors. NTF [41] combines generalized CP and tensorized MLP to compute the tensor.

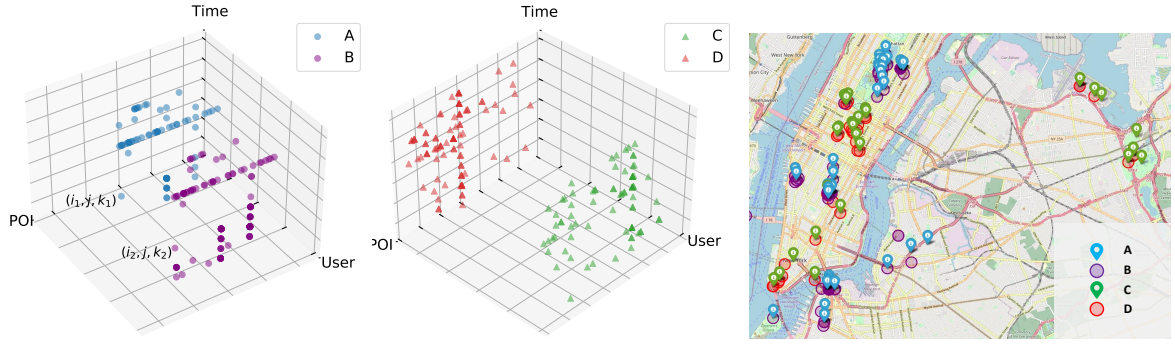
Chapter 3

Tensor Completion with Side Information for POI Recommendation

3.1 Motivation

Context has been recognized as an important factor to consider in personalized recommender systems [41]. With the prosperity of location-based social network (LBSN) services such as Foursquare and Yelp, a large number of check-ins have been collected by the LBSN companies and utilized for POI recommendations. The time dimension has always been a key context factor in such recommendations [47], since users' visiting preferences are highly time sensitive. For example, the season plays an important factor in where people visit; hot pot restaurants are the most crowded in the winter season, while holiday hotspots can transition from aquatics centers in summer to ski resorts in winter.

Since such time-awareness cannot be captured by a conventional collaborative filtering approach such as low-rank matrix completion over a low-rank user-item rating/purchase matrix, we adopt a three-dimensional user-POI-time tensor to effectively represent the check-in events. Figure 3.1(a) (resp. Figure 3.1(b)) shows the check-ins by User A and User B (resp. User C and User D) who are friends, in an order-3 user-POI-time tensor. The check-ins are obtained



(a) An order-3 tensor (b) An order-3 tensor (c) User friendship pairs (A-B and C-D) and their visited POIs

Figure 3.1: Low-Rank Tensor Completion with Time and Social-Spatial Context

from the Gowalla dataset used in our experiments. We can observe that friends tend to visit similar POIs, such as the two markers at (i_1, j, k_1) and (i_2, j, k_2) which indicate that both A and B visited POI j albeit at different time slots k_1 and k_2, respectively.

Tensor completion is a popular approach to fill the missing entries of a partially observed tensor in recommender systems [65], based on the fact that tensors in real applications often exhibit a low-rank property. The approach reconstructs a partially observed tensor through multilinear multiplication of latent factors, such as CP (CANDECOMP/PARAFAC) and Tucker factorizations [44]. However, real-world tensors in recommender systems often exhibit complex non-linear factor interactions, so recent works CoSTCo [47] and NTM [16] combine deep neural networks with tensor algebra in order to capture the nonlinear interactions among the tensor factors.

While these works can utilize the time dimension as a factor, they do not utilize the social-spatial information present in LBSNs, which is a very important factor when users decide where to visit. For example, we tend to turn to our friends for recommendations of nice restaurants, shopping stores and/or maintenance/repair services in our daily life, which aligns well with he social homophily theory [52]. Moreover, people tend to visit places that they are familiar with most of the time, so the visited POIs tend to form localized clusters which aligns well with Tobler's first law of geography [67]. As a result of these observations, we can conclude that two users tend to check in the same or nearby places if they are friends.

We have verified this observation using real data and we found it to be generally true and robust. As an illustration, Figure 3.1(c) plots the check-in locations of the two pairs of friends (A-B and C-D) on a map. We can see that the check-in locations of Users A and B (blue markers and purple circles) are highly overlapped, and so are the check-in locations of Users C and D (green markers and red circles). This shows that friends' check-ins tend to colocate more than non-friends' check-ins (e.g., those of Users A and C).

Social network information is proven to benefit the performance of recommendation [51] and the locations information can also improve the performance of POI recommendation [34]. In this paper, we aim to leverage the social-spatial information in LBSN as the side information to improve neural tensor completion based POI recommendation.

Our neural tensor completion model associates each user, POI and time unit with an embedding vector to learn, so that the value for each entry (i, j, t) in the tensor can be recovered from the embedding vectors of User i, POI j and time unit t. Most existing tensor completion models initialize the embeddings randomly or with one-hot encoding, especially when the content information of users and items (POI in our case) is not available due to privacy reasons. However, careful initialization is critical in various tensor problems, particularly for neural tensor models where gradient descent could get trapped in undesirable stationary points (e.g., a saddle point) if it starts from an arbitrary point. We, therefore, adopt a spectral method to obtain rough estimates of the initial user, POI and time embeddings for embedding initialization.

Our neural tensor completion model is trained to minimize the least-squares error between the reconstructed tensor and the original tensor. Generally, only positive data (check-ins) are observed in a POI recommender system. To train the model additionally with negative data, existing deep learning methods use the strategy of negative sampling to generate the negative samples, the performance of which is highly sensitive to the sampling strategy and the number of negative samples [14]. In particular, negative sampling is a biased approximation and often does not converge to the same loss as computed from all entries, making it difficult to converge to the optimal ranking performance regardless of how many update steps have been taken. Inspired by [15] which solves matrix completion without negative sampling, we train our neural tensor model with the whole data including all unlabeled data to combat the drawback of negative sampling; to address the high computational cost of directly computing a least-squares loss over all the tensor entries, we rewrite this loss function in a smart way to allow more efficient calculation and learning.

A key contribution of our model design is a backpropagatable formulation of the social Hausdorff distance function. The Hausdorff distance metric measures how far two sets of points are from each other. In our model design, we want to minimize the Hausdorff distance between two sets of check-in locations from each pair of friends to enforce the social homophily theory, but several challenges exist. Firstly, the inputs to the original Hausdorff distance metric are locations, not the outputs of our neural tensor model (i.e., the probability of each user-POI-time interaction). Secondly, even if we revise our social Hausdorff distance function to admit

user-POI-time interaction probabilities, the minimization operator $min(\cdot)$ in the original Hausdorff distance function is not a smooth function with respect to its inputs, so it does not support backpropagation. We, therefore, need to modify this operator to support learning with backpropagation. The proposed social Hausdorff distance function enables our model to integrate both the social relations and POI locations to regularize the tensor completion formulation. To boost the diversity of recommendation, we leverage location entropy [20] to allow those less frequently visited POIs to be recommended, considering that they often better reveal the real user social strength than those places that everyone visits.

We hereby summarize our major contributions as follows:

- We present a backpropagatable formulation of the social Hausdorff distance-based loss function which enables our model to leverage both social graph and POI locations to regularize the tensor completion formulation.
- We integrate the social-spatial loss head with location entropy POI weights to improve recommendation diversity.
- We train our model with the whole data including all unlabeded data to combat the drawback of negative sampling in tensor completion, and rewrite the least-squares loss smartly to reduce the time complexity.
- We carefully initialize the latent user/POI/time embeddings with a spectral method to avoid being trapped in stationary points during training.

3.2 Related Works to POI Recommendation

The problem of POI recommendation aims to predict where a user will visit next. The most common approach for recommendation is known as collaborative filtering[73]. Matrix completion [61] is used as a popular collaborative filter algorithm. These methods decompose the uer-POI matrix into two smaller matrices to discover the potential relationship between users and POIs. While the majority of matrix completion models apply an inner product on the latent factors, recent works tend to replace it with more complicated operation, such as

neural network architecture [32]. Content-based filtering is another common approach for recommendation. Content-based recommender systems associate each user or POI with content information, such as location contexts [46] and spatial topics [34]. However, the performance of content-based methods relies on the quality of user and item features, and the content information is not always available due to privacy concerns. Graph neural networks [43, 37, 35] has also been used for recommendation [36]. TenInt [74] also formulates the recommendation as the tensor completion problem. There are several differences between TenInt and our model. First, TenInt does not integrate the spatial information (i.e., the distances between POIs) into tensor completion. It solves the problem by minimizing the squared loss regularized by the difference of user factors between each pair of friends. Thus, spatial information is not involved in their problem formulation. In contrast, our solution is able to leverage the spatial information. Instead of simply minimizing the difference of user factors between each pair of friends, we minimize the Hausdorff distance between check-ins of two users with friendship relations. Moreover, TenInt simply employs CP decomposition, while our model is a nonlinear factorization model that captures the complex interactions in the real world.

As a separate line of research different from the tensor completion formulation, many spatial-temporal models [79] have been developed to leverage both the location information and the temporal order of check-ins.

3.3 Preliminary

3.3.1 Problem Formulation

We use $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ to denote the check-in tensor of an LBSN, where I, J and K refer to the number of users, POIs, and time units (aka. intervals), respectively. The set $\Omega = \{(i,j,k) \,|\, \forall i \in \{1,\cdots,I\}, \forall j \in \{1,\cdots,J\}, \forall k \in \{1,\cdots,K\}\}$ contains indexes to all entries in $\mathcal{X}$. If i^{th} user checks in j^{th} POI at time interval k, then $\mathcal{X}_{i,j,k} = 1$ ($\mathcal{X}_{i,j,k}$ is an observed entry), otherwise $\mathcal{X}_{i,j,k} = 0$ (unlabeled). Let $\Omega_+ = \{(i,j,k) | \mathcal{X}_{i,j,k} = 1\}$ and $\Omega_- = \{(i,j,k) | \mathcal{X}_{i,j,k} = 0\}$ be the set of positive entries and unlabeled entries, respectively.

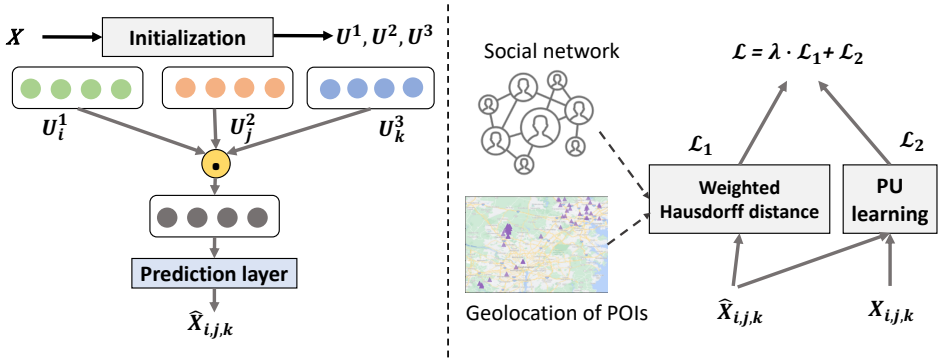


Figure 3.2: The Model Framework

Given a social graph $G = (V, E)$, each node v_i corresponds to an LBSN user. If two users v_i and $v_{i'}$ are friends, there will be an edge $e_{i,i'}$ between v_i and $v_{i'}$. We use l_j to denote the location of j^{th} POI, which is a tuple of longitude and latitude. Then the problem is formulated as reconstructing tensor $\mathcal{X}$ as $\hat{\mathcal{X}}$ with a social graph $G = (V, E)$ and geolocations of POIs $\{l_j | \forall j \in \{1, \cdots, J\}\}$. The goal is to estimate the values of the unobserved entries $\hat{\mathcal{X}}_{i,j,k}$ as the scores among User i, POI j, and time interval k to recommend high-score entries.

3.3.2 Model Framework

Figure 3.2 overviews the framework of our model. Specifically, we first associate each user, POI and time interval with an embedding by a spectral initialization method. Then, the value of each entry is learned from corresponding embeddings by a neural network. The key novelty of our model is to use a hybrid loss function with two heads: (1) a weighted Hausdorff distance head to leverage the social-spatial information; and (2) the mean-squared error between the original tensor and reconstructed tensor on the whole data. We use the strategy of joint training to optimize the reconstructed tensor.

3.4 Methodology

3.4.1 Embedding Initialization

Tensor factorization models usually use gradient descent (GD) as the optimization algorithm. However, GD could get trapped in undesirable stationary points (e.g., saddle point) if it starts from arbitrary points. Therefore, careful initialization is important in tensor factorization, and often necessary in order to achieve fast convergence [18]. In this paper, we use a spectral

method to obtain rough estimates of the initial user, POI and time unit embeddings. Specifically, we first "unfold" the tensor $\mathcal{X}$ into three subspace matrices: the mode-1, mode-2 and mode-3 matricizations $\mathbf{A} \in \mathbb{R}^{I\times(JK)}, \mathbf{B} \in \mathbb{R}^{J\times(IK)}$ and $\mathbf{C} \in \mathbb{R}^{K\times(IJ)}$, respectively, where $\mathbf{A}_{i,(j-1)K+k} = \mathcal{X}_{i,j,k}$, $\mathbf{B}_{j,(i-1)K+k} = \mathcal{X}_{i,j,k}$, and $\mathbf{C}_{k,(i-1)I+j} = \mathcal{X}_{i,j,k}$. To estimate the rank-$r$ factors, a natural choice is to explore the principal subspace of $\mathbf{A}\mathbf{A}^T$, $\mathbf{B}\mathbf{B}^T$ and $\mathbf{C}\mathbf{C}^T$. Specifically, we zero out the diagonal entries of the three matrices, and then use the top-r eigenvectors of all off-diagonal matrices as the estimated factors in each subspace:

$$\begin{aligned}
\mathbf{U}^1 &= eigen((\mathbf{A}\mathbf{A}^T)|_{\text{off-diag}}, r) \in \mathbb{R}^{I\times r} \\
\mathbf{U}^2 &= eigen((\mathbf{B}\mathbf{B}^T)|_{\text{off-diag}}, r) \in \mathbb{R}^{J\times r} \\
\mathbf{U}^3 &= eigen((\mathbf{C}\mathbf{C}^T)|_{\text{off-diag}}, r) \in \mathbb{R}^{K\times r},
\end{aligned} \tag{3.1}$$

where $|_{\text{off-diag}}$ extracts out the off-diagonal entries of a squared matrix, and $eigen(., r)$ extracts the top-r eigenvectors of the input matrix as the r columns of the output matrix. We zero out all diagonal entries because the diagonal entries bear too much influence on the principal directions and should be down-weighed [6]. Intuitively, we conduct PCA along one mode, by treating the values in the other two modes as features. The benefit of this initialization method is that the roughly estimated tensor factors can result in fast convergence, and we shall justify this in our experiments.

3.4.2 Tensor Factorization Formulation

We model the ternary interactions among users, POIs and time intervals, by computing the value of each entry $\mathcal{X}_{i,j,k}$ with the corresponding embedding vectors. Specifically, given the embedding vectors $\mathbf{U}^1_i$, $\mathbf{U}^2_j$ and $\mathbf{U}^3_k$ for User i, POI j and time interval k, we first compute a vector that equals their element-wise product:

$$\phi(\mathbf{U}^1_i, \mathbf{U}^2_j, \mathbf{U}^3_k) = \mathbf{U}^1_i \odot \mathbf{U}^2_j \odot \mathbf{U}^3_k, \tag{3.2}$$

where $\odot$ denotes the element-wise product of vectors. Note the similarity between Eq (3.2) and the CP formulation in Eq (2.1). To predict the value of $\mathcal{X}_{i,j,k}$, we pass the vector $\phi(\mathbf{U}^1_i, \mathbf{U}^2_j, \mathbf{U}^3_k)$

through a dense layer with parameter $\mathbf{h} \in \mathbb{R}^r$:

$$\hat{\mathcal{X}}_{i,j,k} = \mathbf{h}^T(\mathbf{U}^1_i \odot \mathbf{U}^2_j \odot \mathbf{U}^3_k) = \sum_{t=1}^{r} \mathbf{h}_t \mathbf{U}^1_{i,t} \mathbf{U}^2_{j,t} \mathbf{U}^3_{k,t}, \tag{3.3}$$

where $\mathbf{h}_t$ corresponds to the importance weight of the t^{th} factor dimension. Semantically, we treat $\hat{\mathcal{X}}_{i,j,k}$ as the probability that $X_{i,j,k} = 1$, i.e., User i visits POI j in time interval k.

We remark that the CP model is a special case of our formulation. Specifically, assume that $\mathbf{h}$ is a vector filled with all ones (i.e., $\mathbf{h} = [1, 1, ..., 1]^T \in \mathbb{R}^r$). Then Eq (3.3) becomes $\sum_{t=1}^{r} \mathbf{U}^1{}_{t,i} \mathbf{U}^2{}_{t,j} \mathbf{U}^3{}_{t,k}$, which is exactly Eq (2.1). Compared with CP, our model with learnable parameter $\mathbf{h}$ is more expressive and can capture more complicated interactions of multiple factors in recommender system.

To optimize the embeddings $\mathbf{U}^1$, $\mathbf{U}^2$, $\mathbf{U}^3$ and parameter $\mathbf{h}$, we may simply minimize the squared error $(\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2$ for observed entries. However this error head alone does not utilize the rich social-spatial information in an LBSN. We, therefore, design a hybrid loss function with two loss heads: one is the prediction error head computed over observed entries, and the other is a novel *social Hausdorff distance* function which measures how far friends' subsets of POIs are from each other, which we introduce next.

3.4.3 Social Hausdorff Distance

In our daily life, people frequently turn to their friends for recommendations of nice restaurants and other places to visit. According the social homophily theory, social networks tend to form clusters of nodes with similar properties or interests. In the meanwhile, Tobler's first law of geography states that near things are more related than distant things, implying that, for example, a user is more likely to have dinner in a restaurant visited and liked by his friends; moreover, if the user decides to go shopping after the dinner, he/she is more likely to check in at a nearby shopping mall.

Let us denote the i^{th} user's vertex in an LBSN by v_i, and we use i and v_i interchangeably in the following discussion. Formally, if a user $v_{i'}$ is a friend of another user v_i in an LBSN, then the set of $v_{i'}$'s check-ins tend to overlap with or at least close to v_i's check-ins. Therefore,

it is natural to recommend a POI to a user if it is visited by his/her friends, or close to a POI visited by his/her friends. We thus add a regularization term to the loss function to enforce such a social-spatial cluster structure in our reconstructed tensor.

Hausdorff distance is metric to measure how far two point sets are from each other. Let us denote by $S(v_i)$ the potential POIs that will be visited by a user v_i, and denote by $\mathcal{N}(v_i)$ the set of POIs that were checked by v_i's friends:

$$S(v_i)=\{j \mid \exists k \in \{1, \cdots K\}, \hat{\mathcal{X}}_{i,j,k} > 0\}, \tag{3.4}$$

$$\mathcal{N}(v_i)=\{j \mid \forall (v_i, v_{i'}) \in E\colon \exists k \in \{1, \cdots K\}, \mathcal{X}_{i',j,k} = 1\} \tag{3.5}$$

Then, the average Hausdorff distance [3] (AHD) between $S(v_i)$ and $\mathcal{N}(v_i)$ is formulated as:

$$\begin{aligned} d_{AH}(S(v_i), \mathcal{N}(v_i)) = & \frac{1}{|S(v_i)|} \sum_{j \in S(v_i)} \min_{j' \in \mathcal{N}(v_i)} d(j, j') \\ & + \frac{1}{|\mathcal{N}(v_i)|} \sum_{j' \in \mathcal{N}(v_i)} \min_{j \in S(v_i)} d(j, j'), \end{aligned} \tag{3.6}$$

where $d(j, j')$ denotes the distance between POIs j and j'.

Intuitively, the AHD formulation above calibrates on average how far each point in $S(v_i)$ is to its closest point in $\mathcal{N}(v_i)$, and vice versa to make the distance measure symmetric.

Figure 3.3 illustrates AHD calculation. Specifically, the dashed pink circles represent those potential POIs that user v_i may be interested in and the yellow circles are the ground-truth POIs that are checked in by v_i's friends. For each dashed pink node $j \in S(v_i)$, $\min_{j' \in \mathcal{N}(v_i)} d(j, j')$ is the distance between j and its nearest neighbor in $\mathcal{N}(v_i)$. The first term in Eq (3.6) averages such distances for all $j \in S(v_i)$. Likewise, the second term averages such minimum distances for all $j' \in \mathcal{N}(v_i)$.

To regularize our loss function with social homophily and Tobler's first law of geography, we want to minimize the AHD between $S(v_i)$ and $\mathcal{N}(v_i)$. However, there are two challenges. First, recall from Eq (3.3) that our neural tensor model estimates the probability of each user-POI-time interaction, not the locations of POIs. The function must be learnable with respect to the output of the neural tensor model. The potential locations of $S(v_i)$ could be as large as the

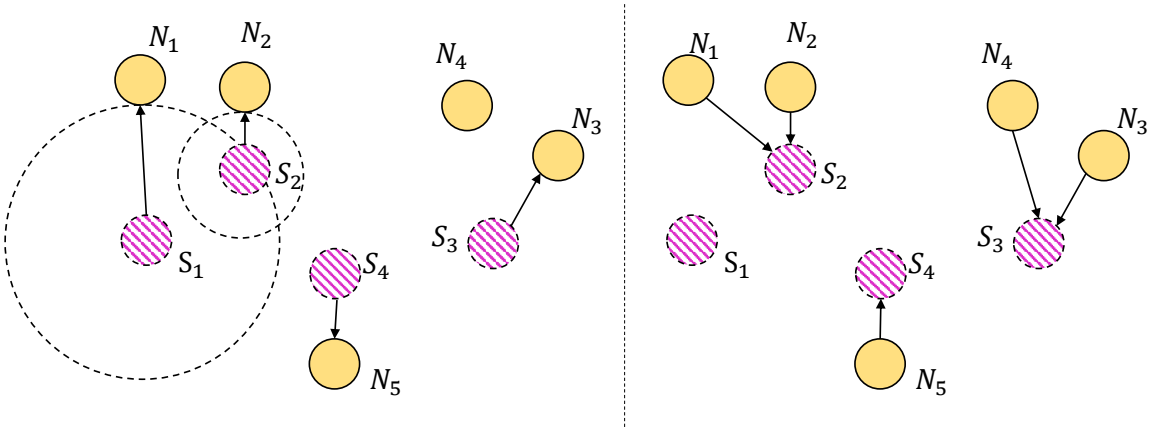


Figure 3.3: Average Hausdorff Distance (Arrows Highlight Nearest Neighbors)

entire set of POIs in an LBSN, and we need to generalize the AHD definition to allow $S(v_i)$ to be set of locations with occurrence probabilities. Second, the minimization function $\min(\cdot)$ is not a smooth function with respect to its inputs, so it does not support backpropagation.

To address these two challenges, we devise a weighted Hausdorff loss function counterpart based on the idea of AHD:

$$\begin{aligned} d_{WH}(S(v_i), \mathcal{N}(v_i)) = & \frac{1}{|A+\epsilon|} \sum_{j \in S(v_i)} p_{i,j} \min_{j' \in \mathcal{N}(v_i)} d(j, j') \\ & + \frac{1}{|\mathcal{N}(v_i)|} \sum_{j' \in N(v_i)} \underset{j \in S(v_i)}{M_\alpha} [p_{i,j} d(j, j') + (1 - p_{i,j}) d_{max}], \end{aligned} \tag{3.7}$$

where $p_{i,j} = 1 - \prod_{k=0}^{K} (1 - \hat{\mathcal{X}}_{i,j,k})$ estimates the probability that User v_i will check in the j^{th} POI (in any of the K time intervals we consider). In the first term of Eq (3.7), $A = \sum_{j \in S(v_i)} p_{i,j}$ is the normalization factor for taking the weighted distance average (where weights are given by probabilities $p_{i,j}$), and the value of ϵ is set as 10^{-6} to avoid division by zero. The second term of Eq (3.7) approximates the second term of Eq (3.6), where $\min_{j \in S(v_i)} d(j, j')$ is now approximated by $\underset{j \in S(v_i)}{M_\alpha} [p_{i,j} d(j, j') + (1 - p_{i,j}) d_{max}]$ which allows the loss to backpropagate through $M_\alpha[.]$ into $p_{i,j}$ to tune the parameters in Eq (3.3), as we shall discuss in the next paragraph. In contrast, we do not apply $M_\alpha[\cdot]$ in the first term since $p_{i,j}$ (and hence $\hat{\mathcal{X}}_{i,j,k}$) is outside the $\min(.)$ function.

Here, $M_\alpha[x_1, \ldots, x_n] = (\frac{1}{n} \sum_{i=1}^{n} x_i^\alpha)^{\frac{1}{\alpha}}$ is the generalized mean, which equals $\min\{x_1, \ldots, x_n\}$ when $\alpha \to -\infty$. However, the more negative α is, the less smooth M_α becomes, and existing

research found that $\alpha = -1$ is already sufficient to strike a good balance between its approximation quality to $\min(.)$ and the smoothness for effective backpropagation [62].

With the first term of Eq (3.7), we multiply the distance by $p_{i,j}$ to penalize the POI j with larger value of $p_{i,j}$ where there is no nearby POI checked by friends. In other words, if POI j is far from any of POIs that are checked by the friends of user v_i, $p_{i,j}$ tends to be 0.

Also in the second term of Eq (3.7), d_{max} corresponds to the maximum distance between any two POIs and it varies in different datasets. When we denote $f(j) = p_{i,j}d(j,j') + (1 - p_{i,j})d_{max}$, then we can see that the second term of Eq (3.7) approximates $\min_{j \in S(v_i)} d(j,j')$ with $\underset{j \in S(v_i)}{M_\alpha}[f(j)]$. Note that when $p_{i,j} \to 1$, $f(j) \to d(j,j')$ as desired; while when $p_{i,j} \to 0$, $f(j) \to d_{max}$ so the "soft" minimum $M_\alpha[.]$ tends to ignore j even if $d(j,j')$ is small, which makes sense since User v_i is unlikely to visit POI j. In other words, POIs with a low visit-probability by v_i near "friend" POIs $j' \in \mathcal{N}(v_i)$ will be penalized. While $f(.)$ is not the only function that enforces $f|_{p_{i,j}=1} = d(j,j')$ and $f|_{p_{i,j}=0} = d_{max}$, this linear function form is favored due to its numerical stability.

Note that in the deterministic scenario where $p_{i,j}$ is either 1 or 0, Eq (3.7) will become Eq (3.6) if we regard $M_\alpha[.]$ as $\min(.)$. Therefore, our loss function can be interpreted as average Hausdorff distance (AHD) extended with input uncertainty.

We remark that both terms in Eq (3.7) are necessary or we will get extreme results. Specifically, if the first term is removed, then $p_{i,j} = 1$ will always optimize $d_{WH}(S(v_i), \mathcal{N}(v_i))$ since $d(j,j') \leq d_{max}$. While if the second term is removed, $p_{i,j} = 0$ will always optimize $d_{WH}(S(v_i), \mathcal{N}(v_i))$ to be 0.

Diversifying Recommendation by Location Entropy. Recommending popular POIs already known by most users does not provide much additional information. Ideally, a user v_i would like to see recommended POIs that well match v_i's current interest but are not already visited by many people (but rather known by only a few who frequently pay visits). We measure the popularity of a POI using the concept of location entropy. Specifically, let $\Phi_{i,j} = \{\langle i,j,k \rangle \,|\mathcal{X}_{i,j,k} = 1\}$ be the set of all check-ins at POI j by a user i, and let $\Phi_j = \{\langle i,j,k \rangle \,|\mathcal{X}_{i,j,k} = 1\}$ be the set of check-ins at POI j by all users. Then the location entropy is defined as:

$$\mathcal{E}_j = -\sum_{i:\|\Phi_{i,j}|>0} \frac{|\Phi_{i,j}|}{|\Phi_j|} \log \frac{|\Phi_{i,j}|}{|\Phi_j|}. \tag{3.8}$$

A high value of $\mathcal{E}_j$ implies that POI j is visited by different users and thus known by many users, such as a local Costco wholesale warehouse. In this case, POI j plays a less important role in reflecting the social strength: two users who visit the same Costco warehouse are mostly likely not mutual friends; while in contrast, if two users show up in the same tennis court, they are more likely to be friends sharing the same hobby.

We thus further adjust $d(j, j')$ in the first (resp. second) term of Eq (3.6) by $e_j = exp(-\mathcal{E}_j)$ (resp. $e_{j'} = exp(-\mathcal{E}_{j'})$). Note that this weighting strategy naturally addresses the diversity of recommendation: a new French restaurant tends to have a higher weight in the learning process than Burger King, though it occurs sparsely in the check-in tensor. Combining the above location-entropy weighted distance with the input uncertainty, our final social Hausdorff distance for User v_i becomes:

$$\begin{aligned} d_{WH}(S(v_i), \mathcal{N}(v_i)) &= \frac{1}{|A+\epsilon|} \sum_{j \in S(v_i)} p_{i,j} e_j \min_{j' \in \mathcal{N}(v_i)} d(j, j') \\ &+ \frac{1}{|\mathcal{N}(v_i)|} \sum_{j' \in N(v_i)} e_{j'} \min_{j \in S(v_i)} M_\alpha [p_{i,j} d(j, j') + (1 - p_{i,j}) d_{max}], \end{aligned} \tag{3.9}$$

Note that Eq (3.9) defines the social Hausdorff distance for just one user v_i. The final social Hausdorff loss function is defined as the sum of social Hausdorff distance for all users:

$$\mathcal{L}_1 = \sum_{v_i \in V} d_{WH}(S(v_i), \mathcal{N}(v_i)). \tag{3.10}$$

The next subsection describes how to combine this social-spatial loss with the least-squares regression problem in Eq (2.3).

3.4.4 Learning with Whole Data

Most existing tensor completion models are formulated to minimize the difference between the original data tensor $\mathcal{X}$ and the reconstructed tensor $\hat{\mathcal{X}}$. Since only the positive entries (check-in records in LBSNs) are observed, negative sampling is commonly used to generate negative

entries for training. However, the performance of negative sampling is highly sensitive to the sampling strategy and the number of negative samples [71, 14]. In this paper, we propose to train our model on the whole data instead of by negative sampling. Then, the task is to minimize the mean-squared error of all entries:

$$\mathcal{L}_2 = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} w_{i,j,k}(\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2, \tag{3.11}$$

where $w_{i,j,k}$ is the entry weight for class balancing:

$$w_{i,j,k} = \begin{cases} w_+, & \text{if entry } (i,j,k) \in \Omega \text{ is positive} \\ w_-, & \text{if entry } (i,j,k) \in \Omega \text{ is unlabeled} \end{cases}$$

where we treat unlabeled entries simply as negative. Since the number of unlabeled entries is much larger than that of the positive entries, we set w_+ to be much larger than w_- so that a positive sample is as important as many unlabeled ones.

We remark that our method is a special case of negative sampling where all negative samples are sampled exactly once. Traditional negative sampling is a biased approximation and often does not converge to the same loss as computed from all entries [72, 4], making it difficult to converge to the optimal ranking due to the sensitivity to the sampling strategy. Our method is proposed to combat the drawback of negative sampling. Also, computing our new loss formulation naïvely as in the existing negative sampling scheme would lead to a prohibitive computational cost due to the large number of negative samples, and that is exactly what we address in this subsection. The time complexity of calculating Eq (3.11) is $O(I \times J \times K)$. Considering that the numbers of users and POIs (i.e., I and J) are large in recommender systems, this time cost is intractable. We, therefore, rewrite this loss function as follows to make

its evaluation affordable:

$$\begin{aligned}\mathcal{L}_2 \;&=\; \sum_{(i,j,k)\in\Omega_+} ((w_+ - w_-)\hat{\mathcal{X}}^2_{i,j,k} - 2\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}) \\ &+\; w_- \sum_{r_1=1}^{r}\sum_{r_2=1}^{r} \mathbf{h}_{r_1}\mathbf{h}_{r_2}(\sum_{i=1}^{I}\mathbf{U}^1_{i,r_1}\mathbf{U}^1_{i,r_2})(\sum_{j=1}^{J}\mathbf{U}^2_{j,r_1}\mathbf{U}^2_{j,r_2}) \\ &(\sum_{k=1}^{K}\mathbf{U}^3_{k,r_1}U^3_{k,r_2}),\end{aligned} \tag{3.12}$$

where Ω_+ and Ω_- are the set of positives and unobserved entries in the train set, respectively. We next establish the equivalence of Eq (3.12) to Eq (3.11), and then explain how Eq (3.12) reduces the time cost.

Remark 1: Eq (3.12) is equivalent to Eq (3.11).

Proof: First, we factorize $(\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2$ in Eq (3.11) and eliminate the constant term $\sum_{(i,j,k)\in\Omega}\mathcal{X}^2_{i,j,k}$:

$$\mathcal{L}_2 = \sum_{(i,j,k)\in\Omega} w_{i,j,k}\hat{\mathcal{X}}^2_{i,j,k} - 2\sum_{(i,j,k)\in\Omega} w_{i,j,k}\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k} \tag{3.13}$$

Note that the set of all entries Ω can be split into positive set Ω_+ and unlabeled set Ω_-, and $\mathcal{X}_{i,j,k}$ is 0 for entries in Ω_- since we treat unlabeled entries as negative as in negative sampling. Therefore, $\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}$ can also be eliminated for the unlabeled set Ω_- which gives:

$$\mathcal{L}_2 = \sum_{(i,j,k)\in\Omega} w_{i,j,k}\hat{\mathcal{X}}^2_{i,j,k} - 2\sum_{(i,j,k)\in\Omega_+} w_{i,j,k}\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k} \tag{3.14}$$

Now we replace $w_{i,j,k}$ with w_+ (resp. w_-) for entries in Ω_+ (resp. Ω_-):

$$
\begin{aligned}
\mathcal{L}_2 &= \sum_{(i,j,k)\in\Omega_+} w_+\hat{\mathcal{X}}^2_{i,j,k} + \sum_{(i,j,k)\in\Omega_-} w_-\hat{\mathcal{X}}^2_{i,j,k} \\
&\quad - 2\sum_{(i,j,k)\in\Omega_+} w_+\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k} \\
&= \sum_{(i,j,k)\in\Omega_+} ((w_+ - w_-)\hat{\mathcal{X}}^2_{i,j,k} - 2w_+\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}) \\
&\quad + \sum_{(i,j,k)\in\Omega_+} w_-\hat{\mathcal{X}}^2_{i,j,k} + \sum_{(i,j,k)\in\Omega_-} w_-\hat{\mathcal{X}}^2_{i,j,k} \\
&= \sum_{(i,j,k)\in\Omega_+} ((w_+ - w_-)\hat{\mathcal{X}}^2_{i,j,k} - 2w_+\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}) \\
&\quad + w_- \sum_{(i,j,k)\in\Omega} \hat{\mathcal{X}}^2_{i,j,k}
\end{aligned} \tag{3.15}
$$

In Eq (3.15), the first term and the second term can be regarded as the loss of the positive data and the whole data, respectively. Since the positive entries account for only a small portion of the whole data, the time complexity of the second term is the bottleneck of computation.

Recall from Eq (3.3) that $\hat{\mathcal{X}}_{i,j,k} = \sum_{t=1}^{r} \mathbf{h}_t \mathbf{U}^1_{i,t}\mathbf{U}^2_{j,t}\mathbf{U}^3_{k,t}$, so we rearrange $\sum\limits_{(i,j,k)\in\Omega} \hat{\mathcal{X}}^2_{i,j,k}$ in the second term as:

$$
\begin{aligned}
&\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\Big((\sum_{r_1=1}^{r}\mathbf{h}_{r_1}\mathbf{U}^1_{i,r_1}\mathbf{U}^2_{j,r_1}\mathbf{U}^3_{k,r_1})(\sum_{r_2=1}^{r}\mathbf{h}_{r_2}\mathbf{U}^1_{i,r_2}\mathbf{U}^2_{j,r_2}\mathbf{U}^3_{k,r_2})\Big) \\
&= \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\Big(\sum_{r_1=1}^{r}\sum_{r_2=1}^{r}(\mathbf{h}_{r_1}\mathbf{U}^1_{i,r_1}\mathbf{U}^2_{j,r_1}\mathbf{U}^3_{k,r_1}\mathbf{h}_{r_2}\mathbf{U}^1_{i,r_2}\mathbf{U}^2_{j,r_2}\mathbf{U}^3_{k,r_2})\Big) \\
&= \sum_{r1=1}^{r}\sum_{r2=1}^{r}\mathbf{h}_{r_1}\mathbf{h}_{r_2}(\sum_{i=1}^{I}\mathbf{U}^1_{i,r_1}\mathbf{U}^1_{i,r_2})(\sum_{j=1}^{J}\mathbf{U}^2_{j,r_1}\mathbf{U}^2_{j,r_2})(\sum_{k=1}^{K}\mathbf{U}^3_{k,r_1}\mathbf{U}^3_{k,r_2}).
\end{aligned} \tag{3.16}
$$

This reduces the time cost of computing the second term from $O(I \times J \times K \times r)$ in Eq (3.11) to $O(r^2(I + J + K))$ here. Since $I, J, K \gg r$, the time cost of computing $\mathcal{L}_2$ now becomes much more tractable.

By replacing it into the second term in Eq (3.15), we obtain Eq (3.12). Note that this rewriting process is unique to our special case rather than general negative sampling, since the last term of Eq (3.15) sums over all entries in Ω, which translates to $\sum\limits_{i=1}^{I}\sum\limits_{j=1}^{J}\sum\limits_{k=1}^{K}$ as in Eq (3.16), rather than a subset of sampled entries.

3.4.5 Loss Function

We combine the social-spatial loss head $\mathcal{L}_1$ and the least-squares loss head $\mathcal{L}_2$ into the final loss function $\mathcal{L}$ to train the model:

$$\mathcal{L} = \lambda \mathcal{L}_1 + \mathcal{L}_2 \tag{3.17}$$

where λ is a hyperparameter to adjust the importance of social Hausdorff distance loss. We use the strategy of joint training to optimize the parameters in our neural network model by directly minimizing the final loss $\mathcal{L}$. Note that a conventional tensor completion method only minimizes the difference between $\hat{\mathcal{X}}$ and $\mathcal{X}$, so it is not utilizing the social graph and POI locations.

3.5 Experiment

In this section, we report our comprehensive suite of experiments that answer the following questions regarding our model, named as TCSS (Tensor Completion with Social-Spatial regularization):

- Is it necessary to incorporate the time dimension for recommendation instead of considering just a user-POI interaction matrix?
- Can TCSS outperform existing state-of-the-art tensor completion models for time-aware recommendation?
- How does the time granularity along the time dimension influence the performance of recommendation?
- Is the performance consistent on different categories of POIs?
- Ablation study: how does training performance of TCSS compare with its counerparts using negative sampling and other initialization methods?
- How do the model hyperparameters influence the result quality?

3.5.1 Dataset

Three datasets are introduced in the experiment:

- **Gowalla**[1]**.** Gowalla was a worldwide location-based social network where users can check in at POIs. The dataset was collected from November 2010 to May 2011, including information such

[1] https://www.yongliu.org/datasets/

as users' friendship and check-in history. We only consider those users with at least 15 POI check-ins and at least one friend. We also filter out those POIs with fewer than 50 visitors. Locations in Gowalla are grouped into categories, and our preprocessing results in 6,392 shopping POIs, 5,667 entertainment POIs, 3,824 restaurant POIs and 2,272 outdoor POIs. The final preprocessed dataset contains 18,737 users and 1,666,455 check-ins in total.

- **Yelp**[2]**.** The original Yelp dataset contains 8,635,403 reviews for 160,585 worldwide businesses. We only consider those users that have at least 15 check-in records and 1 friendship relation. We filtered out those POIs with fewer than 50 visitors. The pre-processed data contains 718,214 check-ins from 17,534 users in 7,757 businesses.

- **Foursquare**[3]**.** This dataset includes global-scale check-in data collected from Foursquare during April 2012 to January 2014 (22 months). As before, we filtered out those users with fewer than 15 check-in POIs as well as those POIs with fewer than 50 visitors. The dataset also contains a user social network and we only consider those users with a least one friend. The preprocessed data contains 20,359 users, 5,777 POIs and 939,394 check-ins.

- **GMU-5K**[4]**.** This is a dense LBSN dataset generated by a simulator that simulates patterns of life. There are 11,189,377 check-ins at 8,901 POIs from 5,000 users. The density of the user-POI-time tensor is 3.21%.

3.5.2 Baselines

In order to verify the necessity of introducing the time dimension for POI recommendation, we implement two matrix-completion baselines that predict user-POI interactions:

- **MCCO**[5] **[8].** This work recovers the low-rank matrix from an incomplete set of entries. The matrix is recovered from multiplicative factors, and semidefinite programming is used to solve the convex relaxation of nuclear norm minimization.

[2]https://www.yelp.com/dataset
[3]https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[4]https://osf.io/e24th/wiki/home/
[5]https://github.com/JoonyoungYi/MCCO-numpy

Table 3.1: Results Comparison

Model		Gowalla		Yelp		Foursquare		GMU-5K	
		Hit@10	MRR	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR
Matrix completion	MCCO	0.3309	0.1804	0.3951	0.1847	0.2880	0.1367	0.3321	0.2192
	PureSVD	0.3779	0.2107	0.4307	0.2087	0.5584	0.2856	0.6651	0.3772
POI recommendation	STRNN	0.3203	0.1908	0.3294	0.1363	0.2586	0.1288	0.4847	0.3481
	STAN	0.5239	0.3311	0.5112	0.3345	0.4723	0.3215	0.6328	0.2667
	STGN	0.5230	0.4127	0.4880	0.2485	0.5101	0.3185	0.4646	0.2810
	LFBCA	0.3513	0.2470	0.3945	0.1575	0.3541	0.1909	0.3828	0.1963
Tensor completion	CP	0.4544	0.2683	0.5522	0.2479	0.5156	0.2955	0.7072	0.4917
	Tucker	0.3742	0.2168	0.5902	0.2701	0.5585	0.3188	0.6989	0.4864
	P-Tucker	0.8162	0.3793	0.6255	0.2627	0.7843	0.3401	0.7473	0.3981
	NCF	0.7891	0.3453	0.6574	0.3057	0.8079	0.4279	0.8077	0.4629
	NTM	0.6181	0.3074	0.1719	0.0839	0.7699	0.3886	0.7897	0.3493
	CoSTCo	0.7571	0.2806	0.5565	0.2106	0.8336	0.3252	0.7465	0.4420
	TCSS	**0.9177**	**0.6206**	**0.7276**	**0.3408**	**0.9298**	**0.6133**	**0.9598**	**0.6376**

Table 3.2: Ablation Study

Model Variants	Gowalla		Yelp		Foursquare		GMU-5K	
	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR
Random initialization	0.8833	0.6107	0.6892	0.3275	0.9163	0.6095	0.9045	0.5196
One-hot initialization	0.8688	0.6036	0.6704	0.3061	0.8851	0.5613	0.8968	0.4926
Remove $\mathcal{L}_1$ ($\lambda = 0$)	0.8442	0.5763	0.6308	0.2896	0.8670	0.5115	0.8421	0.4854
Negative samping	0.8549	0.4098	0.5637	0.2218	0.8917	0.4348	0.9231	0.5550
Self-Hausdorff	0.8614	0.5858	0.6478	0.3029	0.8783	0.5354	0.8654	0.5071
Zero-out	0.8574	0.5571	0.6538	0.3064	0.8321	0.5248	0.8047	0.5012
Full-Fledged TCSS	**0.9177**	**0.6206**	**0.7276**	**0.3408**	**0.9298**	**0.6133**	**0.9598**	**0.6376**

- **PureSVD**[6] **[21].** The method treats all missing values as zeros, and performs conventional single value decomposition (SVD) to factorize a sparse user-item matrix as the product of a user orthonormal matrix, a diagonal matrix of singular values, and an item orthonormal matrix.

We further introduce 10 baselines, including 6 tensor completion methods, 3 spatiotemporal POI recommendation models that predict the user-POI-time interactions, and an algorithm that predicts user-POI interactions. We feed the historical check-ins as input to these baseline methods.

- **CP & Tucker**[7] **[44].** Eq (2.1) and Eq (2.2) define the tensor completion method of CP and Tucker decomposition, respectively. CP decomposes an order-3 tensor as a sum of three rank-one tensors, while the Tucker model factorizes an order-3 tensor into one core tensor along with three factor matrices.

[6]https://github.com/yoongi0428/RecSys_PyTorch
[7]http://tensorly.org/stable/user_guide/tensor_decomposition.html

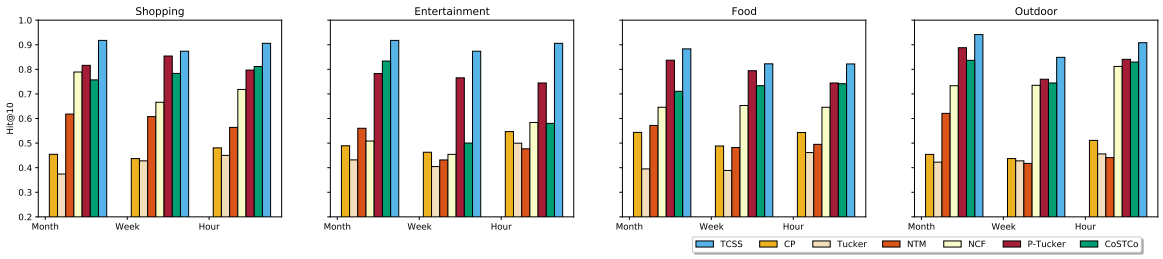


Figure 3.4: Hit@10 on Different Categories

- **NTM [16].** Nonlinear Tensor Machine also learns multi-aspect factors in recommender systems. It combines deep neural networks and tensor algebra to capture nonlinear interactions among multi-aspect factors.

- **NCF**[8] **[32].** Neural Collaborative Filtering is a neural network model that uses a multi-layer perceptron learn the nonlinear interaction relationship between the latent features. We follow NTM to feed the element-wise product of three MF vectors (user, POI, time) as the input of GMF Layer and concatenate three MLP vectors as the input of MLP Layer.

- **P-Tucker**[9] **[55]** is a scalable Tucker factorization method for sparse tensors. It performs alternating least squares with a row-wise update rule in a fully parallel way, which significantly reduces memory requirements for updating factor matrices.

- **CoSTCo**[10] **[47].** This work proposes a novel convolutional neural network (CNN) for tensor completion. It leverages the expressive power of CNN to model the complex interactions inside tensors and its parameter sharing scheme to preserve the desired low-rank structure.

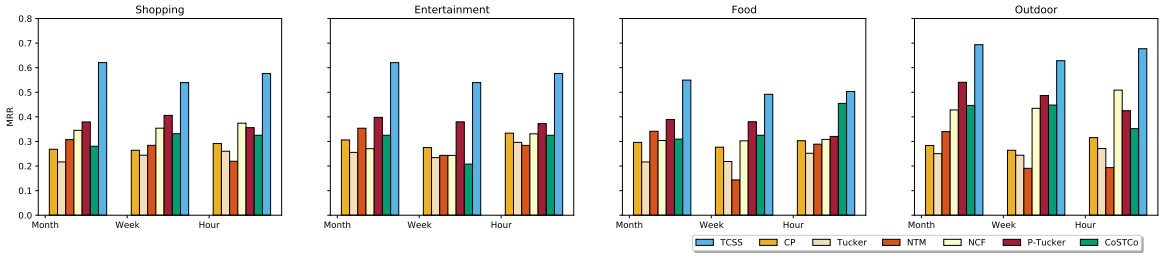


Figure 3.5: MRR on Different POI Categories

- **STRNN [49].** This work extends recurrent neural networks (RNN) and proposes a Spatial Temporal RNN to model local temporal and spatial contexts for POI recommendation.

[8] https://github.com/yihong-chen/neural-collaborative-filtering
[9] https://github.com/sejoonoh/P-Tucker
[10] https://github.com/USC-Melady/KDD19-CoSTCo

- **STAN [50].** This work proposes the Spatio-Temporal Attention Network (STAN) to exploit spatiotemporal information of all the checkins with self-attention layers along the trajectory.
- **STGN [79]** This work proposes the Spatio-Temporal Gated Network (STGN) by enhancing long-short term memory network, where spatio-temporal gates are introduced to capture the spatio-temporal relationships between successive checkins.
- **LFBCA [70]** This work proposes the location-friendship bookmark-coloring algorithm (LFBCA) to reconcile social interaction and location similarity in POI recommendation.

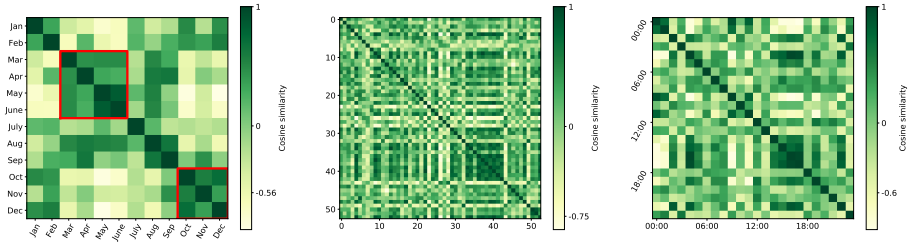
3.5.3 Performance Metrics

We adopt two widely used performance metrics that are designed to evaluate recommender systems: (1) hit ratio (Hit) [76] and (2) mean reciprocal rank (MRR) [76]. Given each entry (i, j, k) in the test set, we sample 100 random POIs $j_1, \ldots, j_{100}$ and predict the values of these entries (i, j_s, k), $s = 1, \ldots, 100$. We then sort the 100 values $\{\hat{\mathcal{X}}_{i,j_s,k}\}$ plus $\hat{\mathcal{X}}_{i,j,k}$ (i.e., 101 values in total) in non-increasing order.

Hit@10 counts the proportion of observed interactions (i, j, k) in the test set such that $\hat{\mathcal{X}}_{i,j,k}$ is within top-10 of the previous sorted list of length 101; while MRR averages the reciprocal ranks of $\hat{\mathcal{X}}_{i,j,k}$ in the sorted list over all interactions (i, j, k) in the test set, where the rank corresponds to the position of $\hat{\mathcal{X}}_{i,j,k}$ in the sorted list. We first average the reciprocal ranks of each user i along time dimension k, and then report their average over all users in the test set. On each dataset, we use 80% of check-ins as the observed tensor entries in $\mathcal{X}$ for training, and the remaining check-ins are used as the test set.

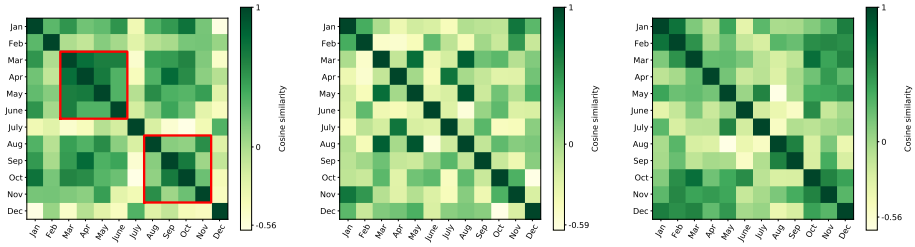
3.5.4 Model Configuration

In our experiments, we configure TCSS with the following default hyperparameters which were extensively tested and found to work consistently well. We set the weights w_+ and w_- as 0.99 and 0.01, respectively, by default. We use the default value $\lambda = 0.1$ as the weight of $\mathcal{L}_1$ in the loss computation. The length of embedding vectors is set as 10 by default, so we extract top-10 eigenvectors as the initialized embedding factors. For the time dimension, we set the granularity as month in a year (i.e., $k = 0, 1, \cdots, 11$). For example, if a check-in occurs in February, then $k = 1$. Recall Eq (3.9), where we set the default smoothing parameter α of the soft minimum function as -1, and set $\epsilon = 10^{-6}$ to



(a) Month (Blocks Highlighted) (b) Week (c) Hour

Figure 3.6: Heatmap of Similarity (Shopping)



(a) Entertainment (Blocks Highlighted) (b) Food (c) Outdoor

Figure 3.7: Similarity on Other Categories

avoid zero division. We use the Haversine formula[11] to calculate the distance between POIs considering that the POIs are distributed in a large area. All the parameters of baseline models are configured as described in the corresponding paper. We train our models using an Adam optimizer with a learning rate of 0.001 and a weight decay of 0.1. Our experiments were run on a machine equipped with a 2.20 GHz CPU, 26 GB RAM, and an NVIDIA Tesla P100 GPU.

3.5.5 Model Comparison

Table 3.1 shows the performance comparison of our TCSS model with the selected representative baseline models. Note that we include not only various tensor completion approaches but also matrix completion methods. For matrix completion, we omit the time dimension and calculate Hit@10 and MRR based on the rank of entry (i, j) in the test set. It is clear from Table 3.1 that tensor completion methods outperform matrix completion. The reason is that recommendation is time-sensitive, and most POIs have a peak period of visits in a year. For example, one may prefer to go to an aquatics center in summer instead of winter. It demonstrates the necessity of introducing the time dimension to conduct tensor completion for recommendation.

Our model achieves the best performance at around 92% Hit@10 and 0.62 MRR on Gowalla and Foursquare, outperforming the best-performing baseline, P-Tucker, by 0.1–0.2 in Hit@10 and doubles

[11] https://pypi.org/project/haversine/

MRR. On Yelp, the performance drops since the sparsity of the data tensor in Yelp is lower than that in Gowalla and Foursquare. It implies that with more entries being observed, the recommendation performance can be further improved. We also observe that neural tensor networks perform much better than CP and Tucker decomposition. It verifies the effectiveness of using neural networks to solve the tensor completion problem. Compared with these tensor completion approaches, our model is able to utilize side information and whole data for model training. The improvements verify the effectiveness of our design. Also, those predictive baselines based on spatial-temporal neural networks (STRNN, STAN, STGN) or social networks (LFBCA) do not show advantage over the tensor completion formulation as adopted by us. Note that TCSS is able to model the spatial-temporal factors and simultaneously leverage the social relations. TCSS achieves the best performance on all datasets, which verifies the effectiveness of our social Hausdorff distance and PU learning strategy for recommendation.

3.5.6 Ablation Study

We also conduct ablation study to consider several variants of TCSS to validate the effectiveness of our techniques in model design.

Social Hausdorff distance. To verify the effectiveness of the proposed Social Hausdorff distance, we introduce two variants of our model: Self-Hausdorff and Zero-out. Specifically, (i) Self-Hausdorff replaces $\mathcal{N}(v_i)$ in Eq(3.10) with the set of POIs already visited by User v_i, to remove the social influence from the loss; (ii) Zero-out is trained with $\mathcal{L}_2$ only, and it disregards any POI that has a distance greater than a threshold σ to its nearest POI of User v_i, where σ is configured as 1% of the maximum distance between any two POIs.

In both the variants, those POIs that are far from the POIs checked by User v_i before are unlikely to be recommended to v_i. However, in a real scenario, v_i may turn to friends for POI recommendations, and some recommended POIs could be far from those POIs already visited by v_i. This scenario cannot be captured by these two variants, but is well captured by our social Hausdorff distance loss head $\mathcal{L}_1$. As shown in Table 3.2, replacing our Social Hausdorff distance module with Self-Hausdorff or Zero-out leads to a performance degrade.

To further verify the effectiveness of our social Hausdorff distance loss head $\mathcal{L}_1$, we introduce another variant that only minimizes the least-squares error head (i.e., $\lambda = 0$). This method is marked by "Removing $\mathcal{L}_1$ ($\lambda = 0$)" in Table 3.2, and we can see that the performance of this variant degrades

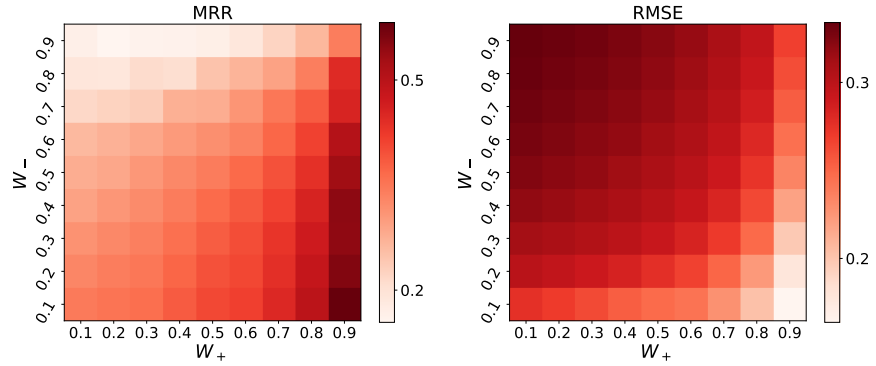
significantly on all three datasets. This further verifies that our social Hausdorff loss head is important to improve the performance of timely recommendation.

Initialization. To verify the effectiveness of our spectral method for initializing latent factors, we introduce two variants which replace our initialization method with (i) naïve random initialization and (ii) one-hot encoding. In particular, the naïve random initialization associates $\mathbf{U}_i^1$, $\mathbf{U}_j^2$ and $\mathbf{U}_k^3$ with a random vector, which is also the initialization strategy of CP and Tucker. The variant of one-hot encoding uses the method of NCF [32], which indexes each user i (resp. POI j or time k) with its corresponding position being 1 and others being 0. This high-dimensional vector is then converted to embeddings using a learnable embedding layer. We use the same value for tensor rank in both model variants that adopt the baseline initialization methods. As Table 3.2 shows, neither of the two variants outperforms TCSS on any of the 3 datasets. Note that our spectral initializing method additionally enjoys fast convergence, which we will demonstrate later.

Learning on Whole Data. Different from existing approaches, we train TCSS with all entries (including all unlabeled entries) instead of sampling a subset of unlabeled entries as negative entries. Alternatively, we introduce a variant which adopts the strategy of negative sampling in [32] to randomly sample some negative entries, the number of which equals that of the observed entries. During training, we minimize the squared error between the original tensor and reconstructed tensor over all the observed entries and the sampled negative entries. Note that the loss head based on social Hausdorff distance remains in this variant and we only replace $\mathcal{L}_1$. This variant is marked as "Negative Sampling" in Table 3.2. We observe that our training strategy on the whole data achieves better performance than negative sampling in terms of both metrics on all three datasets.

3.5.7 Analysis on POI Types and Time Granularity

Effect of POI Category. The Gowalla dataset assigns each POI a category, which provides us an opportunity to investigate how recommendation performance may vary depending on different types of POIs. Figures 3.4 and 3.5 show the results of TCSS and other baselines on four different POI categories: shopping, entertainment, food and outdoor. Note that we only consider one category in the training and test process, i.e., each tensor only involves one specific category of POIs. We observe that our model consistently outperforms all the baselines by a largin margin on all categories. Also interestingly, the performance on outdoor POIs is stronger than others. Intuitively, this is because outdoor POIs exhibit more seasonal characteristics (e.g., few people go to swimming in winter, so it is less likely to



(a) MRR (b) RSME

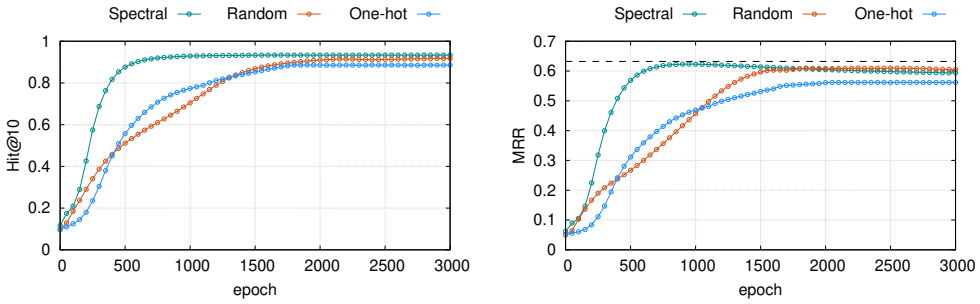
Figure 3.8: Effect of Different Weight Combinations (Gowalla)

Table 3.3: Performance with different (w_+, w_-)

(w_+, w_-)	RMSE		Hit@10	MRR
	Positive	Negtive		
(0.9, 0.1)	0.4254	0.1627	0.9061	0.5875
(0.95, 0.05)	0.4197	0.1601	0.9077	0.6008
(0.99, 0.01)	0.4148	0.1577	0.9177	0.6206
(0.995, 0.005)	0.4163	0.1593	0.9198	0.6198
(0.999, 0.001)	0.4171	0.1602	0.9184	0.6039

recommend an aquatics center to a user with a high score). Also surprisingly, the performance on the food category is relatively weak. There are two possible reasons. First, food POIs are less seasonal: people can go to a restaurant at anytime of the year. Second, people taste different types of food on a daily basis. Therefore, it is difficult to infer the potential check-in. In all four POI categories, TCSS outperforms the baselines by a large margin thanks to its leverage of social homophily that is naturally present in LBSNs.

Effect of Time Granularity. In both Figures 3.4 and 3.5, we also show the model performance when the granularity of time is varying. Specifically, for "month", the length of the time dimension is 12 since there are 12 months in a year. For example, if a check-in occurs in August, the entry has $k = 7$ (k starts from 0). Likewise, "week" indicates during which week in a year a check-in occurs. Since there are 53 weeks in year, the length of time dimension is 53. Different from "month" and "week" using the index inside a year, "hour" uses the index of hour in a day. For example, if a user visits a bar at 22:00, the time index will be 21. Generally, few users will visit a bar during the daytime, so a recommender system would be able to utilize this time-unit specific knowledge when a proper time unit is chosen to build the data tensor. From Figures 3.4 and 3.5, we observe that recommendation in the granularity of "month" maintains a stronger predictive accuracy than "week", which justifies that our month-based



(a) Hit@10 (b) MRR

Figure 3.9: Effectiveness of Initialization

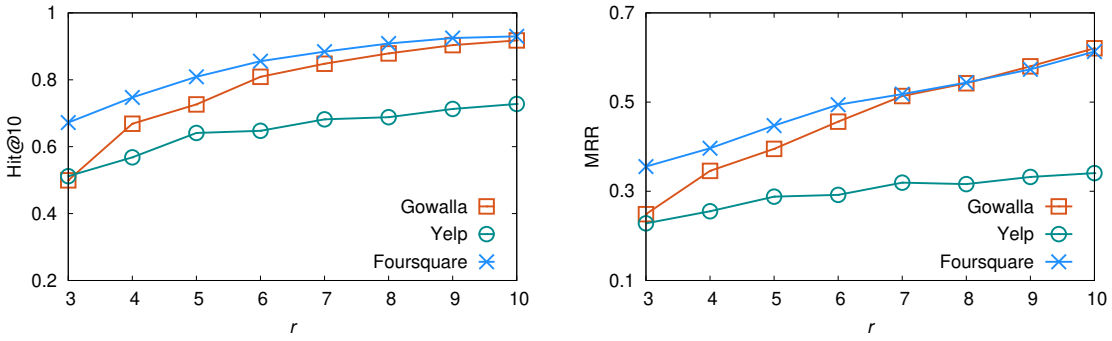
Table 3.4: Training Time (One Epoch)

Methods	Gowalla	Yelp	Foursquare
Original Loss: Eq (3.11)	2.29×10^5 s	2.49×10^5 s	4.51×10^5 s
Negative Sampling	30.12 s	11.03 s	30.61 s
Rewritten Loss: Eq (3.12)	0.13 s	0.11 s	0.17 s

time granularity in the previous subsections is a fair setting. Again, we see that TCSS significantly outperforms the baseline approaches in all the three granularity units being considered.

To further investigate the correlation of time units when using different time granularity, we calculate the cosine similarity between the latent factors (columns of $\mathbf{U}_3$) of two months, weeks and hours, respectively, the heatmaps of which are shown in Figure 3.6 where a darker color means that the factors of two time units are more similar (cosine similarity closer to 1). In Figure 3.6(a), we highlight two dark blocks in the heat map using red boxes, which suggest that the learned monthly temporal factors capture seasonal changes for recommendation. For example, one dark block in Figure 3.6(a) indicates that March, April, May and June are similar to each other. We also observe some blocks in Figures 3.6(b) and 3.6(c), but the seasonal changes revealed by weekly and hourly factors are weaker, partially explaining why their recommendation performance is not as good as the model using the "month" granularity in Figures 3.4 and 3.5.

Note that Figure 3.6 only shows the cosine similarity of different time units for the shopping POI category. Figure 3.7 further shows the cosine similarity between months for different POI categories, where we see that dark block patterns vary with the POI categories. Compared to other categories, there are fewer dark blocks on heatmap of "food", which further explains the poorer recommendation performance on the "food" category. To sum up, the more seasonality that is present in the time units adopted to re-construct the data tensor, the better TCSS prediction performance would be.



(a) Hit@10 (b) MRR

Figure 3.10: Varying r

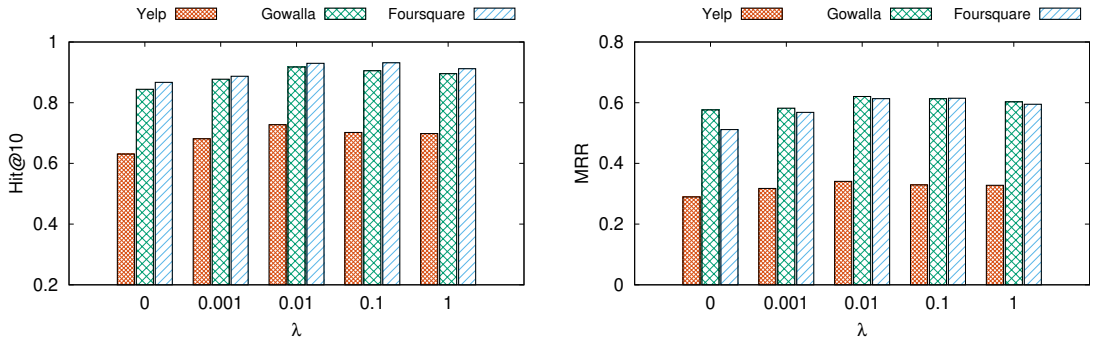
3.5.8 Setting of weights

Since the number of negative entries is much larger than that of positive entries, we give more weight to positive entries to balance their importance during training. We have carefully tuned the setting of w_+ and w_- to obtain good default values. As shown in Figure 3.8, given a fixed w_-, the MRR will increase and the RMSE will decrease as w_+ increases. It verifies that setting the positive weight w_+ to be much larger than the negative weight w_- can improve the overall performance. Note that the weight scale matters. For example, $w_+ = w_- = 0.1$ is not equivalent to $w_+ = w_- = 0.9$, because only $\mathcal{L}_2$ in Eq (3.12) contains the weights so its relative importance w.r.t. $\mathcal{L}_1$ changes with weight scaling.

We also fine-tune (w_+, w_-) beyond $(0.9, 0.1)$. As Table 3.3 shows, as the ratio w_+/w_- increases, the performance improves till $(w_+, w_-) = (0.995, 0.005)$, beyond which the performance drops. Our default $(w_+, w_-) = (0.99, 0.01)$ achieves the highest Hits@10.

3.5.9 Training Efficiency

We next report the experiments to verify the training efficiency of our initialization method and the rewritten loss function. Recall that we introduced two TCSS variants that utilize different initialization methods: (i) random and (ii) one-hot. We compare the training process of our model to these two variants. Figure 3.9 shows the change of Hit@10 and MRR in the training process. Compared with random and one-hot vector initialization, our method can lead to a much faster convergence rate and a slight performance gain in both metrics. This is because our method initializes the factors with eigenvectors which are the estimation of the genuine factors. However, random or one-hot vector initialization may get trapped in undesirable stationary points when we use gradient descent to learn the factors.



(a) Hit@10 (b) MRR

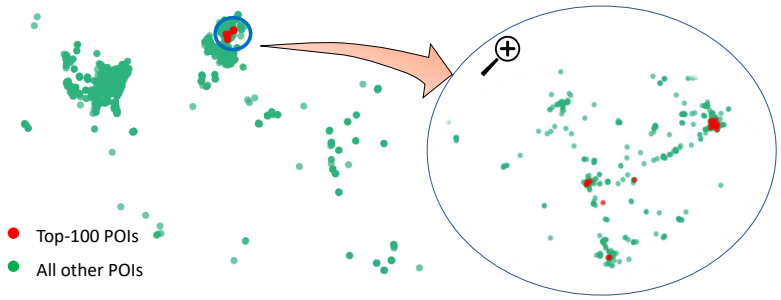
Figure 3.11: Varying λ

We also compare our training time of each epoch with two other variants: (i) negative sampling and (ii) learning on the whole data directly by Eq (3.11) without rewriting. Note that we follow [32] to generate the negative samples. Since the number of samples is large, if we put all samples in a single batch, it will crash due to memory limit. Therefore, we fill each batch with 8096 samples instead of using a single large batch. Recall that the original loss function based on the whole data has a time complexity of $O(I \times J \times K \times r)$, while our rearrangement method reduces the complexity to $O((I + J + K) \times r^2)$.

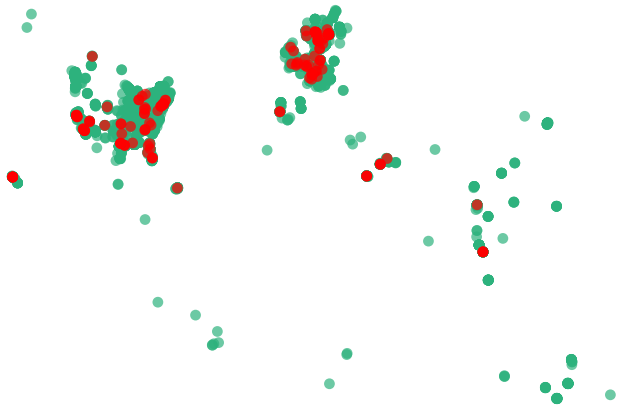
Table 3.4 shows a comparison of training time. Considering that the values of I and J are large in recommender systems (e.g., I = 20,359 in "Foursquare"), our rearranged loss can save a lot of computational resources. We observe in Table 3.4 many orders of magnitude speedup in the training time of our method over the variant that directly computes Eq (3.11). While learning on the whole data combats the sensitivity of negative sampling, our rewritten loss function makes it truly practical to learn on large real-world data. Note that our training time is only around 1% of that of negative sampling (8096 entries per batch), while being more accurate as Table 3.2 has indicated.

3.5.10 Parameter Sensitivity

The value of tensor rank r (i.e., the length of embeddings) plays a vital role in tensor completion. In Figure 3.10, we plot the performance of our model when the value of rank r varies. On all our three datasets, we see that a large value of r leads to significant gains in performance. Note that r is limited by the number of units along the time dimension (12 when month is used), K, which is much smaller than I and J. Due to the limitation of eigenvectors computation process, the maximum of r is 10 (less than $K - 1$). We can see that $r = 10$ gives the better performance than smaller values, since it allows TCSS to capture more factors.



(a) Top 100



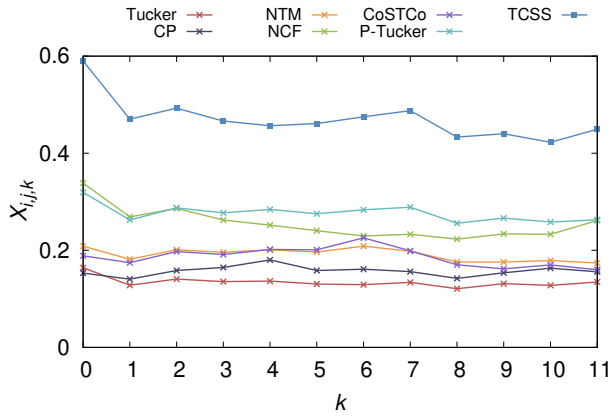
(b) Top 200

Figure 3.12: POIs with High Scores

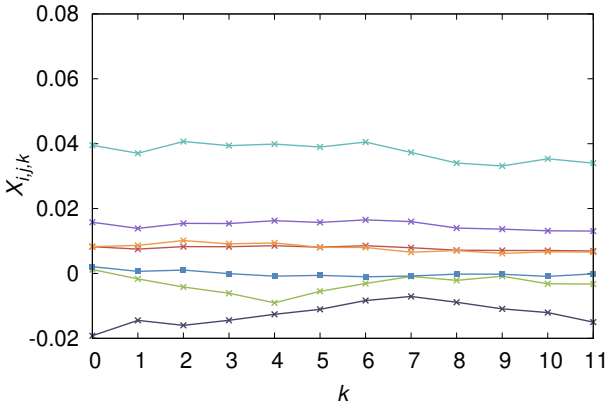
The value of λ indicates the weight of the social Hausdorff distance in the loss function $\mathcal{L} = \lambda\mathcal{L}_1 + \mathcal{L}_2$. Figure 3.11 shows the effect of the regularization weight λ of the social-homophily loss head on the overall recommendation performance. We can see that as λ increases towards 0.01, the model performance improves in terms of all metrics on all three datasets; but the performance degrades as λ increases further to 1. It indicates that there exists a tradeoff in weighing the social Hausdorff distance loss head with the least-squares error.

3.5.11 A Case Study

We next conduct a study case to investigate the recommendation scores from TCSS. Figure 3.12 shows the locations of all POIs in green. For a randomly selected user i, we sort his/her recommendation scores at a specific time k. The red points in Figure 3.12(a) highlight the top-100 scored POIs, and we can see that the POIs are clustered in small areas, which verifies the presence of Tobler's first law of geography in recommendation. As shown in Figure 3.12(b), the top-200 POIs are distributed in a much larger area, meaning that we can find a diverse set of recommended POIs as we move down the list.



(a) A positive entry



(b) A negtive entry

Figure 3.13: Score Along the Time Dimension

We also investigate how the scores vary along the time dimension for different methods. As shown in Figure 3.13(a), for a randomly selected observed entry (i, j, k), we fix User i and POI j and compute the scores along the time dimension. Compared with baselines, our model leads higher scores though the score values vary over time. For a randomly selected negative (i.e., unobserved) entry, Figure 3.13(b) shows that the scores predicted by TCSS are near 0 as expected. In summary, our model associates a possible check-in with a higher score than baselines and assigns a negative entry with a low score in the meanwhile, which verifies its recommendation accuracy.

3.6 Conclusion

In this paper, we studied the tensor completion problem for timely POI recommendation in an LBSN. We proposed a tensor completion model called TCSS to take advantage of the social-spatial side information to enforce social homophily and Tobler's first law of geography. TCSS adopts many techniques to improve recommendation quality, including a spectral based factor initialization, a novel social Hausdorff

distance head to encode the social-spatial side information, and a smart way to reduce the computational cost of least-squares error over the whole data, the latter of which improves model stability compared with negative sampling. The effectiveness of these techniques has been empirically verified.

Chapter 4

Tensor Decomposition in Hyperbolic Space

4.1 Motivation

Despite the success of existing tensor decomposition models, all previous works study the problem in Euclidean space, where the tensor is decomposed into Euclidean vectors. Recent studies show that hyperbolic space is roomier than Euclidean space due to the underlying hyperbolic geometry, i.e., space with constant negative curvature [11]. Hyperbolic vectors can represent richer information than Euclidean vectors with the same dimension. To the best of the author's knowledge, tensor decomposition in hyperbolic space remains unexplored. Therefore, we propose to decompose the input tensor not in Euclidean space but in hyperbolic space.

We remark that the benefits of tensor decomposition in hyperbolic space are two-fold. *First*, the hyperbolic vectors can embed hierarchical information efficiently [53]. For example, for the knowledge graph completion task, many knowledge graphs exhibit an underlying tree-like structure. It is intuitive to preserve hierarchical information while representing nodes or edges in the knowledge graph. As another example, [12, 68] recent studies show that hyperbolic geometry is a more suitable underlying geometry for recommendation systems. In fact, the empirical analysis indicates that many power-law distributed data and real-world interactions can often be traced back to hierarchical structures [60, 2]. *Second*, the tensor decomposition is especially challenging for large and high-dimensional tensors, since the number of parameters to be optimized scales linearly or exponentially with the tensor dimension and the length of each dimension. At the same time, as the length of decomposed vector increases, the performance of tensor decomposition models will also increase [47, 16]. A distinctive property of hyperbolic space is that the circle circumference and disc area grow exponentially with respect to radius. It means that hyperbolic space is in some sense larger than Euclidean space and the hyperbolic vectors are more expressive than Euclidean vectors with the same dimension. Therefore, we can reduce the number of

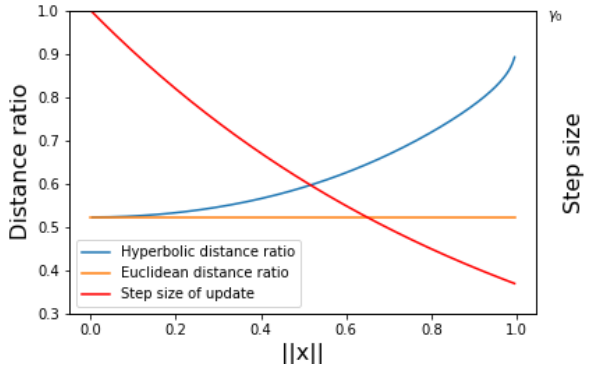


Figure 4.1: Distance ratio and step size of update

parameters to be optimized and further increase training speed while achieving the same performance as these methods in Euclidean space.

A challenge of tensor decomposition in hyperbolic space is the absence of optimization tool for hyperbolic vectors. The most popular optimization tools such as Stochastic gradient descent (SGD) and Adam have not been generalized in hyperbolic space. The reason is that existing Euclidean gradient based optimization makes no sense as an operation in hyperbolic space. For example, the basic addition operation of Euclidean vectors is not defined on hyperbolic manifold. To address this challenge, we design an adaptive optimization algorithm according to distinctive property of hyperbolic manifold. We learn the decomposed hyperbolic vectors using the gradient of loss function. To perform backpropagation of the gradient, we develop a gradient descent algorithm on hyperbolic manifold. Specifically, we first calculate the derivatives of our loss function based on hyperbolic distance metrics. Then we update the decomposed hyperbolic vectors with the derivatives of loss.

An important novelty of our algorithm is that we adaptively set the step size of update regarding the distance to the original point. In hyperbolic space, the disc area and the distance ratio grow exponentially with respect to radius. As depicted in Figure 4.1, the hyperbolic distance ratio increases exponentially as x moving towards the boundary, where Euclidean distance ratio is constant. To address this distinctive feature, we propose to assign different step size of update for factors with different distance to the original point in the space. Specifically, as x moves toward to the boundary, we reduce the step size of update exponentially in the optimization process. A unified step step will result in different moving distance for difference points. With our method, the parameters can be tuned carefully as the latent factor is moving towards the boundary in the optimization process.

Owing to the highly non-convex nature of the optimization, tensor decomposition is in general daunting to solve. The decomposed vectors may get trapped in undesirable stationary points when we

use gradient descent to learn the factors. Also, in many real-world tensors, only positive entries are observed. To train with negative data, existing methods use the strategy of negative sampling to generate the negative samples. However, the sampled data may be noisy since some potential positive entries can be treated as negative. Moreover, there is noise in the observed data since the value of an entry may be labeled by mistake. To achieve global optimal estimation instead of local minimum and decrease uncertain of negative sampling, we further improve the optimization based on positive-unlabeled data by gradient ascent trick. Specifically, instead of employing early stopping or the small-loss trick, we go up the landscape for these mini-batches whose loss reaches a fluctuation level. We employ the exponential function on the loss and use the upper-bound of local Lipschitz constant instead of the gradient to dynamically adjust the step-size of climbing according the loss. With this strategy, the negative effect of noisy data can be erased. And the optimization can jump out from the local optimal landscape.

We summarize our major contributions as follows:

- To the best of the authors' knowledge, this is first work that studies the tensor decomposition problem in hyperbolic space.
- We develop an adaptive optimization algorithm to address the distinctive feature of hyperbolic geometry.
- We adopt gradient ascent in the training process with the positive-unlabeled data to address the non-convex nature of the optimization and negative effects of noisy data.
- We verify the superiority of our tensor decomposition method on various tensors including knowledge graph tensors, user-item-time tensors and weather tensor.

4.2 Related Works to Hyperbolic Space

Hyperbolic embeddings [53, 54] are first successfully introduced in the natural language processing (NLP) motivated by the innate ability of hyperbolic spaces to embed hierarchies with low distortion [11]. Specifically, [53, 54] proposed Poincaré for learning hierarchical representations of symbolic data. Hyperbolic embeddings have also been extended to images embeddings in computer vision [42] and graph representations learning [24]. Rather than determining the hyperbolic embedding directly, Sarkar's construction is used by [66] to extract tree structure on the data. By applying the prior advancements of

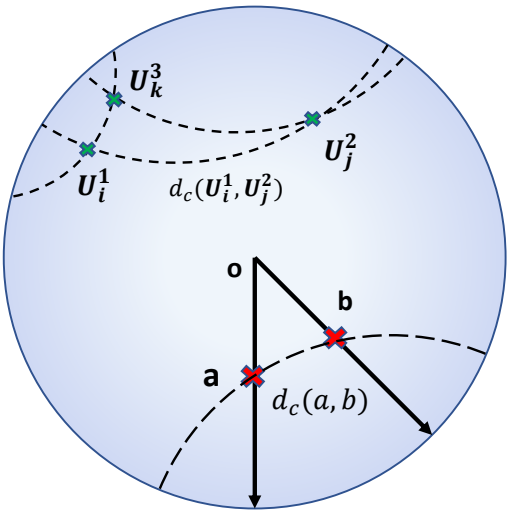


Figure 4.2: Distance ratio

hyperbolic representation learning, [1] proposes to embed the state set into the hyperbolic space and create a model of the environment in reinforcement Learning.

Despite the advancements of hyperbolic embeddings, how to use hyperbolic embeddings for downstream tasks such as classification is a challenge due to the absence of corresponding hyperbolic neural network layers. To bridge this gap, [25] combines the formalism of Möbius gyrovector spaces with the Riemannian geometry of hyperbolic spaces in a principled manner to derive hyperbolic versions of feed-forward and recurrent neural networks. Based on this framework, hyperbolic graph neural networks [48, 13] have also been developed and lead to substantial improvements on various tasks. Attention mechanism has also been introduced in hyperbolic neural networks and Klein models and extended to Klein models [26]. Different from these works based on the tangent spaces of hyperbolic manifolds, H2H-GCN directly works on hyperbolic manifold to avoid the distortion caused by tangent space approximations.

4.3 Preliminary

4.3.1 Hyperbolic Geometry

We start the problem formulation by briefly presenting some basic background of hyperbolic space. Formally, hyperbolic space is a complete simply connected Riemannian manifold with constant negative curvature. Due to the property of constant negative curvature, the geometrical properties of the hyperbolic space are very different from Euclidean space or Euclidean sphere (which has constant positive

curvature). Currently, hyperbolic space has five isometric models that one can work with: half-space, Poincaré, jemisphere, Klein, and 'Loid. In this paper, we use the r-dimensional Poincaré ball model which is most popular in machine learning:

$$\mathbb{B}^{d,c} = \{\mathbf{x} \in \mathbb{R}^d : ||\mathbf{x}||^2 < \frac{1}{c}\}, \tag{4.1}$$

where $-c(c > 0)$ is the negative curvature. Different from Euclidean addition of two vectors, the Möbius addition of $\mathbf{a}$ and $\mathbf{b}$ in hyperbolic space $\mathbb{B}^{d,c}$ is defined as:

$$\mathbf{a} \oplus_c \mathbf{b} = \frac{(1 + 2c\langle \mathbf{a}, \mathbf{b} \rangle + c||\mathbf{b}||^2)\mathbf{a} + (1 - c||\mathbf{a}||^2)\mathbf{b}}{1 + 2c\langle \mathbf{a}, \mathbf{b} \rangle + c^2||\mathbf{a}||^2||\mathbf{b}||2}. \tag{4.2}$$

Geodesic is the curve of shortest length connecting two points. It is always perpendicular to the boundary of the disk. Then the geodesic distance between $\mathbf{a}$ and $\mathbf{b}$ on the manifold $\mathbb{B}^{d,c}$ is given by:

$$d_c(\mathbf{a}, \mathbf{b}) = (2/\sqrt{c}) arcosh(\sqrt{c}|| - \mathbf{a} \oplus_{\mathbf{c}} \mathbf{b}||). \tag{4.3}$$

As $c \to 0$, $d_c(\mathbf{a}, \mathbf{b})$ becomes the Euclidean distance.

A distinctive property of hyperbolic space is that the circle circumference and disc area grow exponentially with respect to radius. Specifically, given three points: the origin $\mathbf{o}$, $\mathbf{a}$ and $\mathbf{b}$ with $||\mathbf{a}|| = ||\mathbf{b}|| = r(a \neq b)$, we measure the hyperbolic distance ratio $\frac{d_c(\mathbf{a},\mathbf{b})}{d_c(\mathbf{a},\mathbf{o})+d_c(\mathbf{o},\mathbf{b})}$ in Figure 4.2. Compare with Euclidean space where the distance ratio $\frac{d_e(\mathbf{a},\mathbf{b})}{d_e(\mathbf{a},\mathbf{o})+d_e(\mathbf{o},\mathbf{b})}$ ($d_e(\cdot)$ represents Euclidean distance) is constant, the hyperbolic distance ratio approaches to 1 exponentially as $r \to 1$. Equivalently, the shortest path from $\mathbf{a}$ to $\mathbf{b}$ is almost the same as the path through the origin as $r \to 1$. It is analogous to the property of tree data structure in which the shortest path between two sibling nodes is the path through their parent [64].

4.3.2 Problem Formulation

For simplicity and without loss of generality, we formulate the problem with an order-three tensor. We use $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ to denote the input non-negative order-three tensor, where I, J and K refer to the length of each dimension respectively. Let $\Omega = \{(i, j, k) \,|\, i \in \{0, \cdots, I-1\}, j \in \{0, \cdots, J-1\}, k \in \{0, \cdots, K-1\}\}$ be the set of observed entries. Given the value ($\mathcal{X}_{i,j,k} \geq 0$) of each (i, j, k) in Ω, we are asked to find latent factor matrices $\mathbf{U}^1 \in \mathbb{R}^{I \times r}$, $\mathbf{U}^2 \in \mathbb{R}^{J \times r}$, $\mathbf{U}^3 \in \mathbb{R}^{K \times r}$ such that the reconstructed

tensor $\hat{\mathcal{X}}$ close to the target tensor $\mathcal{X}$. Specifically, we require $\mathbf{U}_i^1$, $\mathbf{U}_j^2$ and $\mathbf{U}_k^3$ to be in the r-dimensional hyperbolic space $\mathbb{R}^r$ with $-c$ as the negative curvature. Then $\hat{\mathcal{X}}$ is reconstructed by

$$\hat{\mathcal{X}}_{i,j,k} = f(\mathbf{U}_i^1, \mathbf{U}_j^2, \mathbf{U}_k^3, c). \tag{4.4}$$

The task is to define the estimation function $f(\cdot)$ and optimize the $\mathbf{U}^1$, $\mathbf{U}^2$, $\mathbf{U}^3$. Note that the formulation and our solution can be easily extended to higher dimension.

4.4 Methodology

4.4.1 Tensor Decomposition

To solve the research problem, a natural strategy is to resort to the least-squares problem:

$$\underset{U^1,U^2,U^3}{Minimize} \sum_{(i,j,k)\in\Omega} (\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2, \tag{4.5}$$

where $\hat{\mathcal{X}}_{i,j,k}$ can be calculated from the decomposed factors as shown in Equation (5.3).

However, it is a challenge to estimate $\hat{\mathcal{X}}_{i,j,k}$ with these latent factors in hyperbolic space, because existing methods based on Euclidean vector are not available for hyperbolic vectors. Given a pair of two hyperbolic vectors, the addition and multiplication in hyperbolic space is totally different from that in Euclidean space. Therefore, the CP model in Equation (2.1) does not work for hyperbolic vectors, since the inner product is not defined in hyperbolic space. Likewise, the Tucker model does not work anymore. In fact, any tensor decomposition algorithms (e.g., neural network) based on multilinear multiplication are not suitable for our setting.

Even though some hyperbolic versions of neural network have been developed, there are some potential problems. *First*, these works resort to Möbius gyrovector spaces to implement hyperbolic neural network. By mapping the hyperbolic vectors to tangent spaces, the mapped vectors can be fed into the neural network layer. Such mapping may cause distortion because of tangent space approximations. *Second*, it is impractical to utilize the same optimization algorithm for both the latent factors and the parameters of neural network, since the latent factors are in hyperbolic space with negative curvature while the parameters of Euclidean neural network are defined in the space with zero curvature.

To address this challenge, we propose to estimate $\hat{\mathcal{X}}_{i,j,k}$ based on distance which is well defined in hyperbolic space. Specifically, we first sum all distances between any pair of two latent factors for an entry (i, j, k):

$$D_{i,j,k} = d_c(\mathbf{U}^1_i, \mathbf{U}^2_j) + d_c(\mathbf{U}^1_i, \mathbf{U}^3_k) + d_c(\mathbf{U}^2_j, \mathbf{U}^3_k). \tag{4.6}$$

Without loss of generality, for an order-M ($M \geq 3$) tensor,

$$D_{i_1,i_2,\ldots i_M} = \sum_{\substack{(m,n)\subset\{1,2,\ldots,M\} \\ m\neq n}} d_c(\mathbf{U}^m_{i_m}, \mathbf{U}^n_{i_n}). \tag{4.7}$$

Then we estimate $\hat{\mathcal{X}}_{i,j,k}$ as:

$$\hat{\mathcal{X}}_{i,j,k} = \varphi \cdot (1 - \tanh D_{i,j,k}), \tag{4.8}$$

where φ is a hyperparameter to scale the range of tensor values. With this formula, we can enforce the latent factors of an entry with larger value to be close to each other since a smaller $D_{i,j,k}$ will result in larger $\hat{\mathcal{X}}_{i,j,k}$. The reason for this design is that most tensors in real applications are highly sparse or the frequency of values follows the power-law distribution. For example, in a knowledge graph, an entity only has a few relations with other entities. In a recommender system, a user may be able to interact with a small number of items in a specific period. Recall that the disc area and the distance ratio grow exponentially with respect to the radius. In the knowledge graph, an entity frequently related with many other entities is supposed to be located near the original point to make sure that the distance to other entities is not biased. Considering that only a few "hot" (frequently related with others) entities and the distance ratio near the original point is small, it is intuitive to enforce $D_{i,j,k}$ for these entries with positive interaction to be 0 while most entries in the sparse tensor tend to be larger or infinite.

For a binary-valued tensor, the value of φ will be 1 since the range of $(1-\tanh D_{i,j,k})$ is $(0, 1]$. For a real-valued non-negative tensor (e.g., ratings in recommender system), φ is assigned as the maximum of the tensor values. For example, if the rating in a recommender system varies from 0 to 5, then $\varphi = 5$. In this paper, we focus on non-negative tensor decomposition. For an input tensor with negative values, we can easily add a constant value to all entries to transform it into a non-negative tensor.

Different from existing works in Euclidean space, our solution provides a new perspective to study the tensor decomposition problem. A tensor value is directly decomposed into hyperbolic vectors with

basic operation in hyperbolic space. Also, the activation function $\tanh$ can smooth the training process. In following sections, we will describe how to optimize the decomposed hyperbolic vectors.

4.4.2 Hyperbolic Optimizer

Let $g(w) = (\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2$ be the square function regarding w, where w is a parameter in the decomposed hyperbolic vectors ($\mathbf{U}_i^1, \mathbf{U}_j^2$ or $\mathbf{U}_k^3$) to be optimized. Recall that the target is to minimize the difference between $\hat{\mathcal{X}}$ and $\mathcal{X}$. To solve the problem in (4.5), we adopt the strategy of SGD to update w with gradient of $g(w)$.

Theorem 1 *The least-square function $g(w)$ is differentiable.*

Proof. To prove $g(w)$ is differentiable, we only need to prove $\hat{\mathcal{X}}_{i,j,k}$ is differentiable. Note that the first derivative of $\tanh(x)$ in Equation (4.8) is $1 - \tanh^2(x)$. Therefore, if $D_{i,j,k}$ is differentiable, $g(w)$ will be differentiable. According to Equation (4.6), $D_{i,j,k}$ is the sum of three distance metrics, where each of them is the distance between two points in the hyperbolic plane. Suppose $\mathbf{s}$ is the curve in hyperbolic space $\mathbb{R}^r$ from point $\mathbf{a}$ to point $\mathbf{b}$. For simplicity, we let $c = 1$. We can approximate the length of a tiny portion of $\mathbf{s}$ (i.e., from $\mathbf{s}(w)$ to $\mathbf{s}(w + \Delta w)$) by the hyperbolic distance between these two points.

To arrive at the distance differential, we map $\mathbf{s}(w)$ to the original point $\mathbf{o}$ following the approach in [5]:

$$T(z) = \frac{\mathbf{z} - \mathbf{s}(w)}{\mathbf{1} - \overline{\mathbf{s}(w)}} \tag{4.9}$$

so that:

$$d_c(\mathbf{s}(w), \mathbf{s}(t + \Delta w)) = \ln(1 + T(\mathbf{s}(t + \Delta w))) - \ln(1 - T(\mathbf{s}(t + \Delta w))). \tag{4.10}$$

We also know that $\ln(1 + x) \approx x$ as x approaches to 0. Thus, for small Δw, we have:

$$\begin{aligned} d_c(\mathbf{s}(w), \mathbf{s}(t + \Delta w)) &\approx 2 \cdot \left| \frac{\mathbf{s}(t + \Delta w) - \mathbf{s}(w)}{\mathbf{1} - \overline{\mathbf{s}(w)}\mathbf{s}(t + \Delta w)} \right| \\ &= 2 \cdot \frac{\left| \frac{\mathbf{s}(t+\Delta w) - \mathbf{s}(w)}{\Delta w} \right|}{|\mathbf{1} - \overline{\mathbf{s}(w)}\mathbf{s}(t + \Delta w)|} \cdot |\Delta w| \end{aligned} \tag{4.11}$$

As $\Delta w \to 0$, the numerator goes to $s'(w)$ and the denominator goes to $1 - |s(w)|^2$. Therefore, the distance from point $\mathbf{a}$ to point $\mathbf{b}$ can be written as:

$$d_c(\mathbf{a}, \mathbf{b}) = \int_{\mathbf{a}}^{\mathbf{b}} \frac{2}{1 - |s(w)|^2} |s'(w)| \Delta w \tag{4.12}$$

Since the $g(w)$ is a continuously differentiable function, we can use the gradient $\nabla g(w)$ to optimize the parameter. At each step of optimization algorithm, standard SGD will perform a gradient descent by $w(t+1) = w(t) - \gamma \nabla g(w)$, where the step size γ is fixed. However, the fixed step size will result in different arc-length of movement on the hyperbolic plane while the point is moving along the radius, since circle circumference and disc area grow exponentially with respect to radius.

Adaptive step size of update. The most popular optimization algorithms such as SGD and Adam use uniform step size of update in the learning process. To take into account the effects of negative curvature in hyperbolic space, we propose to calculate the step size γ adaptive according to the location. Specifically, if w is a value in U_i^1 we set γ as:

$$\gamma = \gamma_0 \cdot e^{-||U_i^1||}, \tag{4.13}$$

where γ_0 is the constant step size of update at the original point. Likewise, if w is the value of U_j^2 (resp. U_k^3), we will use $|U_j^2|$(resp. U_k^3) to calculate γ. With our method, the step size of update will be reduced if U_i^1 is moving toward the boundary. Therefore, the parameters can be tuned carefully while the distance ratio is increasing significantly in the optimization process. Also, the optimization algorithm can be stabilized regarding all factors.

Gradient Ascent. There are some challenges to solve the least-squares problem. First, due to the highly non-convex nature of the optimization, the problem is in general daunting to solve. In the learning process, the parameters may get trapped in undesirable stationary points while using gradient descent to optimize the factors. Second, in real applications, there is noise in the tensor, where some entries may be labeled with wrong values. In some scenarios such as knowledge graph completion and recommendation, only positive entries have been observed. To train with negative data, existing methods use the strategy of negative sampling to generate the negative samples randomly. However, the sampled data may be noisy since a part of potential positive entries can be treated as negative. Also, the sampling result is highly sensitive to the sampling strategy and the performance is unstable.

To achieve global optimal estimation instead of local minimum and decrease the uncertainty of negative sampling, we further improve the optimization based on positive-labeled data by gradient ascent trick. Specifically, instead of employing early stopping or the small-loss trick, we go up the gradient to prevent further fluctuation of the training loss when it reaches a small value. By gradient ascent, the landscape is

Algorithm 1: Hyperbolic latent factor optimization

Input : Latent factor $\mathbf{U}_i^1, \mathbf{U}_j^2, \mathbf{U}_k^3$,
w is a parameter in $\mathbf{U}_i^1$ to be optimized,
fluctuation level l_0

Output: New parameter $w(t+1)$

Let $\nabla g(w)$ be the gradient of loss regarding w
while *no stopping criterion has been met* **do**
  if $g(w) > l_0$ **then**
    Set step size of update $\gamma = \gamma_0 \cdot e^{-||U_i^1||}$
    $w(t+1) = w(t) - \gamma \nabla g(w)$
    `// Gradient descent`
  else
    $L_q(w) = qLg^{(q-1)}(w) + q(q-1)g^{(q-2)}(2)||\nabla g(w)||^2$
    $w(t+1) = w(t) + L_q(w)$
    `// Gradient ascent with upper-bound of the Lipschitz constant`
  end
end
return $w(t+1)$

expected to drift into an area with a loss landscape that leads to better generalization. With our aggressive strategy, the negative effect of noisy data and negative sampling can be erased with a probability and the optimization can jump out from the local optimal landscape.

Theorem 2 *The upper-bound for the local Lipschitz constant of the gradient of $g^q(w)$ at point 2 is $qLg^{(q-1)}(w) + q(q-1)g^{(q-2)}(w)\ ||\nabla g(w)||^2$ where $g(w)$ has a Lipschitz gradient with constant L.*

Proof. At any point w, we compute the Hessian $\nabla^2(g^q(w))$ as:

$$\nabla^2(g^q(w)) = qg^{(q-1)}(w)\nabla^2 g(w) + q(q-1)g^{(q-2)}(w)\nabla g(w)\nabla^T g(w) \tag{4.14}$$

Since $\nabla^2 g(w) \preceq L \times \mathbf{I}$ and $\nabla g(w)\nabla^T g(w) \preceq ||\nabla g(w)||^2 \times \mathbf{I}$ ($\mathbf{I}$ is an Identity matrix), we have:

$$||\nabla^2(g^q(w))||_2 \leq qLg^{(q-1)}(w) + q(q-1)g^{(q-2)}(w)||\nabla g(w)||^2 \tag{4.15}$$

We describe the details of our optimization process in Algorithm 1. For a parameter in the latent factor U_i^1, we update it based on the loss the input entry (i, j, k). In the learning process, we adopt the gradient descent policy before we observe that the training loss is lower than l_0. We calculate the step size of gradient descent using Equation (4.13). Once $g(w)$ is lower than the threshold l_0, we go up with

the upper-bound of the Lipschitz constant to update w. We use Lipschitz constant instead of gradient because the Lipschitz constant L can define how fast the loss can change. To jump out of the local optimum, we propose to compute the upper-bound of local Lipschitz constant of the gradient of $g^q(w)$ ($q > 1$). We employ the exponential function on the loss because it allows us to dynamically adjust the step-size of climbing according the loss. When $g^q(w)$ is large, it may be a local optimum with high probability. Since $L_q(w)$ is monotonic with respect to $g^q(w)$, $L_q(w)$ will be large. Then with a large step, w can jump out of the valley of the landscape.

4.5 Experiment

Our experiments were run on a machine equipped with a 2.20 GHz CPU, 26 GB RAM, and an NVIDIA Tesla P100 GPU. All sets of experiments were repeated 3 times and the reported experimental results were averaged over 3 runs to combat randomness.

4.5.1 Setup

Datasets. Since tensor decomposition can be applied into various tasks, we conduct experiments on three kinds of tensors: knowledge graph tensors, user-item-time tensors, weather tensor.

For the knowledge graph tensor, we choose two Benchmark datasets: DBpedia and Yago. In the tensor, each entry represents the existence of relation between two entities. Specifically, DBpedia is a knowledge graph containing 15,000 entities and 165 relations. The number of observed triples (head entity, the relation, the tail entity) is 30,291 on this knowledge graph. Yago contains 15,000 entities and 28 relations. Likewise, 26,638 triples have been observed on this dataset. We use the observed entries as Ω_+ and sample the same amount of entries as negative entries Ω_-.

For the user-item-time tensors, we choose two widely used datasets: Gowalla[1] and Yelp[2] and we adopt a three-dimensional tensor to represent the check-in events. Gowalla was a worldwide location-based social network where users can check in at POIs. The dataset was collected from 2010 to 2011, including users' check-in history. We only consider those users with at least 15 POI check-ins. The final preprocessed dataset contains 18,737 users and 1,666,455 check-ins in total. For the Yelp dataset, we only consider those users that have at least 15 check-in records. The pre-processed data contains 718,214 check-ins from 17,534 users in 7,757 businesses. For both datasets, we set the month in a year as the

[1]https://www.yongliu.org/datasets/
[2]https://www.yelp.com/dataset

Table 4.1: Result of KG tensor

	Metric	MAE					RMSE				
Dataset	model/rank	5	10	15	20	25	5	10	15	20	25
DBpedia	CP	0.2492	0.2403	0.2228	0.2142	0.2162	0.3359	0.3268	0.3179	0.3091	0.3006
	Tucker	0.2282	0.2294	0.2150	0.1960	0.1897	0.3341	0.3358	0.3177	0.3095	0.2933
	P-Tucker	0.1659	0.1634	0.1627	0.1536	0.1529	0.2383	0.2265	0.2276	0.2192	0.2177
	NCF	0.1565	0.1478	0.1451	0.1467	0.1368	0.2299	0.2106	0.1915	0.2021	0.2029
	NTM	0.1755	0.1794	0.1704	0.1609	0.1611	0.2463	0.2421	0.2454	0.2366	0.2255
	CoSTCo	0.1305	0.1328	0.1282	0.1094	0.1062	0.2090	0.2015	0.1873	0.1734	0.1767
	AGH	**0.0651**	**0.0711**	**0.0623**	**0.0635**	**0.0596**	**0.1405**	**0.1560**	**0.1445**	**0.1475**	**0.1290**
Yago	CP	0.2168	0.2157	0.2076	0.1936	0.1888	0.3511	0.3493	0.3440	0.3321	0.3128
	Tucker	0.2060	0.1993	0.1952	0.1855	0.1741	0.3362	0.3124	0.3104	0.2927	0.2873
	P-Tucker	0.1750	0.1778	0.1651	0.1627	0.1670	0.2927	0.2878	0.2979	0.2768	0.2799
	NCF	0.1801	0.1711	0.1595	0.1553	0.1503	0.2998	0.2798	0.2917	0.2983	0.2726
	NTM	0.1411	0.1381	0.1320	0.1330	0.1343	0.2457	0.2310	0.2409	0.2138	0.2355
	CoSTCo	0.1320	0.1319	0.1220	0.1183	0.1077	0.2334	0.2336	0.2348	0.2581	0.2468
	AGH	**0.0503**	**0.0541**	**0.0569**	**0.0515**	**0.0615**	**0.1077**	**0.0977**	**0.1123**	**0.0913**	**0.1466**

Table 4.2: Result of user-item-time tensor

	Metric	MAE					RMSE				
Dataset	model/rank	5	10	15	20	25	5	10	15	20	25
Gowalla	CP	0.3005	0.2991	0.2585	0.1646	0.1625	0.3588	0.3335	0.3050	0.2052	0.2066
	Tucker	0.2766	0.2765	0.2346	0.2087	0.1845	0.3350	0.3209	0.2908	0.2575	0.2185
	P-Tucker	0.2057	0.1881	0.1667	0.1792	0.1574	0.2748	0.2351	0.2151	0.2211	0.1982
	NCF	0.1474	0.1370	0.1451	0.1571	0.1511	0.2005	0.1729	0.1828	0.2006	0.1838
	NTM	0.1434	0.1305	0.1204	0.1206	0.1232	0.1901	0.1748	0.1576	0.1602	0.1558
	CoSTCo	0.1247	0.1133	0.1142	0.1077	0.1040	0.1652	0.1485	0.1493	0.1397	0.1352
	AGH	**0.0984**	**0.0992**	**0.0959**	**0.0945**	**0.0763**	**0.1341**	**0.1325**	**0.1287**	**0.1383**	**0.1036**
Yelp	CP	0.3725	0.3717	0.3418	0.3315	0.3417	0.4088	0.4069	0.3816	0.3652	0.3644
	Tucker	0.3355	0.3041	0.2985	0.2910	0.2818	0.3640	0.3350	0.3391	0.3329	0.3237
	P-Tucker	0.2625	0.2737	0.2672	0.2291	0.2192	0.3156	0.3299	0.3022	0.2443	0.2283
	NCF	0.1648	0.1620	0.1595	0.1546	0.1390	0.2067	0.1972	0.1829	0.1771	0.1601
	NTM	0.1839	0.1784	0.1504	0.1495	0.1317	0.2161	0.2029	0.1834	0.1716	0.1582
	CoSTCo	0.1572	0.1428	0.1412	0.1392	0.1380	0.2032	0.1981	0.1896	0.1637	0.1626
	AGH	**0.0893**	**0.0812**	**0.0857**	**0.0772**	**0.0785**	**0.1143**	**0.1021**	**0.1083**	**0.0937**	**0.0991**

time dimension. Likewise, we use the check-in as positive entries Ω_+ and sample the same number of entries for the set of negative entries Ω_-.

For the task of predicting extreme climate, we use the simulated dataset ExtremeWeather [59] including 768x1152 images of the global atmospheric state. We organize the weather data as an order-4 tensor recording 16 climate variables for 90 days.

We split the set of all entity pairs Ω ($\Omega = \Omega_+ \cup \Omega_-$) for the datasets of KG tensors and user-item-time tensors into 70%/30% as training and test sets.

We further introduce 6 tensor decomposition baselines:

- **CP & Tucker**[3] **[44].** Equation (2.1) and Equation (2.2) define the method of CP and Tucker decomposition, respectively. CP decomposes an order-3 tensor as a sum of three rank-one tensors, while the Tucker model factorizes an order-3 tensor into one core tensor along with three factor matrices.

- **NTM [16].** Nonlinear Tensor Machine also learns multi-aspect factors in recommender systems. It combines deep neural networks and tensor algebra to capture nonlinear interactions among multi-aspect factors.

- **NCF**[4] **[32].** Neural Collaborative Filtering is a model that uses a multi-layer perceptron to learn the nonlinear interaction relationship between the latent features. We follow NTM to use the element-wise product of three MF vectors as the input of GMF Layer and concatenate three MLP vectors as the input of MLP Layer.

- **P-Tucker**[5] **[55]** is a scalable Tucker factorization method for sparse tensors. It performs alternating least squares with a row-wise update rule in a fully parallel way, which significantly reduces memory requirements for updating factors.

- **CoSTCo**[6] **[47].** This work proposes a novel convolutional neural network (CNN) for tensor completion. It leverages the expressive power of CNN to model the complex interactions inside tensors and its parameter sharing scheme to preserve the desired low-rank structure.

Model Configuration. In our experiments, we configure our tensor decomposition algorithm named as **AGH** (**A**daptive **G**radient-based optimization in **H**yperbolic space) with the following default hyperparameters which were extensively tested and found to work consistently well. We set the negative curvature as -1 ($c = 1$). We initialize the hyperbolic factors with Xavier normal initializer. The rank (length) of hyperbolic factor is set as 20 by default. The value of φ is assigned as the maximum of the tensor values. The fixed step size of update γ_0 the origianl point is set as 0.01. In Algorithm 1, we set $q = 2$ by default. All the parameters of baseline models are configured as described in the corresponding paper.

[3] http://tensorly.org/stable/user_guide/tensor_decomposition.html
[4] https://github.com/yihong-chen/neural-collaborative-filtering
[5] https://github.com/sejoonoh/P-Tucker
[6] https://github.com/USC-Melady/KDD19-CoSTCo

Table 4.3: Result on extreme weather tensor

	Metric	MAE					RMSE				
Dataset	model/rank	5	10	15	20	25	5	10	15	20	25
ExtremeWeather	CP	0.1804	0.3951	0.1847	0.1735	0.1665	0.2245	0.2167	0.2125	0.2084	0.2054
	Tucker	0.2361	0.2344	0.2319	0.2294	0.2270	0.2893	0.2879	0.2858	0.2837	0.2817
	P-Tucker	0.0449	0.0584	0.0649	0.0832	0.0904	0.0786	0.0996	0.1120	0.1226	0.1241
	NCF	0.0351	0.0369	0.0395	0.0393	0.0378	0.0652	0.06591	0.0662	0.0679	0.0684
	NTM	0.0449	0.0484	0.0439	0.0422	0.0394	0.0786	0.0796	0.0720	0.0698	0.0684
	CoSTCo	0.0347	0.0341	0.0341	0.0337	0.0333	0.0631	0.0622	0.0621	0.0609	0.0589
	AGH	**0.0343**	**0.0326**	**0.0329**	**0.0331**	**0.0325**	**0.0587**	**0.0574**	**0.0567**	**0.0564**	**0.0551**

4.5.2 Result and Analysis

Following existing [47, 16], we report the MAE (Mean Absolute Error) and RMSE (Root Mean Square Error) between the ground-truth and the estimated value by tensor decomposition algorithms. The result of KG tensor, user-item-time tensor and weather tensor are shown in Table 4.1, Table 4.2 and Table 5.1, respectively. It is clear from tables that our methods outperform these baselines with a large margin in terms of all performance metrics. Compared with CP, Tucker and P-Tucker, our AGH addresses the non-linear interaction in the tensor. Different from all these baselines, our AGH is the only algorithm using hyperbolic vectors instead of Euclidean vectors. The advantages of our AGH over these baselines are two-fold. First, hyperbolic space is much roomier than Euclidean space and hyperbolic latent factors can deliver rich information such as hierarchical structure. For example, the knowledge graph and the user relationships in the datasets often exhibit hierarchies. Second, our optimization algorithm based on gradient ascent trick can avoid getting trapped in local optimal landscape, which can lead to overall improvement.

We also investigate the effectiveness of rank (the length of latent factor) on the result. We can see that the performance of these baselines will further improve as the rank is increasing. It implies that more parameters enable the algorithm to tune the latent factors carefully. However, the GPU memory cost and running time will also increase significantly. We also observe that the performance of our model is consistent over different ranks. It verifies that the hyperbolic vector can carry richer information than Euclidean vectors with the same dimension. Compared with these approaches in Euclidean space, our model is able to reduce the number of parameters by using small rank.

4.5.3 Ablation Study

Effectiveness of gradient ascent. To demonstrate the impact of gradient ascent in the optimization algorithm, we compare our model with one variant that only employs gradient descent. As shown in

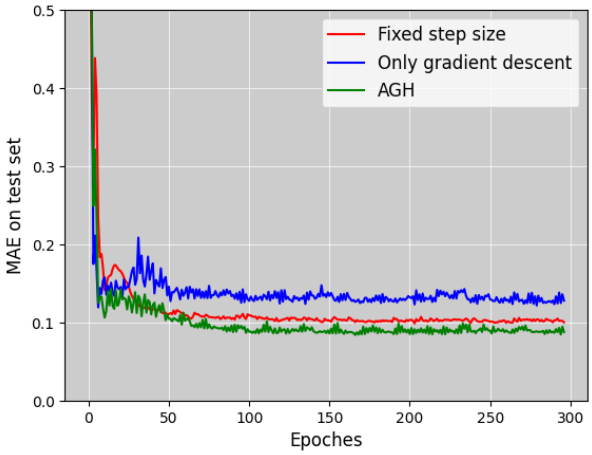


Figure 4.3: MAE over epoches

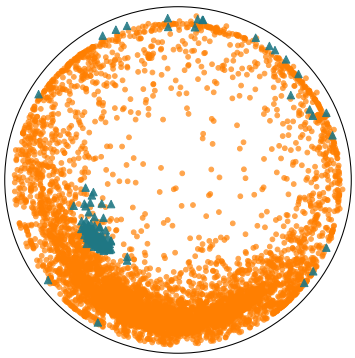


Figure 4.4: Visualization of factors in hyperbolic space

Figure 4.3, removing gradient ascent (labeled as ”Only gradient descent”) will cause the overall performance to drop. We can observe that MAE on the test starts to rise in the training process. Compared with ”Only gradient descent”, our gradient ascent can avoid the phenomenon and the factors can drift into an area with a loss landscape that leads to better generalization.

Effectiveness of adaptive step size. To verify the effectiveness of using adaptive step isze of update, we introduce a variant which uses fixed step size of update (labeled as ”Fixed step size”). As shown in Figure 4.3, with fixed step size, MAE on the test set will increase. This is because that fixed step size will ignore the dynamic distance ratio on the hyperbolic manifold. With our method, the step size of update will be reduced and the parameters can be tuned carefully while the latent factor is moving toward the boundary.

4.6 Visualization

We visualize the rank-2 factors of entities and relations for KG tensor from dataset ”DBpedia” in 2-dimensional Poincaré disk. Figure 4.4 shows factors of 15000 entities and 165 entities, where entities

are marked with yellow circles and the relations are marked with blue triangles. We can see that only a small portion of entities are located near the center. It verifies our motivation that only a small portion frequently related with many other entities. Also, we can see the relations tend to be placed near the most dense part (left-bottom corner). This is because a significant portion of relation is general, such as ”birthPlace” and ”genre”. Most entities have these general relations with other entities. Therefore, these relations tend to be placed near the most dense area.

4.7 Conclusion

In this paper, we introduce a tensor decomposition algorithm to factorize an input tensor into a number of latent factors in hyperbolic space. Compared with Euclidean factors, hyperbolic factors can carry more information with the same number of parameters. We also devise an optimization algorithm based on adaptive step size of update to address the distinctive property of hyperbolic manifold. To address the non-convex property of the problem, we adopt gradient ascent in our optimization algorithm. We verify the effectiveness of our method through extensive experiments on several tasks.

Chapter 5

Tensor Completion with Neural Ordinary Differential Equation Networks

5.1 Motivation

The ordinary differential equation describes evolution in time of some process that depends on variables and the change in time can be described via a derivative such as:

$$x + \frac{dx}{dt} = 1, \tag{5.1}$$

One of solutions to this differential equation is Euler's method. It approximates the solution function step by step using tangents lines and we will see how the process will evolve up to some final state with initial condition.

At the same time, recurrent neural network extract hidden information from input by composing a sequence of transformations:

$$\mathbf{h}(t+1) = \mathbf{h}(t) + f(\mathbf{h}(t), t, \theta_t), \tag{5.2}$$

where θ_t is the parameters in tth layer. Intuitively, these iterative updates can be considered as an Euler discretization of a continuous transformation. An ODE solver is proposed in [17] to parameterize the derivative of the hidden state using a blackbox differential equation solver based on a neural network. This neural ODE solver has been applied to various tasks including traffic prediction [23] and node classification [56]. However, there is still a gap between neural ODE and tensor decomposition.

Intuitively, if we use neural ODE to reconstruct the tensor, we can model the derivative of the latent factors. Also, an ODE solver allows us to train the neural tensor models with constant memory. Without storing all intermediate quantities in deep neural models such as RNNs (Recurrent neural networks), we can reduce the memory cost significantly.

5.2 Methodology

Same as our previous work, we use $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ to denote the input non-negative order-three tensor, where I, J and K refer to the length of each dimension respectively. Let $\mathbf{U}^1 \in \mathbb{R}^{I \times r}$, $\mathbf{U}^2 \in \mathbb{R}^{J \times r}$, $\mathbf{U}^3 \in \mathbb{R}^{K \times r}$ be the decomposed factors such that the reconstructed tensor $\hat{\mathcal{X}}$ close to the target tensor $\mathcal{X}$. We use the spectral method in 3.4.1 to initialize our decomposed factors. With a neural ODE model, we reconstruct $\mathcal{X}_{i,j,k}$ by

$$\hat{\mathcal{X}}_{i,j,k} = \alpha(\mathbf{h}(T)) = \alpha(\mathbf{h}(0) + \int_0^T \frac{d\mathbf{h}(t)}{dt} dt), \tag{5.3}$$

where T is the depth of the neural ODE model and $\mathbf{h}(0)$ is the aggregation (e.g., an element-wise mean or sum, denoted as γ) on $\mathbf{U}^1$, $\mathbf{U}^2$ and $\mathbf{U}^3$:

$$\mathbf{h}(0) = \gamma(\mathbf{U}^1_i, \mathbf{U}^2_j, \mathbf{U}^3_k) \tag{5.4}$$

This aggregation step provides a vector representation of the decomposed factors and enables us to feed the factors into a neural ODE model. In this paper, we use the concatenation operation as γ. Empirically we observe significant performance gains when using concatenation operations instead of the average operation such as element-wise sum, mean and minimization. We will show the difference in the experiment. The activation function α is used to map the output into the valid tensor value.

We parameterize the continuous dynamics of hidden units using an ordinary differential equation (ODE) specified by a neural network $f(\cdot)$:

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta) \tag{5.5}$$

We rewrite Eq. (5.5) by approximation:

$$\mathbf{h}(t+1) - \mathbf{h}(t) = f(\mathbf{h}(t), t, \theta)\Delta t, \Delta t = 1. \tag{5.6}$$

Now we replace $f(\cdot)$ with a neural network. Then the output of each layer can be written as Eq. 5.2.

Starting from the input $\mathbf{h}(0)$, we can define the output layer $\mathbf{h}(T)$ to be the solution to this ODE initial value problem and the reconstructed value in the tensor. Figure 5.1 shows the framework of our Neural ODE model.

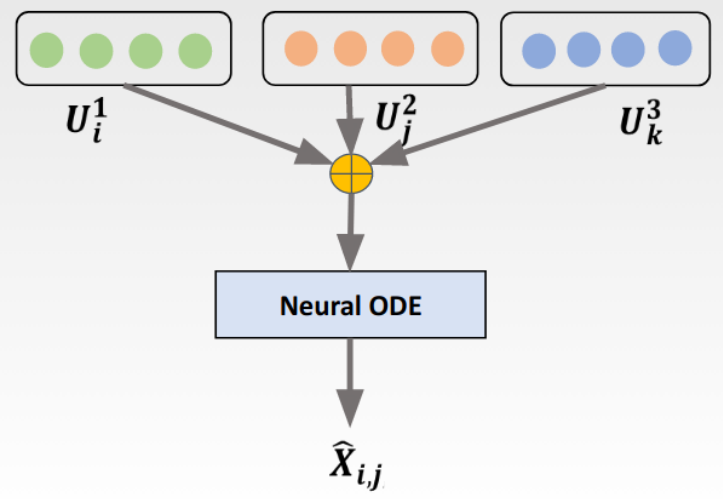


Figure 5.1: Neural ODE for tensor completion

It is straightforward to perform differentiation through the operations of the forward pass. However, the challenge lies in the high memory cost in training continuous-depth networks. The ODE solver can be considered as a black box and the gradients can be computed using the adjoint sensitivity method [57]. Concretely, this method estimates gradients by solving a second, augmented ODE backwards in time.

We use $L(\cdot)$ to represent the a scalar-valued loss function which minimizes the squared error between the original tensor and the reconstructed tensor. At a specific depth t_1, the loss can be written as:

$$L(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt) = L(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)) \tag{5.7}$$

Euler's method and Runge–Kutta method are two most representative ODE solvers. In this paper, we use the implicit Adams in the scipy.integrate package. Compared with explicit methods such as Runge-Kutta, an implicit method has better guarantees.

To optimize the model parameters, we need to compute the gradients with respect to θ. A key step is to estimate how the gradient of the loss depends on the hidden state z(t) for each entry in the training set. The terminology is called the adjoint $\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}$. The adjoint sensitivity method can be used to solve an augmented ODE backwards in steps. It can be considered as the instantaneous analog of the chain rule:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\intercal \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} \tag{5.8}$$

Table 5.1: Result of KG tensor

	Metric	MAE					RMSE				
Dataset	model/rank	5	10	15	20	25	5	10	15	20	25
DBpedia	CP	0.2492	0.2403	0.2228	0.2142	0.2162	0.3359	0.3268	0.3179	0.3091	0.3006
	Tucker	0.2282	0.2294	0.2150	0.1960	0.1897	0.3341	0.3358	0.3177	0.3095	0.2933
	P-Tucker	0.1659	0.1634	0.1627	0.1536	0.1529	0.2383	0.2265	0.2276	0.2192	0.2177
	NCF	0.1565	0.1478	0.1451	0.1467	0.1368	0.2299	0.2106	0.1915	0.2021	0.2029
	NTM	0.1755	0.1794	0.1704	0.1609	0.1611	0.2463	0.2421	0.2454	0.2366	0.2255
	CoSTCo	**0.1305**	0.1328	0.1282	0.1094	0.1062	0.2090	0.2015	0.1873	0.1734	0.1767
	ResNet	0.1476	0.1436	0.1395	0.1397	0.1381	0.1868	0.1843	0.1751	0.1725	0.1741
	Neural ODE (CONCAT)	1.3125	**0.1211**	**0.1180**	**0.1031**	**0.0918**	**0.1829**	**0.1697**	**0.1607**	0.1702	0.1860
	Neural ODE (Mean)	0.1453	0.1380	0.1291	0.1128	0.1125	0.1956	0.1833	**0.1693**	**0.1699**	**0.1703**
	Neural ODE (Sum)	0.1425	0.1376	0.1263	0.1178	0.1139	0.1990	0.1894	0.1724	0.1714	0.1757
Yago	CP	0.2168	0.2157	0.2076	0.1936	0.1888	0.3511	0.3493	0.3440	0.3321	0.3128
	Tucker	0.2060	0.1993	0.1952	0.1855	0.1741	0.3362	0.3124	0.3104	0.2927	0.2873
	P-Tucker	0.1750	0.1778	0.1651	0.1627	0.1670	0.2927	0.2878	0.2979	0.2768	0.2799
	NCF	0.1801	0.1711	0.1595	0.1553	0.1503	0.2998	0.2798	0.2917	0.2983	0.2726
	NTM	0.1411	0.1381	0.1320	0.1330	0.1343	0.2457	0.2310	0.2409	0.2138	0.2355
	CoSTCo	0.1320	0.1319	0.1220	0.1183	0.1077	0.2334	0.2336	0.2348	0.2581	0.2468
	ResNet	0.1258	0.1248	0.1131	0.1097	0.1040	0.2405	0.2346	0.2353	0.2702	0.2680
	Neural ODE (CONCAT)	0.1150	**0.1107**	**0.1065**	**0.1047**	**0.0943**	0.2378	0.2268	**0.2177**	**0.2120**	0.2229
	Neural ODE (Mean)	**0.1129**	0.1116	0.1095	0.1059	0.1046	0.2317	0.2240	0.2253	0.2290	0.2354
	Neural ODE (Sum)	0.1289	0.1230	0.1227	0.1128	0.1087	**0.2286**	**0.2227**	0.2214	0.2205	**0.2218**

With another ODEsolver, we can compute $\frac{\partial L}{\partial \mathbf{z}(t_0)}$ by running back from its final value $\mathbf{z}(t_1)$. Then computing the gradients with respect to θ depends on both $\mathbf{z}(t)$ and $\mathbf{a}(t)$:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^\intercal \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt \tag{5.9}$$

We follow [17] to compute all integrals for solving $\mathbf{z}$, $\mathbf{a}$ and $\frac{dL}{d\theta}$ with a single call to an ODE solver so that all gradients will be computed by the ODE solver at once.

5.3 Experiment

In this section, we experimentally investigate the training of neural ODEs for tensor decomposition.

5.3.1 Experimental Setup

In the experiment, the adjoint sensitivity method is implemented in Python's autograd framework. We first implement a residual network model which stacks 5 standard residual blocks to reconstruct the tensor. The output of the model will be the estimated value in the tensor. We use ResNet to indicate this baseline. To verify the effectiveness of neural ODE, we replace the residual block with an ODESolve module. We configure the value of T as 1. In the training process, we minimize the squared error between the original tensor and the reconstructed tensor. Following our previous chapter, we report

Table 5.2: Result of user-item-time tensor

	Metric	MAE					RMSE				
Dataset	model/rank	5	10	15	20	25	5	10	15	20	25
Gowalla	CP	0.3005	0.2991	0.2585	0.1646	0.1625	0.3588	0.3335	0.3050	0.2052	0.2066
	Tucker	0.2766	0.2765	0.2346	0.2087	0.1845	0.3350	0.3209	0.2908	0.2575	0.2185
	P-Tucker	0.2057	0.1881	0.1667	0.1792	0.1574	0.2748	0.2351	0.2151	0.2211	0.1982
	NCF	0.1474	0.1370	0.1451	0.1571	0.1511	0.2005	0.1729	0.1828	0.2006	0.1838
	NTM	0.1434	0.1305	0.1204	0.1206	0.1232	0.1901	0.1748	0.1576	0.1602	0.1558
	CoSTCo	0.1247	0.1133	0.1142	0.1077	0.1040	0.1652	0.1485	0.1493	0.1397	**0.1352**
	ResNet	0.1397	0.1265	0.1235	0.1117	0.1119	0.1566	0.1455	0.1442	0.1350	0.1433
	Neural ODE (CONCAT)	0.1263	0.1151	**0.1078**	**0.1037**	**0.1051**	**0.1406**	**0.1364**	**0.1335**	0.1379	0.1383
	Neural ODE (Mean)	**0.1200**	**0.1113**	01107	0.1181	0.1056	0.1519	0.1531	0.1528	0.1439	0.1356
	Neural ODE (Sum)	0.1331	0.1271	0.1268	0.1129	0.1197	0.1507	0.1438	0.1460	**0.1353**	0.1380
Yelp	CP	0.3725	0.3717	0.3418	0.3315	0.3417	0.4088	0.4069	0.3816	0.3652	0.3644
	Tucker	0.3355	0.3041	0.2985	0.2910	0.2818	0.3640	0.3350	0.3391	0.3329	0.3237
	P-Tucker	0.2625	0.2737	0.2672	0.2291	0.2192	0.3156	0.3299	0.3022	0.2443	0.2283
	NCF	0.1648	0.1620	0.1595	0.1546	0.1390	0.2067	0.1972	0.1829	0.1771	0.1601
	NTM	0.1839	0.1784	0.1504	0.1495	0.1317	0.2161	0.2029	0.1834	0.1716	0.1582
	CoSTCo	0.1572	0.1428	0.1412	0.1392	0.1380	0.2032	0.1981	0.1896	0.1637	0.1626
	ResNet	0.1442	0.1417	0.1357	0.1334	0.1337	0.2322	0.2218	0.2318	0.2263	0.2192
	Neural ODE (CONCAT)	**0.1299**	0.1301	**0.1286**	0.1197	0.1185	**0.1744**	**0.1666**	**0.1506**	**0.1564**	**0.1486**
	Neural ODE (Mean)	0.1312	**0.1275**	0.1285	**0.1158**	**0.1118**	0.1753	0.1728	0.1681	0.1686	0.1584
	Neural ODE (Sum)	0.1395	0.1372	0.1313	0.1227	0.1240	0.1823	0.1760	0.1753	0.1686	0.1567

MAE and RMSE on two kinds of tensors: knowledge graph tensors and user-item-time tensors. All factors are initialized with Xavier normal initializer. The rank (length) of factor is set as 20 by default. The length of hidden state is configured as 32 for all neural network models. The depth of ResNet is 6. We introduce 7 baselines (including ResNet) for comparison. Our experiments were run on a machine equipped with a 2.20 GHz CPU, 26 GB RAM, and an NVIDIA Tesla P100 GPU.

5.4 Result and Analysis

We evaluate the performance of neural ODE against the baselines by comparing the MAE and RMSE for tensor completion. We implement several variants of our proposed method when evaluating performance:

- **CONCAT** performs concatenation operation on the corresponding factors.
- **Mean** uses the element-wise mean as an aggregation function.
- **Mean** uses the element-wise sum as a aggregation function.

As shown in Table 5.1 and 5.2, Neural ODE is competitive in terms of both two performance metrics on all datasets. This is because we can model the derivative of latent factors with the neural ODE model. Our comparison with ResNet further confirms that the improvement comes from the Neural

ODE. With a small memory cost, we can implement a deep neural network to estimate the value in the input tensor. Also, the concatenation operation can achieve the best performance on average. With both mean and sum aggregation function, we may lose some hidden information in the latent factors. Instead, with a concatenation operation, the neural network can learn the hidden interaction without losing information.

5.4.1 Memory Cost

One advantage of Neural ODE is that it does not save the intermediate states of the forward pass. As a result, we can train the model with constant memory cost as a function of depth. Table 5.3 shows the memory cost, number of parameters and running time of each batch for both ResNet and Neural ODE. We can observe that Neural ODE has much fewer parameters than ResNet. Also, we can train Neural ODE for tensor decomposition with similar memory cost and running time as ResNet.

Table 5.3: Performance on DBpedia

	Memory cost	# of parameters	Time
ResNet	2,081 MB	2,352,394	1.718 s
Neural ODE	2,217 MB	208,266	1.787 s

Chapter 6

Conclusion

In this dissertation, we leverage the power of tensor decomposition to understand sparse tensors in real world. In our first work, we study if the side information can be used to improve the performance of tensor decomposition. This work is inspired by the social homophily theory. We demonstrate the superiority of our proposed framework by experimenting with multiple real world datasets. The result shows that our method can efficiently utilize side information in a tensor decomposition setting. In our second work, we investigate the problem of decomposing the input tensor in hyperbolic space instead of Euclidean space. We propose a method to reconstruct the tensor with hyperbolic vectors. To optimize the hyperbolic factors, we design a special optimization algorithm. Out algorithm is able to address the distinctive feature of hyperbolic geometry. The empirical result shows that hyperbolic vectors can carry richer information than Euclidean vectors with the same dimension. In our third work, we propose to model the derivatives of input factors in tensor decomposition. Concretely, we aggregate the decomposed factors and feed it into a neural ODE model to estimate the values in the tensor. The experiments reveals that considering the derivatives of factors can further improve the performance for tensor decomposition.

Up to now, we mainly focus completing the sparse tensor by tensor decomposition. In the last few years, the diffusion model has achieved a great success in computer vision and natural language processing. Considering that the diffusion model is powerful and tractable to generate data, we think that diffusion models are promising to fill the unobserved value in sparse tensors.

References

[1] Hyperbolic embeddings for learning options in hierarchical reinforcement learning. *CoRR*, abs/1812.01487, 2018.

[2] Aaron B. Adcock, Blair D. Sullivan, and Michael W. Mahoney. Tree-like structure in large social and information networks. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, pages 1–10. IEEE Computer Society, 2013.

[3] Orhun Utku Aydin, Abdel Aziz Taha, Adam Hilbert, Ahmed A. Khalil, Ivana Galinovic, Jochen B. Fiebach, Dietmar Frey, and Vince Istvan Madai. On the usage of average hausdorff distance for segmentation performance assessment: Hidden bias when used for ranking, 2020.

[4] Guy Blanc and Steffen Rendle. Adaptive sampled softmax with kernel based sampling. In *ICML*, 2018.

[5] Ralph P Boas. *Invitation to complex analysis*, volume 20. American Mathematical Soc., 2020.

[6] Changxiao Cai, Gen Li, H. Vincent Poor, and Yuxin Chen. Nonconvex low-rank symmetric tensor completion from noisy data. *CoRR*, 2019.

[7] Changxiao Cai, H. Vincent Poor, and Yuxin Chen. Uncertainty quantification for nonconvex tensor completion: Confidence intervals, heteroscedasticity and optimality. In *ICML*, 2020.

[8] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119, 2012.

[9] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 1970.

[10] J Douglas Carroll, Sandra Pruzansky, and Joseph B Kruskal. Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika*, 1980.

[11] Benjamin Paul Chamberlain, James R. Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space. *CoRR*, abs/1705.10359, 2017.

[12] Benjamin Paul Chamberlain, Stephen R. Hardwick, David R. Wardrope, Fabon Dzogang, Fabio Daolio, and Saúl Vargas. Scalable hyperbolic recommender systems. *CoRR*, abs/1902.08648, 2019.

[13] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *NeurIPS*, 2019.

[14] Chong Chen, Min Zhang, Chenyang Wang, Weizhi Ma, Minming Li, Yiqun Liu, and Shaoping Ma. An efficient adaptive transfer neural network for social-aware recommendation. In *SIGIR*, 2019.

[15] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. Efficient neural matrix factorization without sampling for recommendation. *ACM Trans. Inf. Syst.*, 38(2):14:1–14:28, 2020.

[16] Huiyuan Chen and Jing Li. Neural tensor model for learning multi-aspect factors in recommender systems. In *IJCAI 2020*.

[17] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

[18] Yuxin Chen, Yuejie Chi, Jianqing Fan, and Cong Ma. Gradient descent with random initialization: fast global convergence for nonconvex phase retrieval. *Math. Program.*, 2019.

[19] Dehua Cheng, Richard Peng, Yan Liu, and Ioakeim Perros. SPALS: fast alternating least squares via implicit leverage scores sampling. In *NeurIPS*, 2016.

[20] Justin Cranshaw, Eran Toch, Jason I. Hong, Aniket Kittur, and Norman M. Sadeh. Bridging the gap between physical location and online social networks. In *UbiComp*, 2010.

[21] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In Xavier Amatriain, Marc Torrens, Paul Resnick, and Markus Zanker, editors, *RecSys*, pages 39–46. ACM, 2010.

[22] Xiaomin Fang, Rong Pan, Guoxiang Cao, Xiuqiang He, and Wenyuan Dai. Personalized tag recommendation through nonlinear tensor factorization using gaussian kernel. In *AAAI*, 2015.

[23] Zheng Fang, Qingqing Long, Guojie Song, and Kunqing Xie. Spatial-temporal graph ODE networks for traffic flow forecasting. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 364–373. ACM, 2021.

[24] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic entailment cones for learning hierarchical embeddings. In Jennifer G. Dy and Andreas Krause, editors, *ICML*, 2018.

[25] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. In *NeurIPS*, 2018.

[26] Çaglar Gülçehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter W. Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. Hyperbolic attention networks. In *ICLR*, 2019.

[27] Richard A Harshman. Parafac2: Mathematical and technical notes. *UCLA working papers in phonetics*, 1972.

[28] Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[29] Richard A Harshman and Margaret E Lundy. Uniqueness proof for a family of models sharing features of tucker's three-mode factor analysis and parafac/candecomp. *Psychometrika*, 1996.

[30] Lifang He, Xiangnan Kong, Philip S. Yu, Xiaowei Yang, Ann B. Ragin, and Zhifeng Hao. Dusk: A dual structure-preserving kernel for supervised tensor learning with applications to neuroimages. In Mohammed Javeed Zaki, Zoran Obradovic, Pang-Ning Tan, Arindam Banerjee, Chandrika Kamath, and Srinivasan Parthasarathy, editors, *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 127–135. SIAM, 2014.

[31] Lifang He, Chun-Ta Lu, Guixiang Ma, Shen Wang, LinLin Shen, Philip S. Yu, and Ann B. Ragin. Kernelized support tensor machines. In *ICML*, 2017.

[32] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *WWW*, 2017.

[33] Joyce C. Ho, Joydeep Ghosh, and Jimeng Sun. Marble: high-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization. In *SIGKDD*, pages 115–124. ACM, 2014.

[34] Bo Hu and Martin Ester. Spatial topic modeling in online social media for location recommendation. In Qiang Yang, Irwin King, Qing Li, Pearl Pu, and George Karypis, editors, *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013*, pages 25–32. ACM, 2013.

[35] Bo Hui, Haiquan Chen, Da Yan, and Wei-Shinn Ku. EDGE: entity-diffusion gaussian ensemble for interpretable tweet geolocation prediction. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 1092–1103. IEEE, 2021.

[36] Bo Hui, Da Yan, and Wei-Shinn Ku. Node-polysemy aware recommendation by matrix completion with side information. In *2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, December 15-18, 2021*, pages 636–642. IEEE, 2021.

[37] Bo Hui, Da Yan, Wei-Shinn Ku, and Wenlu Wang. Predicting economic growth by region embedding: A multigraph convolutional network approach. In Mathieu d'Aquin, Stefan Dietze, Claudia

Hauff, Edward Curry, and Philippe Cudré-Mauroux, editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 555–564. ACM, 2020.

[38] Prateek Jain and Sewoong Oh. Provable tensor factorization with missing data. In *NeurIPS*, 2014.

[39] Jun-Gi Jang and U Kang. D-tucker: Fast and memory-efficient tucker decomposition for dense tensors. In *ICDE*, 2020.

[40] U Kang, Evangelos E. Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, *SIGKDD*, 2012.

[41] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys*, 2010.

[42] Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan V. Oseledets, and Victor S. Lempitsky. Hyperbolic image embeddings. In *CVPR*, 2020.

[43] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*. OpenReview.net, 2017.

[44] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 2009.

[45] Timothée Lacroix, Guillaume Obozinski, and Nicolas Usunier. Tensor decompositions for temporal knowledge base completion. In *ICLR*, 2020.

[46] Defu Lian, Cong Zhao, Xing Xie, Guangzhong Sun, Enhong Chen, and Yong Rui. Geomf: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *SIGKDD*, 2014.

[47] Hanpeng Liu, Yaguang Li, Michael Tsang, and Yan Liu. Costco: A neural tensor completion model for sparse tensors. In *SIGKDD*, 2019.

[48] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. In *NeurIPS*, 2019.

[49] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. Predicting the next location: A recurrent model with spatial and temporal contexts. In *AAAI*, 2016.

[50] Yingtao Luo, Qiang Liu, and Zhaocheng Liu. STAN: spatio-temporal attention network for next location recommendation. In *WWW*, 2021.

[51] Sreekanth Madisetty. Event recommendation using social media. In *ICDE*, 2019.

[52] Peter R Monge, Noshir S Contractor, Peter S Contractor, R Peter, S Noshir, et al. *Theories of communication networks*. Oxford University Press, USA, 2003.

[53] Maximilian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *NeurIPS*, 2017.

[54] Maximilian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *ICML*, 2018.

[55] Sejoon Oh, Namyong Park, Lee Sael, and U Kang. Scalable tucker factorization for sparse tensors - algorithms and discoveries. In *ICDE*, 2018.

[56] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *CoRR*, abs/1911.07532, 2019.

[57] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[58] Aaron Potechin and David Steurer. Exact tensor completion with sum-of-squares. In *COLT*, 2017.

[59] Evan Racah, Christopher Beckham, Tegan Maharaj, Samira Ebrahimi Kahou, Prabhat, and Chris Pal. Extremeweather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events. In *NeurIPS*, 2017.

[60] Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical review E*, 67(2):026112, 2003.

[61] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, 2010.

[62] Javier Ribera, David Guera, Yuhao Chen, and Edward J. Delp. Locating objects without bounding boxes. In *CVPR*, pages 6479–6489. Computer Vision Foundation / IEEE, 2019.

[63] Bernardino Romera-Paredes and Massimiliano Pontil. A new convex relaxation for tensor completion. In *NeurIPSs*, 2013.

[64] Frederic Sala, Christopher De Sa, Albert Gu, and Christopher Ré. Representation tradeoffs for hyperbolic embeddings. In Jennifer G. Dy and Andreas Krause, editors, *ICML*, 2018.

[65] Qingquan Song, Xiao Huang, Hancheng Ge, James Caverlee, and Xia Hu. Multi-aspect streaming tensor completion. In *SIGKDD*, 2017.

[66] Rishi Sonthalia and Anna C. Gilbert. Tree! I am no tree! I am a low dimensional hyperbolic embedding. In *NeurIPS*, 2020.

[67] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970.

[68] Lucas Vinh Tran, Yi Tay, Shuai Zhang, Gao Cong, and Xiaoli Li. Hyperml: A boosting metric learning approach in hyperbolic space for recommender systems. In *WSDM*, 2020.

[69] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[70] Hao Wang, Manolis Terrovitis, and Nikos Mamoulis. Location recommendation in location-based social networks using user check-in data. In Craig A. Knoblock, Markus Schneider, Peer Kröger, John Krumm, and Peter Widmayer, editors, *21st SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2013, Orlando, FL, USA, November 5-8, 2013*, pages 364–373. ACM, 2013.

[71] Menghan Wang, Mingming Gong, Xiaolin Zheng, and Kun Zhang. Modeling dynamic missingness of implicit feedback for recommendation. In *NeurIPS*, 2018.

[72] Xin Xin, Fajie Yuan, Xiangnan He, and Joemon M. Jose. Batch IS NOT heavy: Learning word representations from all samples. In *ACL*, 2018.

[73] Carl Yang, Lanxiao Bai, Chao Zhang, Quan Yuan, and Jiawei Han. Bridging collaborative filtering and semi-supervised learning: A neural approach for POI recommendation. In *SIGKDD*, 2017.

[74] Lina Yao, Quan Z. Sheng, Yongrui Qin, Xianzhi Wang, Ali Shemshadi, and Qi He. Context-aware point-of-interest recommendation using tensor factorization with social regularization. In *SIGIR*, pages 1007–1010. ACM, 2015.

[75] Quanming Yao, James Tin-Yau Kwok, and Bo Han. Efficient nonconvex regularized tensor completion with structure-aware proximal iterations. In *ICML*, 2019.

[76] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983. ACM, 2018.

[77] Rose Yu, Dehua Cheng, and Yan Liu. Accelerated online low rank tensor learning for multivariate spatiotemporal streams. In Francis R. Bach and David M. Blei, editors, *ICML*, 2015.

[78] Zhanqiu Zhang, Jianyu Cai, and Jie Wang. Duality-induced regularizer for tensor factorization based knowledge graph completion. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020.

[79] Pengpeng Zhao, Haifeng Zhu, Yanchi Liu, Jiajie Xu, Zhixu Li, Fuzhen Zhuang, Victor S. Sheng, and Xiaofang Zhou. Where to go next: A spatio-temporal gated network for next POI recommendation. In *AAAI*, 2019.