Neural Tensor Model for Learning Multi-Aspect Factors in Recommender Systems

Huiyuan Chen and **Jing Li**
Department of Computer and Data Sciences, Case Western Reserve University
hxc501@case.edu, jingli@cwru.edu

Abstract

Recommender systems often involve multi-aspect factors. For example, when shopping for shoes online, consumers usually look through their images, ratings, and product's reviews before making their decisions. To learn multi-aspect factors, many context-aware models have been developed based on tensor factorizations. However, existing models assume multilinear structures in the tensor data, thus failing to capture nonlinear feature interactions. To fill this gap, we propose a novel nonlinear tensor machine, which combines deep neural networks and tensor algebra to capture nonlinear interactions among multi-aspect factors. We further consider adversarial learning to assist the training of our model. Extensive experiments demonstrate the effectiveness of the proposed model.

1 Introduction

Context-aware recommendations have attracted great attention due to their beneficialness of exploring multi-aspect factors [Bhargava *et al.*, 2015; Chen and Li, 2017]. As an example in Figure 1, user reviews are valuable to explain what properties of an item they prefer; Item images provide abundant information, including color, shape, and fashion style. Furthermore, recommending a proper item to users at the right time is also a fundamental task [Karatzoglou *et al.*, 2010]. Incorporating multi-aspect factors is thus a promising strategy to improve recommendation accuracy.

Tensor-based models are naturally preferred due to their ability to express multi-aspect data [Kolda and Bader, 2009]. However, existing tensor models suffer from three drawbacks. First, it is well known that the performance can be hindered by its multilinear assumption, thus failing to capture nonlinear patterns. Second, when the observed tensor is sparse, only mining the tensor provides little information. The coupled tensor-matrix models can alleviate data sparsity by incorporating auxiliary information [Acar *et al.*, 2011; Chen and Li, 2018]. Nevertheless, they need to construct *fixed-dimensional* feature matrices to couple with tensors. As such, these feature matrices are insufficient to capture the complicated data manifold, especially for image or textual data [Zhang *et al.*, 2017]. Finally, sampling negative training instances becomes a nontrivial task since the observed tensor typically only contains positive instances [Yu *et al.*, 2018; Kolda and Bader, 2009; Chen and Li, 2019]. A uniformly random sampling is widely used to generate negative instances. However, those negative samples can be completely unrelated and can be easily discriminated from observed samples, leading to *zero loss* phenomenon [Wang *et al.*, 2017].

Figure 1: A user's decision involves multi-aspect factors.

Here we propose a Neural Tensor Machine (NTM), which extends multilinear tensor factorizations by combining the deep neural networks and tensor algebra. Specifically, NTM takes a multi-aspect tensor as input, and aims to learn nonlinear feature interactions among data. NTM contains two components: a shallow GCP layer and a deep tensorized MLP (Figure 2). As such, NTM is able to characterize nonlinear feature interactions. We further consider adversarial learning to supply high-quality negative samples, which improve its generalization performance.

2 Related Work

Context-Aware Recommendation. Many studies have focused on using contextual factors to improve system accuracy, such as review-aware [Catherine and Cohen, 2017; Zhang *et al.*, 2017], visually-aware [Yu *et al.*, 2018; He and McAuley, 2016], and time-aware recommendation [Karatzoglou *et al.*, 2010; Wu *et al.*, 2019]. For instance, DeepCoNN [Zheng *et al.*, 2017] and ConvMF [Kim *et al.*, 2016] explored user reviews to study users' behaviors; Vista [He *et al.*, 2016] and DCFA [Yu *et al.*, 2018] incorporated visual features of items to enhance the accuracy; MR [Karatzoglou *et al.*, 2010] and NTF [Wu *et al.*, 2019] exploited temporal behaviors by mining a user-item-time tensor. Inspired by these models, we seek to build a new tensor model that integrate multi-aspect factors together.

Tensor Factorization. Tensor machines have yielded great promise in context-aware recommendation. Many of them are built upon the multilinear CANDECOMP/PARAFAC (CP) or Tucker tensor factorizations [Kolda and Bader, 2009].

The CP model decomposes a third-order tenor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{M\times N\times L}$ into three factor matrices: $\mathbf{U} \in \mathbb{R}^{M\times r}$, $\mathbf{V} \in \mathbb{R}^{N\times r}$, and $\mathbf{W} \in \mathbb{R}^{L\times r}$, such that a tensor entry can be estimated by:

$$\hat{\boldsymbol{\mathcal{X}}}_{ijk} = \sum_{t=1}^{r} \mathbf{U}_{it}\mathbf{V}_{jt}\mathbf{W}_{kt} \tag{1}$$

The Tucker decomposes a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{M\times N\times L}$ into a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1\times r_2\times r_3}$ with three orthogonal factor matrices: $\mathbf{U} \in \mathbb{R}^{M\times r_1}$, $\mathbf{V} \in \mathbb{R}^{N\times r_2}$, and $\mathbf{W} \in \mathbb{R}^{L\times r_3}$, such that:

$$\hat{\boldsymbol{\mathcal{X}}}_{ijk} = \sum_{a=1}^{r_1}\sum_{b=1}^{r_2}\sum_{c=1}^{r_3} \boldsymbol{\mathcal{G}}_{abc}\mathbf{U}_{ia}\mathbf{V}_{jb}\mathbf{W}_{kc} \tag{2}$$

As we can see, both the CP and Tucker models interpret the three-way interactions using multilinear multiplication. For example, the CP model estimates $\hat{\boldsymbol{\mathcal{X}}}_{ijk}$ by linearly combining of its latent factors with equal contribution. We argue that this strategy may be insufficient to capture nonlinear feature interactions. Our work here builds on this line of work and address this limitation by using deep neural networks.

Nonlinear Tensor Factorization. Recent studies have shown that nonlinear tensor models have superior performance over multilinear tensor models [Fang *et al.*, 2015; Zhe *et al.*, 2016; Xu *et al.*, 2012; Liu *et al.*, 2019; Wu *et al.*, 2019]. For instance, NLTF [Fang *et al.*, 2015] and InfTucker [Xu *et al.*, 2012] proposed to use Gaussian kernel to capture the nonlinear feature interactions. NTF [Wu *et al.*, 2019] replaced the multilinear operations with multi-layer perceptrons. CoSTCo [Liu *et al.*, 2019] used convolutional neural network to compress the tensor. In contrast, our work explicitly uses tensor algebra to capture triple-wise interactions among multi-aspect factors.

Lastly, it is worth mentioning that our work is different from recent deep tensor networks [Novikov *et al.*, 2015; Socher *et al.*, 2013; Cohen *et al.*, 2016; Lebedev *et al.*, 2015]. The tensor in [Socher *et al.*, 2013] aimed to connect nodes in knowledge graphs. [Novikov *et al.*, 2015; Lebedev *et al.*, 2015; Cohen *et al.*, 2016] focused on establishing relationships between tensor and deep learning, not on nonlinear tensor factorization. Our model here is more general to study nonlinear patterns for multi-aspect tensor data.

3 Problem Formulation

3.1 Problem Setup

Following [Yu *et al.*, 2018; Karatzoglou *et al.*, 2010], we use a $user \times item \times time$ tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{M\times N\times L}$ to indicate the purchase events, where M, N, and L denote the number of users, items, and time intervals, respectively. We consider recommendation problem with implicit feedback: $\boldsymbol{\mathcal{X}}_{ijk} = 1$, if user i purchases item j during time interval k, otherwise $\boldsymbol{\mathcal{X}}_{ijk} = 0$. Our goal is to generate for users a personalized ranking of items that they have not yet provided feedback at certain time [Yu *et al.*, 2018; Wu *et al.*, 2019]. In other words, we aim to infer the unobserved elements $\hat{\boldsymbol{\mathcal{X}}}_{ijk}$ to estimate a score among user i, item j, and time k.

3.2 Feature Extraction

In this section, we present the related features in details.

Embedding Look-up. Given a user i, an item j, and a time interval k, their one-hot features $\mathbf{a}_i \in \mathbb{R}^M$, $\mathbf{b}_j \in \mathbb{R}^N$, and $\mathbf{c}_k \in \mathbb{R}^L$ can be obtained based on their identities. One can obtain dense embeddings via three lookup tables:

$$\widehat{\mathbf{u}}_i \leftarrow \text{lookup}(\mathbf{a}_i), \quad \widehat{\mathbf{v}}_j \leftarrow \text{lookup}(\mathbf{b}_i), \quad \widehat{\mathbf{w}}_k \leftarrow \text{lookup}(\mathbf{c}_i) \tag{3}$$

here $\widehat{\mathbf{u}}_i \in \mathbb{R}^{d_1}$, $\widehat{\mathbf{v}}_j \in \mathbb{R}^{d_2}$, and $\widehat{\mathbf{w}}_k \in \mathbb{R}^{d_3}$ are new embeddings for user i, item j, and time k, respectively.

In addition to identity features, we further extract the features from users' reviews and items' images as follows.

Textual features. We extract features from users' reviews via the KimCNN [Kim, 2014]. Let $text_i$ denote the textual reviews of user i, which concatenates *all* reviews written by user i [Zheng *et al.*, 2017], we have:

$$\mathbf{x}_i \leftarrow \text{KimCNN}(text_i)$$

here $\mathbf{x}_i \in \mathbb{R}^m$ is the review-based representation for user i.

Visual features. We adopt the well-known Caffe model to extract visual features from images [Jia *et al.*, 2014]. Given an item's $image_j$, we obtain its visual feature by pulling out the second fully-connected layer of the Caffe (i.e. FC7):

$$\mathbf{y}_j \leftarrow \text{CNN}(image_j)$$

here $\mathbf{y}_j \in \mathbb{R}^n$ is the image-based representation for item j.

Fusion features. We fuse all features together for better representation learning. Given user's features $(\widehat{\mathbf{u}}_i, \mathbf{x}_i)$, item's features $(\widehat{\mathbf{v}}_j, \mathbf{y}_j)$, and temporal feature $\widehat{\mathbf{w}}_k$, we have:

$$\begin{aligned} \mathbf{u}_i &\leftarrow \text{FC}(\Theta_u; \textsc{concat}(\widehat{\mathbf{u}}_i, \mathbf{x}_i)), \\ \mathbf{v}_j &\leftarrow \text{FC}(\Theta_v; \textsc{concat}(\widehat{\mathbf{v}}_j, \mathbf{y}_j)), \\ \mathbf{w}_k &\leftarrow \widehat{\mathbf{w}}_k \end{aligned} \tag{4}$$

here $\text{CONCAT}(\cdot)$ is the concatenation; $\{\mathbf{u}_i, \mathbf{v}_j, \mathbf{w}_k\} \in \mathbb{R}^r$ are new representations for user i, item j, and time k, respectively. $\mathbf{w}_k$ only contains the one-hot feature since there is no other feature for time. Two fully-connected layer with parameters Θ_u and Θ_v are applied to obtain more sophisticated representations for both users and items. Furthermore, by properly choosing the weights in Θ_u and Θ_v, we can reshape the vectors $\mathbf{u}_i$ and $\mathbf{v}_j$ to have the same dimension as $\mathbf{w}_k$. As such, $\{\mathbf{u}_i, \mathbf{v}_j, \mathbf{w}_k\}$ can be interpreted as *latent vectors* in CP tensor model as shown latter in Sec. 4.1.

4 The Proposed Method

To learn multi-aspect features, a straightforward method is to train a Multi-Layer Perceptron (MLP) [He *et al.*, 2017; Wu *et al.*, 2019] However, MLP fails to capture the triple-wise interactions among latent vectors due to its black-box structure, which are important in recommender systems. For example, a user will purchase an item if the item fits the user's preference and if the item is in season (at the right time).

To avoid such information loss, we propose NTM with two modules: a GCP layer and a tensorized MLP layer (Figure 2).

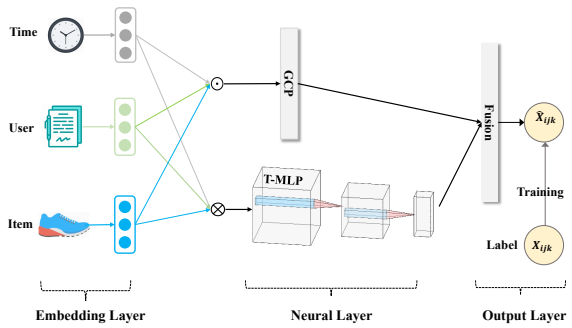

Figure 2: The architecture of the proposed NTM that consists of GCP and T-MLP to exploit the multi-aspect factors.

4.1 Generalized CP Tensor Factorization

As shown in Eq. (1), the CP model can interpret triple-wise interactions by its multilinear multiplication. Here we generalize the CP model to learn nonlinear feature interactions.

Given the embeddings $\{\mathbf{u}_i, \mathbf{v}_j, \mathbf{w}_k\} \in \mathbb{R}^r$ from Eq. (4), we design a novel GCP layer $\phi(\cdot)$, which contains a pooling operator that converts a set of embeddings to one vector:

$$\phi(\mathbf{u}_i, \mathbf{v}_j, \mathbf{w}_k) = \mathbf{u}_i \odot \mathbf{v}_j \odot \mathbf{w}_k \tag{5}$$

here $\odot$ is the element-wise product. Clearly, the GCP layer $\phi(\cdot)$ does not introduce extra model parameters, and more importantly, it can be efficiently computed with linear time. Then, we can project the hidden vector ϕ into an output layer:

$$\hat{\boldsymbol{\mathcal{X}}}_{ijk} = a_{out}(\mathbf{h}^T(\mathbf{u}_i \odot \mathbf{v}_j \odot \mathbf{w}_k)) \tag{6}$$

here a_{out} and $\mathbf{h}$ denote the activation function and weights.

Proposition 1. *The CP tensor factorization in Eq. (1) is a special case of generalized tensor factorization in Eq. (6).*

Proof. Let a_{out} be an identity function ($a_{out}(x) = x$), weight $\mathbf{h}$ be a uniform vector of 1 ($\mathbf{h} = [1, \ldots, 1]^T \in \mathbb{R}^r$), and $\mathbf{u}_i(t)$ be the t-th element in the column vector $\mathbf{u}_i$, we have:

$$\hat{\boldsymbol{\mathcal{X}}}_{ijk} = a_{out}(\mathbf{h}^T(\mathbf{u}_i \odot \mathbf{v}_j \odot \mathbf{w}_k)) = \sum_{t=1}^{r} \mathbf{u}_i(t)\mathbf{v}_j(t)\mathbf{w}_k(t)$$

which exactly recovers the CP factorization in Eq. (1) and the embedding size r now becomes the tensor CP rank. □

This is a very appealing connection, meaning that we can design a nonlinear CP tensor model. For example, if we adopt a nonlinear function a_{out} and allow $\mathbf{h}$ to be trained from data without uniform property, GCP provides a more powerful way to express the nonlinear interactions for multi-aspect factors. Here we implement the GCP by using the sigmoid function $\sigma(\cdot)$ and learning $\mathbf{h}$ with the pairwise loss function.

4.2 Tensorized Multi-layer Perceptron

To better capture the triple-wise feature interactions, we further propose to use an outer product on $\mathbf{u}_i$, $\mathbf{v}_j$, and $\mathbf{w}_k$:

$$\boldsymbol{\mathcal{E}} = \mathbf{u}_i \otimes \mathbf{v}_j \otimes \mathbf{w}_k \tag{7}$$

here $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{r \times r \times r}$ is a tensor feature map and $\otimes$ denotes the outer product. Our motivation for $\boldsymbol{\mathcal{E}}$ is straightforward. The $\boldsymbol{\mathcal{E}}$ captures more signals than element-wise product in Eq. (5) since it encodes *any* tripe-wise interactions. Such strategy is simple but widely used in deep learning [He *et al.*, 2018].

One can flatten $\boldsymbol{\mathcal{E}}$ into a vector and use a MLP to exploit its nonlinearity. However, the size of $\boldsymbol{\mathcal{E}}$ requires massive neurons. Assuming we have a map $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{64 \times 64 \times 64}$ and adopt a MLP with a half-size tower structure. As such, even the first layer requires $262,144 \times 131,072$ parameters in total.

To address this issue, we turn our attention to *n-mode Product* [Kolda and Bader, 2009]: Given a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \cdots \times I_n}$ and a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$, the n-mode product of $\boldsymbol{\mathcal{X}}$ and $\mathbf{U}$ is denoted as $\boldsymbol{\mathcal{X}} \times_n \mathbf{U}$ and is of size $\mathbb{R}^{I_1 \cdots \times I_{n-1} \times J \times I_{n+1} \cdots \times I_N}$. Inspired by this shrinking technique, we propose a Tensorized MLP to successively compress the tensor map $\boldsymbol{\mathcal{E}}$ by using a sequence of three weight matrices along *each mode* of $\boldsymbol{\mathcal{E}}$ with nonlinear transformation.

T-MLP block: We apply one T-MLP block as:

$$\boldsymbol{\mathcal{H}} = g(\boldsymbol{\mathcal{E}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} + \boldsymbol{\mathcal{B}}) \tag{8}$$

here $\boldsymbol{\mathcal{H}}$ is output in the hidden layer; $g(\cdot)$ is the activation function; $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ and $\boldsymbol{\mathcal{B}}$ are the weights and bias. At first glance, our T-MLP is very similar to Tucker in Eq. (2), but they are essentially different. Tucker requires the orthogonality of latent factors and it does not have the bias $\boldsymbol{\mathcal{B}}$ or activation function as in T-MLP.

As such, T-MLP significantly reduces the number of training parameters. For example, we can compress $\boldsymbol{\mathcal{E}}$ with size $\mathbb{R}^{64 \times 64 \times 64}$ to $\mathbb{R}^{16 \times 16 \times 16}$ by weights $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\} \in \mathbb{R}^{16 \times 64}$ and bias $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{16 \times 16 \times 16}$, resulting in $3 \times 16 \times 64 + 16^3$ parameters.

In back propagation, the gradient of weights and bias can be derived using chain rule. We give the gradient of Eq. (8) $w.r.t.$ $\mathbf{A}$ inside $g(\cdot)$. Let $\boldsymbol{\mathcal{J}} = \boldsymbol{\mathcal{E}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$ denote the term $w.r.t.$ $\mathbf{A}$, its matricized form is written as: $\boldsymbol{J}_{(1)} = \mathbf{A}\boldsymbol{E}_{(1)}(\mathbf{C} \circ \mathbf{B})^T$, where $\circ$ is the Kronecker product; $\boldsymbol{J}_{(1)}$ and $\boldsymbol{E}_{(1)}$ denote the mode-1 matricization of $\boldsymbol{\mathcal{J}}$ and $\boldsymbol{\mathcal{E}}$. We have:

$$\frac{\partial \boldsymbol{J}_{(1)}}{\partial \mathbf{A}} = [(\mathbf{C} \circ \mathbf{B})\boldsymbol{E}_{(1)}^T] \circ \mathbf{I}$$

here $\mathbf{I}$ is the identity matrix. Similar strategy can be used to compute the gradient for $\partial\boldsymbol{\mathcal{J}}/\partial\mathbf{B}$, $\partial\boldsymbol{\mathcal{J}}/\partial\mathbf{C}$, and $\partial\boldsymbol{\mathcal{J}}/\partial\boldsymbol{\mathcal{B}}$.

Stacking T-MLP blocks: Arguably two of the key contributors of the neural networks are its nonlinearity and the stacking of multiple layers. Likewise, we stack multiple T-MLP blocks with ReLU$(\cdot)$ as the activation function:

$$\boldsymbol{\mathcal{H}}_1 = \text{ReLU}(\boldsymbol{\mathcal{E}} \times_1 \mathbf{A}^{(0)} \times_2 \mathbf{B}^{(0)} \times_3 \mathbf{C}^{(0)} + \boldsymbol{\mathcal{B}}^{(0)}), \ldots,$$

$$\boldsymbol{\mathcal{H}}_L = \text{ReLU}(\boldsymbol{\mathcal{H}}_{L-1} \times_1 \mathbf{A}^{(L-1)} \times_2 \mathbf{B}^{(L-1)} \times_3 \mathbf{C}^{(L-1)} + \boldsymbol{\mathcal{B}}^{(L-1)}),$$

Dropout: Dropout [Srivastava *et al.*, 2014] is a regularization technique for neural networks to prevent overfitting. The idea is simple: randomly "turn off" neurons with probability ρ during training, and use all neurons when testing. To avoid overfitting, we also apply a dropout layer on the feature $\boldsymbol{\mathcal{E}}$, i.e., randomly dropping ρ percent of its elements.

Prediction layer: At last, the output of the last hidden layer $\boldsymbol{\mathcal{H}}_L$ is transformed to the final predictive score:

$$\hat{\boldsymbol{\mathcal{X}}}_{ijk} = \sigma(\mathbf{W}_o \times \text{Reshape}(\boldsymbol{\mathcal{H}}_L) + \mathbf{b}_o) \tag{9}$$

here Reshape$(\cdot)$ flattens $\boldsymbol{\mathcal{H}}_L$ into a vector. The output layer is a fully connected layer with the sigmoid function as predictor.

4.3 Joint Learning

We present our unified model NTM by joint learning GCP and T-MLP. Let $\boldsymbol{\phi}$, and $\boldsymbol{\mathcal{H}}_L$ denote the outputs of the last hidden layers of GCP and T-MLP. Then, we merge them as:

$$\hat{\boldsymbol{\mathcal{X}}}_{ijk} = \sigma(\mathbf{W}_f \times \text{CONCAT}(\boldsymbol{\phi}, \text{Reshape}(\boldsymbol{\mathcal{H}}_L)) + \mathbf{b}_f) \tag{10}$$

this fusion strategy is inspired by the well-known wide&deep learning [Cheng *et al.*, 2016; He *et al.*, 2017], which has the benefits of memorization and generalization. Our GCP can be regarded as a wide module whereas the T-MLP can be viewed as a deep module. The key difference is that our model aims to learn multi-aspect tensor data, while existing work can only learn two-dimensional matrices.

Pairwise Loss Training. To optimize the model, we opt for the margin-based ranking loss function [Socher *et al.*, 2013]:

$$\mathcal{L}(\Theta) = \sum_{(i,j,k)\in\mathcal{T}} [1 + f(i,j',k) - f(i,j,k)]_+ \tag{11}$$

where $[x]_+ = \max(x; 0)$; $f(\cdot)$ and Θ denote our predictive function and model parameters in Eq. (10); $\mathcal{T}$ denotes the training set, where each instance is a positive triplet i.e., $\boldsymbol{\mathcal{X}}_{ijk} = 1$. (i, j', k) is a negative triplet $w.r.t.$ (i, j, k), which can be randomly generated such that $\boldsymbol{\mathcal{X}}_{ij'k} = 0$.

Although random sampling is commonly used to generate negative triplets, it suffers from *zero loss* phenomenon [Wang *et al.*, 2018a]. For example, it can easily generate some women's dresses for a male user as negative samples, which contributes little for the model training.

4.4 Adversarial Training

To overcome above drawback, we adopt adversarial learning to provide high-quality negative samples in the training step [Wang *et al.*, 2017; Goodfellow *et al.*, 2014]. Concretely, the framework contains a *Discriminator* D and a *Generator* G. The generator tries to generate useful negative samples, while the discriminator aims to discriminate whether such samples come from G or true data.

Discriminator: The discriminator D aims to minimize:

$$\begin{aligned}\mathcal{L}_D(\Theta_D) = &\sum_{(i,j,k)\in\mathcal{T}} [1 + f_D(i,j',k) - f_D(i,j,k)]_+ \\ &(i,j',k) \sim p_G(i,j',k|i,j,k)\end{aligned} \tag{12}$$

here $p_G(i, j', k|i, j, k)$ is the probability for generating (i, j', k) (as Eq. (14)) ; f_D denotes the loss function, which is identical to f in Eq. (11). The only difference between these two is that Eq. (12) uses negative samples from the generator other than random sampling in Eq. (11).

Generator: The goal of the generator G is to maximize:

$$\mathcal{L}_G(\Theta_G) = \sum_{(i,j,k)\in\mathcal{T}} \mathbb{E}[f_D(i,j',k)]; \quad (i,j',k) \sim p_G(i,j',k|i,j,k) \tag{13}$$

The distribution $p_G(i, j', k|i, j, k)$ is:

$$p_G(i,j',k|i,j,k) = \frac{\exp(f_G(i,j',k))}{\sum_t \exp(f_G(i,j'_t,k))}; \quad (i,j'_t,k) \in \text{Neg}(i,j,k) \tag{14}$$

here f_G is the generator function, which is different from the f_D. Similar to [Wang *et al.*, 2018b], we feed embeddings of negative samples into a MLP, $f_G := \text{MLP}(\text{CONCAT}(\mathbf{u}_i, \mathbf{v}_{j'}, \mathbf{w}_k))$. Also, we generate $\text{Neg}(i, j, k)$ by uniformly sampling $N_r = 60$ triplets for each positive triplet, i.e., 60 items that user i has not interacted with at time k. We adopt the policy gradient to optimize the generator [Wang *et al.*, 2017]. Its gradient is:

$$\nabla_{\Theta_G}\mathcal{L}_G = \sum_{(i,j,k)\in\mathcal{T}} \mathbb{E}_{(i,j',k)\sim p_G}[f_D(i,j',k)\ \nabla_{\Theta_G} \log p_G(i,j',k|i,j,k)]$$

Similar to many GAN-based models, the generator and discriminator are trained alternatively toward their objectives.

Dataset	|User|	|Item|	|Interaction|	Density
Clothing	39,371	23,022	275,547	0.031%
Beauty	22,279	12,079	192,377	0.072%
Cell Phones	27,872	10,361	191,648	0.067%
Movies & TV	101,916	47,975	984,060	0.021%

Table 1: The statistics of Amazon datasets (after 2010)

4.5 Computational Complexity

The computational complexity of our model mainly comes from GCP and T-MLP modules. The complexity for the shallow GCP is $O(r)$, where r is the embedding size in Eq. (4). The complexity for the tensor-matrix multiplication in T-MLP is $L \cdot O(r^3 d^2)$, where d is the embedding size of hidden layers and L is the number of layers. In practice, L and d are typically small and the embedding size $r \ll \min(M, N, L)$ in recommendation problems. The overall complexity can be simplified as $O(|c| \cdot r^3)$, where $|c|$ is a constant.

5 Experiments

In this section, we aim to answer the following questions:

RQ1: Do our proposed models capture better nonlinear feature interactions than existing tensor factorizations?

RQ2: Does the proposed NTM outperform the state-of-the-art context-aware recommendation methods?

RQ3: What is the influence of various components (e.g., GCP, T-MLP, and sampling strategy) in the NTM?

5.1 Experimental Setup

Dataset. We consider four benchmark *Amazon*[1] datasets: *Clothing Shoes&Jewelry*, *Beauty*, *Cell Phones&Accessories*, and *Movies&TV*. The standard 5-core dataset for each category is utilized here. Similar to [Yu *et al.*, 2018], we remove the purchase record before 2010 and discretize the timestamps by weeks, leading to 237 time intervals. The statistics are listed in Table 1. Also, a positive triplet $\mathcal{X}_{ijk}$ means that the user i purchase the item j in the week k.

Baselines. (1) CP and Tucker [Kolda and Bader, 2009]: both are linear models. (2) nTucker [Zhe *et al.*, 2016]: a nonlinear Gaussian Tucker. (3) NCF [He *et al.*, 2017]: a neural collaborative filtering model. (4) CoSTCo [Liu *et al.*, 2019]: a CNN-based tensor model. (5) CMTF [Acar *et al.*, 2011]: a coupled tensor-matrix model. (6) VBPR [He and McAuley, 2016]: a Bayesian ranking method. (7) NTF [Wu

[1] http://jmcauley.ucsd.edu/data/amazon/index.html

et al., 2019]: a neural model with MLP. (8) DCFA [Yu *et al.*, 2018]: a tensor model with images and time factors. (9) JRL [Zhang *et al.*, 2017]: a model with images and reviews.

Parameter Settings. The parameters for the baselines are initialized as in the original papers and are then carefully tuned to achieve optimal performance. For NTM, its embedding size r in Eq. (4) is searched in $[16, 32, 64, 128]$. For T-MLP, we employ three hidden layers with dropout ratio 0.5 and each layer sequentially decreases the half size of the input. We implement NTM in PyTorch[2] with Adam [Kingma and Ba, 2015] as optimizer. We use grid-based search to find the best batch size within $[128, 256, 512, 1024]$ and the learning rate within $[0.0005, 0.001, 0.005, 0.01]$.

Evaluation Protocols. We randomly split the datasets into training, validation, and test sets in a 8:1:1 ratio. The validation set is used for tuning hyper-parameters and the final performance is conducted on the test set. We adopt two widely used metrics, Hit@k and NDCG@k [He *et al.*, 2017], to evaluate the performance. To avoid heavy computation on all triplets, we follow the strategy in [He *et al.*, 2017; Yu *et al.*, 2018]. For each user u_p at time t_p in the test set, we randomly sample 100 negative items, and rank these items with the ground-truth items. Based on the ranking results, Hit@k and NDCG@k can be evaluated.

5.2 Effect of Nonlinear Tensor Models (RQ1)

The proposed models GCP (Eq. (6)), T-MLP (Eq. (9)), and the unified NTM (Eq. (10)) are able to capture nonlinear feature interactions. In this part, we compare them with CP, Tucker, nTucker, NCF, and CoSTCo. These baselines are plain factorization machine without any information of user reviews and item images. For fair comparison, we only use the one-hot features in Eq. (3) and the loss function in Eq. (11) for our methods. We simply omit the *Movies&TV* dataset because its large size of data makes Tucker challenging to perform high-order SVD. Due to page limitation, we only show the performance at top-10 scenario, and similar trends can be observed at different top-k scenarios. Table 2 shows the results $w.r.t.$ Hit@10 and NDCG@10.

We can observe that nonlinear tensor models consistently outperform the multilinear CP and Tucker models. nTucker performs better than Tucker due to its nonlinear Gaussian process; GCP outperforms CP since it generalizes CP by using a neural layer. Second, CoSTCo is comparable to nTucker and NCF but worse than GCP. CoSTCo adopts a CNN to perform convolution. Nevertheless, many multi-aspect tensor data may not have spatial locality like images. Third, the performances of T-MLP are better than GCP about 3.1% on Hit@10 and 3.3% on NDCG@10, implying the benefit of a deeper tensorized MLP. Finally, NTM achieves the best performance, which demonstrates the high expressiveness of NTM by fusing both wide and deep components.

5.3 Performance Comparison (RQ2)

In this section, we compare the overall performance of NTM with the baselines of interest. The results of CP, Tucker, NCF, CoSTCo, and nTucker are omitted due to their inferior performances without auxiliary data. Figure 3 shows the top-k performance, where k is set to 5, 10, and 15.

Model	Clothing		Beauty		Phones	
	H@10	N@10	H@10	N@10	H@10	N@10
CP	0.546	0.191	2.239	0.713	2.134	0.745
Tucker	0.521	0.189	2.396	0.755	2.008	0.713
nTucker	0.816	0.283	2.607	0.909	2.544	0.902
NCF	0.965	0.407	3.012	1.135	2.598	1.178
CoSTCo	0.983	0.426	3.187	1.189	2.632	1.235
GCP	1.272	0.466	3.225	1.285	2.866	1.329
T-MLP	1.304	0.495	3.373	1.302	2.927	1.360
NTM	**1.353**	**0.512**	**3.401**	**1.326**	**3.012**	**1.404**

Table 2: Results of pure tensor models. (all number are percentage numbers without %, H is short for Hit and N is for NDCG).

As shown in Figure 3, we observe that deep neural models (e.g., NTF, DCFA, JRL, and NTM) substantially outperform the shallow methods (e.g, CMTF and VBPR), indicating the superiority of deep neural networks. More importantly, unlike requiring pre-defined feature matrices in the CMTF, these models allow end-to-end learning for contextual factors like user reviews and item images without manually crafted combinatorial features. Moreover, NTF performs much better than VBPR and CMTF, but falls behind DCFA, JRL, and NTM. This is because NTF only relies on the identity features of users and items. In contrast, DCFA alleviates this issue by incorporating visual features from item images. JRL and NTM further improve the quality of recommendations by joint exploring the contextual factors of user reviews and item images. Finally, NTM achieves the best performance over all the comparison methods, showing an average improvement of 24.6% and 25.7% than the state-of-the-art JRL model in terms of Hit@k and NDCG@k, respectively. The improvements of NTM mainly come from its wide and deep components for modeling triple-wise nonlinear interactions of multi-aspect factors. In addition, NTM adopts the effectiveness of a self-adversarial negative sampling technique to assist the model training (see Figure 5(b)).

5.4 Study of NTM (RQ3)

Ablation Study. We analyze the influence of each component via ablation studies. For each variant, we simply remove one module and compare the results with the default NTM.

Table 3 shows the performance of five variants on *Clothing* and *Beauty* datasets. Our results are summarized as follows: 1) *Remove GAN*: we find that our GAN-based sampling method can provide high-quality negative samples for NTM, resulting in a better performance. In contrast, randomly sampling negative samples may be effortlessly discriminated in the training step; 2) *Remove GCP*: the performances are inferior without GCP layer. Presumably this is because our wide component GCP serve an important role of generalization in the Wide&Deep learning; 3) *Remove T-MLP*: this variant substantially decreases the overall performance with a large margin, verifying the effectiveness of the T-MLP in capturing useful interactions from feature outer product space; 4) *Remove image CNN or Review CNN*: not surprisingly, deleting the module of either user reviews or item images significantly

[2]https://pytorch.org/

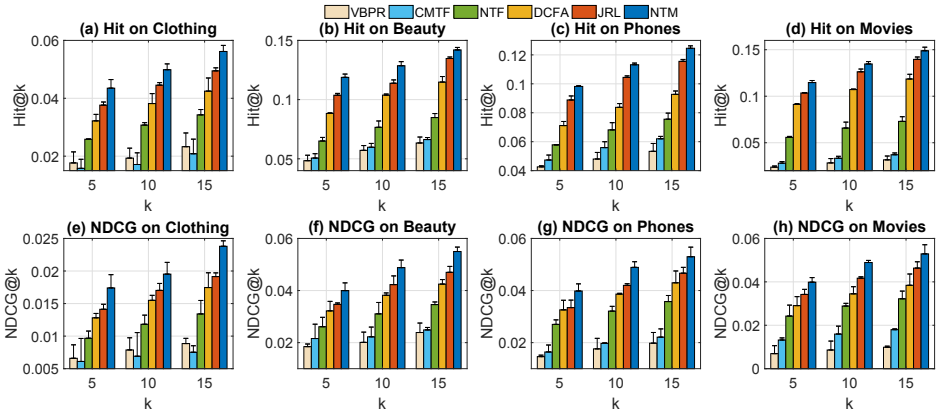

Figure 3: Performance of Top-k recommendations in terms of Hit@k (3a-3d) and NDCG@k (3e-3h) with error bars.

Ablation Study	Clothing		Beauty	
	Hit@10	NDCG@10	Hit@10	NDCG@10
NTM	4.992	1.986	12.931	4.902
Remove GAN	4.670	1.803	12.113	4.712
Remove GCP	4.691	1.831	12.275	4.763
Remove T-MLP	4.023↓	1.691↓	10.615↓	4.124↓
Remove Image CNN	3.881↓	1.563↓	10.218↓	3.853↓
Remove Review CNN	3.782↓	1.547↓	10.324↓	3.902↓

Table 3: Ablation analysis on our model (all numbers are percentage without %, '↓' means a severe performance drop).

hurts the overall performance. This implies that both review-based CNN features and image-based CNN features are important to understand users' preference.

Embedding Size of NTM. The embedding size r in Eq. (4) is a key hyper-parameter since it affects the representation ability of NTM. Here we vary embedding size r within $[16, 32, 64, 128]$ and compare with the JRL model in terms of Hit@10 and NDCG@10 on *Clothing* dataset. As shown in Figure 4(a)-4(b), we observe that both NTM and JRL models benefit from a large embedding size. In general, NTM achieves satisfactory performance with $r \geq 64$. Assessments on other datasets are similar and omitted here.

Layers of NTM. We also conduct experiments to see whether using a deeper network architecture is beneficial to the recommendation task. To this end, we vary the number of T-MLP blocks in the NTM within $L = [2, 3, 4, 5]$. As shown in Figure 4(c)-4(d), stacking more layers gradually enhances the recommendation performance. We attribute the improvements to the usage of stacking more nonlinear layers to model complex user-item-time interactions.

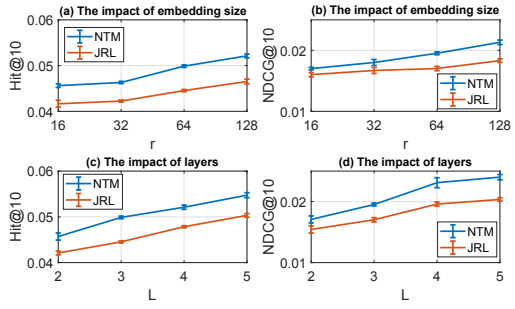

Figure 4: The impact of embedding sizes r (4a-4b), and number of layers L (4c-4d). on *Clothing* dataset.

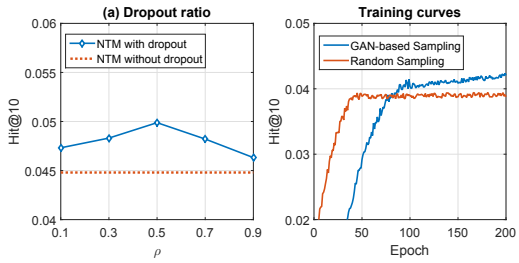

Figure 5: (a) The impact of dropout ratio. (b) Training loss $w.r.t$ Eq. (11) and Eq. (12). Both are for *Clothing* datasets.

Dropout and Adversarial Training. Figure 5(a) shows the performances of NTM $w.r.t.$ dropout ratio. Our results show that dropout offers better performance. Specifically, using a dropout ratio $\rho \approx 0.5$ achieves an optimal accuracy.

The training progress $w.r.t.$ Eq. (11) and Eq. (12) are also shown in Figure 5(b). The performance of our GAN-based sampling method is always in increasing trends, verifying the effectiveness of its high-quality negative training samples. In contrast, random negative samples are often too trivial to fit the model, which possibly leads to zero loss phenomenon [Wang *et al.*, 2018a].

6 Conclusion

In this work, we propose a nonlinear tensor machine that combines deep neural networks and tensor algebra to investigate the impacts of multi-aspect factors. We also develop a self-adaptive negative sampling strategy to assist the model training. Experimental results demonstrate the effectiveness of our proposed model for context-aware recommendations. For future work, we will incorporate more contextual factors such as location and user social network.

Acknowledgments

This work has been supported in part by NSF CCF1815139.

References

[Acar *et al.*, 2011] Evrim Acar, Tamara G Kolda, and Daniel M Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *Computing Research Repository (CORR)*, 2011.

[Bhargava *et al.*, 2015] Preeti Bhargava, Thomas Phan, Jiayu Zhou, and Juhan Lee. Who, what, when, and where: Multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data. In *WWW*, 2015.

[Catherine and Cohen, 2017] Rose Catherine and William Cohen. Transnets: Learning to transform for recommendation. In *RecSys*, 2017.

[Chen and Li, 2017] Huiyuan Chen and Jing Li. Learning multiple similarities of users and items in recommender systems. In *ICDM*, 2017.

[Chen and Li, 2018] Huiyuan Chen and Jing Li. Drugcom: Synergistic discovery of drug combinations using tensor decomposition. In *ICDM*, 2018.

[Chen and Li, 2019] Huiyuan Chen and Jing Li. Adversarial tensor factorization for context-aware recommendation. In *RecSys*, 2019.

[Cheng *et al.*, 2016] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *DLRS*, 2016.

[Cohen *et al.*, 2016] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *COLT*, 2016.

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

[Goodfellow *et al.*, 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

[He and McAuley, 2016] Ruining He and Julian McAuley. Vbpr: visual bayesian personalized ranking from implicit feedback. In *AAAI*, 2016.

[He *et al.*, 2016] Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. Vista: A visually, socially, and temporally-aware model for artistic recommendation. In *RecSys*, 2016.

[He *et al.*, 2017] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, 2017.

[He *et al.*, 2018] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. In *IJCAI*, 2018.

[Jia *et al.*, 2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *MM*, 2014.

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

[Kim *et al.*, 2016] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *RecSys*, 2016.

[Kim, 2014] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[Kingma and Ba, 2015] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

[Lebedev *et al.*, 2015] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*, 2015.

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

[Novikov *et al.*, 2015] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *NeurIPS*, 2015.

[Socher *et al.*, 2013] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*, 2013.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[Wang *et al.*, 2017] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, 2017.

[Wang *et al.*, 2018a] Peifeng Wang, Shuangyin Li, and Rong Pan. Incorporating gan for negative sampling in knowledge representation learning. In *AAAI*, 2018.

[Wang *et al.*, 2018b] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. Neural memory streaming recommender networks with adversarial training. In *KDD*, 2018.

[Wu *et al.*, 2019] Xian Wu, Baoxu Shi, Yuxiao Dong, Chao Huang, and Nitesh V Chawla. Neural tensor factorization for temporal interaction learning. In *WSDM*, 2019.

[Xu *et al.*, 2012] Zenglin Xu, Feng Yan, and Yuan Qi. Infinite tucker decomposition: Nonparametric bayesian models for multiway data analysis. In *ICML*, 2012.

[Yu *et al.*, 2018] Wenhui Yu, Huidi Zhang, Xiangnan He, Xu Chen, Li Xiong, and Zheng Qin. Aesthetic-based clothing recommendation. In *WWW*, 2018.

[Zhang *et al.*, 2017] Yongfeng Zhang, Qingyao Ai, Xu Chen, and W Bruce Croft. Joint representation learning for top-n recommendation with heterogeneous information sources. In *CIKM*, 2017.

[Zhe *et al.*, 2016] Shandian Zhe, Kai Zhang, Pengyuan Wang, Kuang-chih Lee, Zenglin Xu, Yuan Qi, and Zoubin Ghahramani. Distributed flexible nonlinear tensor factorization. In *NeurIPS*, 2016.

[Zheng *et al.*, 2017] Lei Zheng, Vahid Noroozi, and Philip S Yu. Joint deep modeling of users and items using reviews for recommendation. In *WSDM*, 2017.