Latent Cross: Making Use of Context in Recurrent Recommender Systems

Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li*, Vince Gatto, Ed H. Chi
Google, Inc.
Mountain View, California
{alexbeutel, pcovington, sagarj, canxu, vgatto, edchi}@google.com, vena900620@gmail.com

ABSTRACT

The success of recommender systems often depends on their ability to understand and make use of the context of the recommendation request. Significant research has focused on how time, location, interfaces, and a plethora of other contextual features affect recommendations. However, in using deep neural networks for recommender systems, researchers often ignore these contexts or incorporate them as ordinary features in the model.

In this paper, we study how to effectively treat contextual data in neural recommender systems. We begin with an empirical analysis of the conventional approach to context as features in feed-forward recommenders and demonstrate that this approach is inefficient in capturing common feature crosses. We apply this insight to design a state-of-the-art RNN recommender system. We first describe our RNN-based recommender system in use at YouTube. Next, we offer "Latent Cross," an easy-to-use technique to incorporate contextual data in the RNN by embedding the context feature first and then performing an element-wise product of the context embedding with model's hidden states. We demonstrate the improvement in performance by using this Latent Cross technique in multiple experimental settings.

ACM Reference Format:
Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, Ed H. Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining , February 5–9, 2018, Marina Del Rey, CA, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3159652.3159727

1 INTRODUCTION

Recommender systems have long been used for predicting what content a user would enjoy. As online services like Facebook, Netflix, YouTube, and Twitch continue to grow, having a high quality recommender system to help users sift through the expanding and increasingly diverse content becomes ever-more important.

Much of the research in recommender systems has focused on effective machine learning techniques — how to best learn from users actions, e.g., clicks, purchases, watches, and ratings. In this effort, there is a large body of research on collaborative filtering and recommendation algorithms, including matrix factorization during the Netflix Prize [24, 28, 30, 35], local focused models [5, 9, 31], and more recently deep learning [11, 36].

[1]Jia Li is affiliated with the University of Illinois at Chicago, but worked on this project while interning at Google.

WSDM'18, February 5–9, 2018, Marina Del Rey, CA, USA
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5581-0/18/02.
https://doi.org/10.1145/3159652.3159727

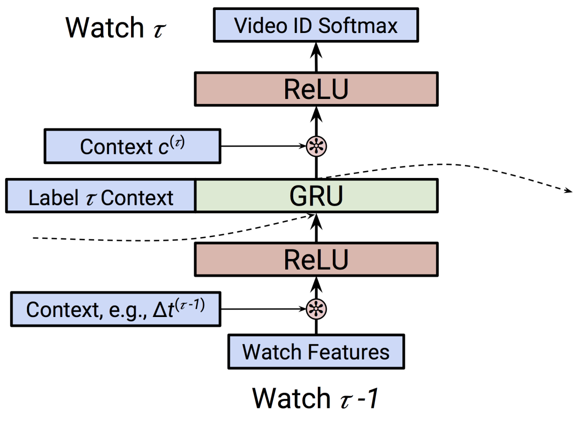

Figure 1: Overall structure of our recommender system, including "latent crosses" of context features throughout.

In parallel, and increasingly prominently, there is an understanding of the importance of modeling the *context* of a recommendation – not just the user who is looking for a video to watch, but also the time of day, the location, the user's device, etc. Many of these models have been proposed in the factorization setting, such as with tensor factorization for location [17], unfolding tensors for different types of user actions [46], or hand-crafted features about the effect of time [29].

As deep learning has grown in prominence, how to incorporate these contextual features in neural recommender systems has been less directly explored. Previous work on deep neural network (DNN) recommender systems has largely relied on modeling context as direct features in the model or having a multi-task objective [11]. One notable exception is the use of recurrent neural networks (RNNs) to model temporal patterns [25, 39, 43]. In this paper, we bridge the contextual collaborative filtering literature and neural recommender literature. We explore how to make use of contextual data in deep neural recommenders (particularly in RNN models) and demonstrate that prevailing techniques miss a significant amount of the information in these features.

We explore the ability to make use of contextual data in an RNN-based recommender system used at YouTube. As with most production settings, we have a significant amount of contextual data that is important to include: request and watch time, the type of device, and the page on the website or mobile app. In this paper, first, we offer a theoretical interpretation of the limitations of modeling context as direct features, particularly using feed-forward neural networks as the example baseline DNN approach. We then offer an easy-to-use technique to incorporate these features that results in improved prediction accuracy, even within a more complicated RNN model.

Our contributions are:

- **First-Order Challenges:** We demonstrate the challenges of first-order neural networks to model low-rank relationships.
- **Production Model:** We describe how we have constructed a large-scale RNN recommender system for YouTube.
- **Latent Cross:** We offer a simple technique, called the "Latent Cross", to include contextual features more expressively in our model. Specifically, latent cross performs an element-wise product between the context embedding and the neural network hidden states.
- **Empirical Results:** We offer empirical results verifying that our approach improves recommendation accuracy.

2 RELATED WORK

We begin with a survey of the various related research. An overview can be seen in Table 1.

Contextual Recommendation. A significant amount of research has focused on using contextual data during recommendation. In particular, certain types of contextual data have been explored in depth, where as others have been treated abstractly. For example, temporal dynamics in recommendation have been explored widely [6]. During the Netflix Prize [4], Koren [29] discovered the significant long-ranging temporal dynamics in the Netflix data set and added temporal features to his Collaborative Filtering (CF) model to account for these effects. Researchers have also explored the how preferences evolve in shorter time-scales, e.g., sessions [39]. More general abstractions have been used to model preference evolution for recommendation such as point processes [15] and recurrent neural networks [43]. Similarly, modeling user actions along with geographical data has been widely explored with probabilistic models [2, 8], matrix factorization [32], and tensor factorization [17]. A variety of methods have built on matrix and tensor factorization for cross domain learning [45, 46]. Methods like factorization machines [34] and other contextual recommenders [22, 37, 48] have provided generalizations of these collaborative filtering approaches.

Neural Recommender Systems. As neural networks have grown in popularity for computer vision and natural language processing (NLP) tasks, recommender systems researchers have begun applying DNNs to recommendation. Early iterations focused on directly applying the collaborative filtering intuition to neural networks, such as through an auto-encoder [36] or joint deep and CF models [20]. More complex networks have been devised to incorporate a wider variety of input features [11]. Cheng et al. [7] handles this problem through a linear model to handle interactions among contextual features, outside the DNN portion of the model.

There has been a recent growth in using *recurrent* neural networks for recommendation [21, 25, 39, 43]. [25, 43] include temporal information as features and supervision in their models, and [41] includes general context features. However, in both cases, these features are concatenated with input, which we show provides limited benefit. Concurrent and independent research [49] improved LSTMs by multiplicatively incorporating temporal information, but did not generalize this to other context data.

Second-order Neural Networks. A major thrust of this paper is the importance of multiplicative relations in neural recommenders. These second-order units show up in a few places in neural networks. Recurrent units, e.g., LSTMs [23] and GRUs [10], are common second-order units with gating mechanisms that use an element-wise multiplication. A more complete tutorial on recurrent networks can be found in [18].

Additionally, softmax layers at the top of networks for classification are explicitly bi-linear layers between the embedding produced by the DNN and embeddings of the label classes. This technique has been extended in multiple papers to include user-item bi-linear layers on top of DNNs [20, 41, 43, 47].

Similar to the technique described in this paper is a body of work on multiplicative models [27, 44]. These multiplicative structures have most commonly been used in natural language processing as in [14, 27]. The NLP approach was applied to personalized modeling of reviews [40] (with a slightly different mathematical structure). Recently, [25] uses the multiplicative technique not over contextual data but directly over users, similar to a tensor factorization. PNN [33] and NFM [19] push this idea to an extreme, taking multiplying all pairs of features at the input and either concatenating or averaging the results before passing through a feed-forward network. The intuition of these models is similar to ours, but differ in that we focus on the relationship between contextual data and user actions, our latent crossing mechanism can be and is applied throughout the model, and we demonstrate the importance of these interactions even within an RNN recommender system.

More complex model structures like attention models [3], memory networks [38], and meta-learning [42] also rely on second-order relations and are increasingly popular. Attention models, for instance, use attention vectors that modulate the hidden states with a multiplication. However, these methods are significantly more complex structurally and are generally found to be much more difficult to train. In contrast, the latent cross technique proposed in this paper we found to be easy to train and effective in practice.

3 MODELING PRELIMINARIES

We consider a recommender system in which we we have a database $\mathcal{E}$ of events e which are k-way tuples. We consider e_ℓ to refer to the ℓth value in the tuple and $e_{\bar{\ell}}$ to refer to the other $k-1$ values in the tuple.

For example, the Netflix Prize setting would be described by tuples $e \equiv (i, j, R)$ where user i gave movie j a rating of R. We may also have context such as time and device such that $e \equiv (i, j, t, d)$ where user i watched video j at time t on device type d. Note, each value can be either discrete categorical variables, as in there are N users where $i \in \mathcal{I}$, or continuous, e.g., t is a unix timestamp. Continuous variables are not uncommonly discretized as a preprocessing step, such as to convert t to only the day on which the event took place.

With this data, we can frame recommender systems as trying to predict one value of the event given the others. For example, the Netflix Prize said for a tuple $e = (i, j, R)$, use (i, j) predict R. From a

	Latent Cross RNN	FM [22, 34]	CF [29, 32, 48]	TF [17, 37, 46]	RRN [21, 43]	YT [11]	NCF [20]	PNN [19, 33]	Conext-RNN [41]	Time-LSTM [49]
Neural Network	✓	×	×	×	✓	✓	✓	✓	✓	✓
Models Sequences	✓	×	×	×	✓	×	×	×	✓	✓
Uses Context	✓	✓	✓	✓	*time*	✓	×	✓	✓	*time*
Multiplicative User/Item	✓	✓	✓	✓	✓	×	✓	✓	✓	✓
Multiplicative Context	✓	✓	✓	✓	×	×	×	✓	×	*time*

Table 1: Relationship with related recommenders: We bridge the intuition and insights from contextual collaborative filtering with the power of recurrent recommender systems.

Symbol	Description
e	Tuple of k values describing an observed event
e_ℓ	Element ℓ in the tuple
$\mathcal{E}$	Set of all observed events
u_i, v_j	Trainable embeddings of user i and item j
$\mathcal{X}_i$	All events for user i
$\mathcal{X}_{i,t}$	All events for user i before time t
$e^{(\tau)}$	Event at step τ in a particular sequence
$\langle\cdot\rangle$	k-way inner product
$*$	Element-wise product
$f(\cdot)$	An arbitrary neural network

Table 2: Notation

machine learning perspective, we can split our tuple e into features x and label y such that $x = (i, j)$ and label $y = R$.

We could further re-frame the recommendation problem as predicting which video a user will watch at a given time by defining $x = (i, t)$ and $y = j$. Note, again, depending on if the label is categorical random value, e.g., video ID, or real value, e.g., rating, the machine learning problem is either a classification or regression problem, respectively.

In factorization models, all input values are considered to be discrete, and are embedded and multiplied. When we "embed" a discrete value, we learn a dense latent representation, e.g., user i is described by dense latent vector u_i and item j is described by dense latent vector v_j. In matrix factorization models, predictions are generally based on $u_i \cdot v_j$. In tensor factorization models, predictions are based on $\sum_r u_{i,r} v_{j,r} w_{t,r}$, where w_t would be a dense vector embedding of time or some other contextual feature. See factorization machines [34] for a clean and clear abstraction for these types of models. For notational simplicity we will consider $\langle\cdot\rangle$ to represent a multi-dimensional inner-product, i.e., $\langle u_i, v_j, w_t\rangle = \sum_r u_{i,r} v_{j,r} w_{t,r}$.

Neural networks typically also embed discrete inputs. That is, given an input (i, j) the input to the network will be $x = [u_i; v_j]$ where u_i and v_j are concatenated and as before are parameters that are trainable (in the case of neural networks, through back-propagation). Thus, we consider neural networks of the form $e_\ell = f(e_{\bar{\ell}})$, where the network takes all but one value of the tuple as the input and we train f to predict the last value of the tuple. We will later expand this definition to allow the model to take relevant previous events also as inputs to the network, as in sequence models.

4 MOTIVATION: CHALLENGES IN FIRST-ORDER DNN

In order to understand how neural recommenders make use of concatenated features, we begin by inspecting the typical building blocks of these networks. As alluded to above, neural networks, especially feed-forward DNNs, are often built on first-order operations. More precisely stated, neural networks often rely on matrix-vector products of the form Wh where W is a learned weight matrix and h is an input (either an input to the network, or the output of a previous layer). In feed-forward networks, fully-connected layers are generally of the form:

$$h_\tau = g(W_\tau h_{\tau-1} + b_\tau) \quad (1)$$

where g is an element-wise operation such as a sigmoid or ReLU, $h_{\tau-1}$ is the output of the last layer, and W_τ and b_τ are learned parameters. We consider this to be a first-order cell in that $h_{\tau-1}$, which is a k-dimensional vector, only has its different values added together, with weights based on W, but never multiplied together.

Although neural networks with layers like these have been shown to be able to approximate any function, their core computation is structurally significantly different from the past intuition on collaborative filtering. As described above, matrix factorization models take the general form $u_i \cdot v_j$, resulting in the model learning low-rank relationships between the different types of inputs, i.e., between users, items, time, etc. Given that low-rank models have been successful in recommender systems, we ask the following question: How well can neural networks with first-order cells model low-rank relations?

4.1 Modeling Low-Rank Relations

To test whether first-order neural networks can model low-rank relations, we generate synthetic low-rank data and study how well different size neural networks can fit that data. To be more precise, we consider an m-mode tensor where each dimension is of size N. For the mN discrete features we generate random vectors u_i of length r using a simple equation:

$$u_i \sim \mathcal{N}\left(\mathbf{0}, \frac{1}{r^{1/2m}}\mathbf{I}\right) \quad (2)$$

The result is that our data is a rank r matrix or tensor with approximately the same scale (with mean of 0, and an empirical variance close to 1). As an example, with $m = 3$, we can use these embeddings to represent events of the form $(i, j, t, \langle u_i, u_j, u_t\rangle)$.

We try to fit models of different sizes using this data. In particular, we consider a model with the discrete features embedded and concatenated as inputs. The model has one hidden layer with ReLU

Hidden Layer	$r = 1, m = 2$	$r = 1, m = 3$	$r = 2, m = 3$
1	0.42601	0.27952	0.287817
2	0.601657	0.57222	0.472421
5	0.997436	0.854734	0.717233
10	0.999805	0.973214	0.805508
20	0.999938	0.996618	0.980821
30	0.999983	0.99931	0.975782
50	0.999993	0.999738	0.997821
100	0.999997	0.999928	0.99943

Table 3: Pearson correlation for different width models when fitting low-rank data.

activation, as is common in neural recommender systems, followed by a final linear layer. The model is programmed in TensorFlow [1], trained with mean squared error loss (MSE) using the Adagrad optimizer [16], and trained to convergence. We measure and report the model's accuracy by the Pearson correlation (R) between the training data and the model's predictions. We use the Pearson correlation so that it is invariant to slight differences in variance of the data. We report the accuracy against the training data because we are testing how well these model structures fit low-rank patterns (i.e., not even whether they can generalize from it).

To model low-rank relations, we want to see how well the model can approximate individual multiplications, representing interaction between variables. All data is generated with $N = 100$. With $m = 2$, we examine how large the hidden layer must be to multiply two scalars, and with $m = 3$ we examine how large the hidden layer must be to multiply three scalars. We use $r \in \{1, 2\}$ to see how size of the model grows as more multiplications are needed. We embed each discrete feature as a 20-dimensional vector, much larger than r (but we found the model's accuracy to be independent of this size). We test with hidden layers $\in \{1, 2, 5, 10, 20, 30, 50\}$.

Empirical Findings. As can be seen in Table 3 and Figure 2, we find that the model continually approximates the data better as hidden layer size grows. Given the intuition that the network is approximating the multiplication, a wider network should give a better approximation. Second, we observe that, as we increase the rank r of the data from 1 to 2, we see the hidden layer size approximately doubles to get the same accuracy. This too matches our intuition, as increasing r means there are more interactions to add—something the network can easily do exactly.

More interestingly, we find that even for $r = 1$ and $m = 2$, it takes a hidden layer of size 5 to get a "high" accuracy estimate. Considering collaborative filtering models will often discover rank 200 relations [28], this intuitively suggests that real world models would require very wide layers for a single two-way relation to be learned.

Additionally, we find that modeling more than 2-way relations increases the difficulty of approximating the relation. That is, when we go from $m = 2$ to $m = 3$ we find that the model goes from needing a width 5 hidden layer to a width 20 hidden layer to get an MSE of approximately 0.005 or a Pearson correlation of 0.99.

In summary, we observe that ReLU layers can approximate multiplicative interactions (crosses) but are quite inefficient in doing so.

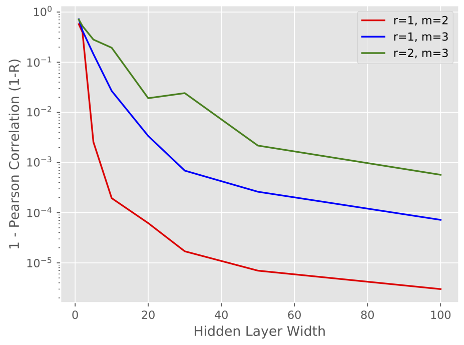

Figure 2: ReLU layers can learn to approximate low-rank relations, but are inefficient in doing so.

This motivates the need for models that can more easily express, and deal with, multiplicative relations. We now turn our attention to using an RNN as a baseline; this is a stronger baseline in that it can better express multiplicative relations compared to feed-forward DNNs.

5 YOUTUBE'S RECURRENT RECOMMENDER

With the above analysis as motivation, we now describe the improvements to YouTube's RNN recommender system. RNNs are notable as a baseline model because they are already second-order neural networks, significantly more complex than the first-order models explored above, and are at the cutting edge of dynamic recommender systems.

We begin with an overview of the RNN recommender we built for YouTube and then in Section 6 describe how we improve it to better make use of contextual data.

5.1 Formal Description

In our setting, we observe events of the form user i has watched video j (uploaded by user $\psi(j)$) at time t. (We will later introduce additional contextual features.) In order to model the evolution of user preferences and behavior, we use a recurrent neural network (RNN) model, where the input to the model is the set of events for user $\mathcal{X}_i = \{e = (i, j, \psi(j), t) \in \mathcal{E} | e_0 = i\}$. We will use $\mathcal{X}_{i,t}$ to denote all watches before t for user $\mathcal{X}_i$:

$$\mathcal{X}_{i,t} = \{e = (i, j, t) \in \mathcal{E} | e_0 = i \wedge e_3 < t\} \subset \mathcal{X}_i. \quad (3)$$

The model is trained to produce sequential prediction $\Pr(j|i, t, \mathcal{X}_{i,t})$, i.e., the video j that user i will watch at a given time t based on all watches before t. For the sake of simplicity, we will use $e^{(\tau)}$ to denote the τth event in the sequence, $x^{(\tau)}$ to denote the transformed input for $e^{(\tau)}$, and $y^{(\tau)}$ to denote the label trying to be predicted for the τth event. In the example above, if $e^{(\tau)} = (i, j, \psi(j), t)$ and $e^{(\tau+1)} = (i, j', \psi(j'), t')$ then the input $x^{(\tau)} = [v_j; u_{\psi(j)}; w_t]$, which is used to predict $y^{(\tau+1)} = j'$, where v_j is the video embedding, $u_{\psi(j)}$ is the uploader embedding, and w_t is the context embedding.

When predicting $y^{(\tau)}$, we of course cannot use the label of the corresponding event $e^{(\tau)}$ as an input, but we can use context from $e^{(\tau)}$, which we will denote by $c^{(\tau)}$, e.g., $c^{(\tau)} = [w_t]$.

5.2 Structure of the Baseline RNN Model

A diagram of our RNN model can be seen in Figure 1, and described below. Recurrent neural networks model a sequence of actions. With each event $e^{(\tau)}$, the model takes a step forward, processing $x^{(\tau)}$ and updating a hidden state vector $z^{(\tau-1)}$. To be more precise, each event is first processed by a neural network $h_0^{(\tau)} = f_i(x^{(\tau)})$. In our setting, this will either be an identity function or fully-connected ReLU layers.

The recurrent part of the network is a function $h_1^{(\tau)}, z^{(\tau)} = f_r(h_0^{(\tau)}, z^{(\tau-1)})$. That is, we use a recurrent cell, such as an LSTM [23] or GRU [10], that takes as an input the state from the previous step and the transformed input $f_i(x^{(\tau)})$.

To predict $y^{(\tau)}$, we use $f_o(h_1^{(\tau-1)}, c^{(\tau)})$, which is another trainable neural network that produces a probability distribution over possible values of $y^{(\tau)}$. In our setting, this network takes the output of the RNN as its input and context of the upcoming prediction, and finally ends with a softmax layer over all videos. This network can include multiple fully-connected layers.

5.3 Context Features

Core to the success of this model is the incorporation of contextual data beyond just the sequence of videos watched. We discuss below how we utilize these features.

TimeDelta. In our system, incorporating time effectively was highly valuable to the accuracy of our RNN. Historically, time context has been incorporated into collaborative filtering models in a variety of ways. Here we use an approach we call *timedelta*:

$$\Delta t^{(\tau)} = \log\left(t^{(\tau+1)} - t^{(\tau)}\right) \tag{4}$$

That is, when considering event $e^{(\tau)}$ we consider how long until the next event or prediction to be made. This is essentially equivalent to time representation described in [25] and [49].

Software Client. YouTube videos can be watched on a variety of devices: in browser, iOS, Android, Roku, Chromecast, etc. Treating these contexts as equivalent misses relevant correlations. For example, users are probably less likely to watch a full-length feature film on their phone than through a Roku device. Similarly, short videos like trailers may be relatively more likely to be watched on a phone. Modeling the software client, and in particular how it interacts with the watch decisions, is important.

Page. We also record where a watch initiated from in our system. For example, we distinguish between watches that starts from the homepage (i.e. Home Page Watches) and watches that initiate as recommended follow-up watches once a user is already watching a video (i.e. Watch Next Watches). This is important as watches from the homepage may be more open to new content, whereas watches following a previous watch may be due to users wanting to dig into a topic more deeply.

Pre- and Post-Fusion. We can use these context features, which we collectively refer to as $c^{(\tau)}$, as direct inputs in two ways. As can be seen in Figure 1, we can include context as an input at the bottom of the network, or concatenated with the output of the RNN cell. We refer to the inclusion of context features before the RNN as pre-fusion, and the inclusion of context features after the RNN cell as post-fusion [12]. Although possibly a subtle point, this decision can have a significant effect on the RNN. In particular, by including a feature through pre-fusion, that feature will affect the prediction through how it modifies the state of the RNN. However, by including a feature through post-fusion, that feature can more directly have an effect on the prediction at that step.

To manage this, when predicting $y^{(\tau)}$, we generally use $c^{(\tau)}$ as a post-fusion feature, and use $c^{(\tau-1)}$ as a pre-fusion feature. This means that $c^{(\tau-1)}$ will effect the RNN state but $c^{(\tau)}$ will be used for predicting $y^{(\tau)}$. Subsequently, at the next step when predicting $y^{(\tau+1)}$, $c^{(\tau)}$ will now be a pre-fusion feature effecting the state of the RNN from that time forward.

5.4 Implementation & Training

Our model is implemented in TensorFlow [1] and trained over many distributed workers and parameter servers. The training uses one of the available back-propagated mini-batch stochastic gradient descent algorithms, either Adagrad [16] or ADAM [26]. During training we use as supervision the 100 most recent watches during the period $(t_0 - 7 \text{ days}, t_0]$, where t_0 is the time of training. This generally prioritizes recent watches because the behavior is more similar to the prediction task when the learned model will be applied to live traffic.

Due to the large number of videos available, we limit our set of possible videos to predict as well as the number of uploaders of those videos that we model. In the experiments below, these sets range in size from 500,000 to 2,000,000. The softmax layer, which covers this set of candidate videos, is trained using sampled softmax with 20,000 negative samples per batch. We use the predictions of this sampled softmax in the cross entropy loss against all labels.

6 CONTEXT MODELING WITH THE LATENT CROSS

As should be clear in the above description of our baseline model, the use of contextual features is typically done as concatenated inputs to simple fully-connected layers. However, as we explained in Section 4, neural networks are inefficient in modeling the interactions between concatenated input features. Here we propose a simple alternative.

6.1 Single Feature

We begin with the case where we have a single context feature that we want to include. For sake of clarity, we will use time as an example context feature. Rather than incorporating the feature as another input concatenated with the other relevant features, we perform an element-wise product in the middle of the network. That is, we perform:

$$h_0^{(\tau)} = (\mathbf{1} + w_t) * h_0^{(\tau)} \tag{5}$$

where we initialize w_t by a 0-mean Gaussian (note, $w = \mathbf{0}$ is an identity). This can be interpreted as the context providing a mask or attention mechanism over the hidden state. However, it also enables low-rank relations between the input previous watch and the time. Note, we can also apply this operation after the RNN:

$$h_1^{(\tau)} = (\mathbf{1} + w_t) * h_1^{(\tau)}. \tag{6}$$

The technique offered in [27] can be viewed as a special case where multiplicative relations are included at the very top of the network along with the softmax function to improve NLP tasks. In that case, this operation can be perceived as a tensor factorization where the embedding for one modality is produced by a neural network.

6.2 Using Multiple Features

In many cases we have more than one contextual feature that we want to include. When including multiple contextual features, say time t and device d, we perform:

$$h^{(\tau)} = (\mathbf{1} + w_t + w_d) * h^{(\tau)} \tag{7}$$

We use this form for a few different reasons: (1) By initializing both w_t and w_d by 0-mean Gaussians, the multiplicative term has a mean of 1 and thus can similarly act as a mask/attention mechanism over the hidden state. (2) By adding these terms together we can capture 2-way relations between the hidden state and each context feature. This follows the perspective taken in the design of factorization machines [34]. (3) Using a simple additive function is easy to train. A more complex function like $w_t * w_d * h^{(\tau)}$ will increase the non-convexity significantly with each additional contextual feature. Similarly we found learning a function $f([w_t; w_d])$ to be more difficult to train and to give worse results. An overview of including these features in a model can be seen in Figure 1.

Efficiency. We note that one significant benefit of using latent crosses is their simplicity and computational efficiency. With N context features and d-dimensional embeddings, the latent cross can be computed in $O(Nd)$ and does not increase the width of the subsequent layers.

7 EXPERIMENTS

We perform two sets of experiments. The first is on a restricted dataset where time is the only contextual feature, and we compare several model families. In the second set of experiments we use our production model and explore the relative improvements based on how we incorporate context features.

7.1 Comparative Analysis

7.1.1 Setup. We begin with an explanation of our experimental setup.

Dataset and Metrics. We use a dataset with sequences of watches for hundreds of millions users. The users are split into training, validation and test sets, with both validation and test sets having tens of millions of users. Watches are restricted to a set of 500,000 popular videos, and all users have at least 50 watches in their sequence. The sequence is given by a list of watched videos and the timestamp of each watch.

Method	Precision@1	MAP@20
RNN with Δt Latent Cross	**0.1621**	**0.0828**
RRN (Concatenated Δt)	0.1465	0.0753
RNN (Plain, no time)	0.1345	0.0724
Bag Of Words	0.1250	0.0707
Bag of Words with time	0.1550	0.0794
Paragraph Vectors	0.1123	0.0642
Cowatch	0.1204	0.0621

Table 4: Results for Comparative Study: RNN with a latent cross performs the best.

The task is to predict the last 5 watches in the user's sequence. To measure this, we use Mean-Average-Precision-at-k (MAP@k) for $k = 1$ and $k = 20$ on the test set.

Model. For this set of experiments we use an RNN with an LSTM recurrent unit. We have no ReLU cells before or after the recurrent unit, and use a pre-determined hierarchical softmax (HSM) to predict the videos. Here, we use all but the first watch in the sequence as supervision during training. The model is trained using back-propagation with ADAM [26].

Since time is the only contextual feature in this data set, we use the video embedding v_j as the input and perform the latent cross with the timedelta value $w_{\Delta t}$ such that the LSTM is given $v_j * w_{\Delta t}$. This is an example of a pre-fusion cross. We will call this RNNLatentCross.

Baselines. We compare the RNNLatentCross model described above to highly tuned models of alternative forms:

- RRN: Use $[v_j; w_{\Delta t}]$ as the input to the RNN; similar to [43] and [41].
- RNN: RNN directly over v_j (without time); similar to [21].
- BOW: Bag of words model over the set of videos in the user's history and user demographics.
- BOW+Time: 3-layer feed-forward model taking as input a concatenation of a bag-of-watches, each of the last three videos watched, Δt, and the time of the week of the request. The model is trained with a softmax over the 50 videos most co-watched with the last watch.
- Paragraph Vector (PV): Use [13] to learn unsupervised embeddings of each user (based on user demographics and previous watches). Use the learned embeddings as well as an embedding of the last watch as input to a 1-layer feed forward classifier trained with sampled softmax.
- Cowatch: Predict the most common co-watched videos based on the last watch in the sequence.

Unless otherwise specified, all models have a hierarchical softmax. All models and their hyperparameters are tuned over the course of a modeling competition. Note, only BagOfWords and Paragraph Vector make use of user demographic data.

7.1.2 Results. We report our results for this experiment in Table 4. As can be seen there, our model, using the RNN with Δt having a latent cross with the watch, gives the best result for both Precision@1 and MAP@20. Possibly even more interestingly, is the relative performance of the models. We observe in both the bag of words models and the RNN models the critical importance of modeling time. Further, observe that the improvement from performing a latent cross instead of just concatenating Δt is *greater* than the improvement from including Δt as an input feature at all.

7.2 YouTube's Model

Second, we study multiple variants of our production model against a larger, more unrestricted dataset.

7.2.1 Setup. Here, we use a production dataset of user watches, which is less restrictive than the above setting. Our sequences are composed of the video that was watched and who created the video (uploader). We use a larger vocabulary on the order of millions of recently popular uploaded videos and uploaders.

We split the dataset into a training and test set based jointly on users and time. First, we split users into two sets: 90% of our users are in our training set and 10% in our test set. Second, to split by time, we select a time cut-off t_0 and during training only consider watches from before t_0. During testing, we consider watches from after t_0 + 4 hours. Similarly, the vocabulary of videos is based on data from before t_0.

Our model consists of embedding and concatenating all of the features defined above as inputs, followed by a 256-dimensional ReLU layer, a 256-dimensional GRU cell, and then another 256-dimensional ReLU layer, before being fed into the softmax layer. As described previously, we use the 100 most recent watches during the period $(t_0 - 7 \text{ days}, t_0]$ as supervision. Here, we train using the Adagrad optimizer [16] over many workers and parameter servers.

To test our model, we again measure the mean-average-precision-at-k. For watches that are not in our vocabulary, we always mark the prediction as incorrect. The evaluation MAP@k scores reported here are measured using approximately 45,000 watches.

7.2.2 The Value of Page as Context. We begin analyzing the accuracy improvements by incorporating Page in different ways. In particular, we compare not using Page, using Page as an input concatenated with the other inputs, and performing a post-fusion latent cross with Page. (Note, when we include page as a concatenated feature, it is concatenated during both pre-fusion and post-fusion.)

As can be seen in Figure 3, using Page with a latent cross offers the best accuracy. Additionally, we see that using both the latent cross and the concatenated input offers no additional improvement in accuracy, suggesting that the latent cross is sufficient to capture the relevant information that would be obtained through using the feature as a direct input.

7.2.3 Total Improvement. Last, we test how adding latent crosses on top of the full production model effects the accuracy. In this case, with each watch the model knows the page, the device type, the time, how long the video was watched for (watch time), how old the watch is (watch age), and the uploader. In particular, our baseline YouTube model uses the page, device, watch time, and timedelta values as pre-fusion concatenated features, and also uses the page, device, and watch age as post-fusion concatenated features.

We test including timedelta and page as pre-fusion latent crosses, as well as device type and page as post-fusion latent crosses. As can be seen in Figure 4, although all of these features were already included through concatenation, including them as latent crosses provides an improvement in accuracy over the baseline model. This also demonstrates the ability for pre-fusion and post-fusion with multiple features to work together and provide a strong accuracy improvement.

8 DISCUSSION

We explore below a number of questions raised by this work and implications for future work.

8.1 Discrete Relations in DNNs

While much of this paper has focused on enabling multiplicative interactions between features, we found that neural networks can also approximate discrete interactions, an area where factorization models have more difficulty. As an example, in [46] the authors find that modeling when user i performs action a on item j, $\langle u_{(i,a)}, v_j \rangle$ has better accuracy than $\langle u_i, v_j, w_a \rangle$. However, discovering that indexing users and actions together performs better is difficult, requiring data insights.

Similar to the experiments in Section 4, we generate synthetic data following the pattern $X_{i,j,a} = \langle u_{(i,a)}, v_j \rangle$ and test how well different network architectures predict $X_{i,j,a}$ given i, j and a are only concatenated as independent inputs. We initialize $u \in \mathbb{R}^{10000}$ and $v \in \mathbb{R}^{100}$ as vectors, such that X is a rank-1 matrix. We follow the same general experimental procedure as in Section 4, measuring the Pearson correlation (R) for networks with varying number of hidden layers and varying width to those hidden layers. (We train these networks with a learning rate of 0.01, ten-times smaller than the learning rate used above.) As a baseline, we also measure the Pearson correlation for tensor factorization ($\langle u_i, v_j, w_a \rangle$) for different ranks.

As can be seen in Figure 5, deep models, in some cases, attain a reasonably high Pearson correlation, suggesting that they are in fact able to approximate discrete crosses. Also interestingly, learning these crosses requires deep networks with wide hidden layers, particularly large for the size of the data. Additionally, we find that these networks are difficult to train.

These numbers are interesting relative to the baseline tensor factorization performance. We observe that the factorization models can approximate the data reasonably well, but requires relatively high rank. (Note, even if the underlying tensor is full rank, a factorization of rank 100 would suffice to describe it.) However, even at this high rank, the tensor factorization models require fewer parameters than the DNNs and are easier to train. Therefore, as with our results in Section 5, DNNs can approximate these patterns, but doing so can be difficult, and including low-rank interactions can help in providing easy-to-train approximations.

8.2 Second-Order DNNs

A natural question to ask when reading this paper is why not try much wider layers, make the model deeper, or more second-order units, like GRUs and LSTMs? All of these are reasonable modeling decisions, but in our experience make training of the model significantly more difficult. One of the strengths of this approach is that it is easy to implement and train, while still offering clear performance improvements, even when applied in conjunction with other second-order units like LSTMs and GRUs.

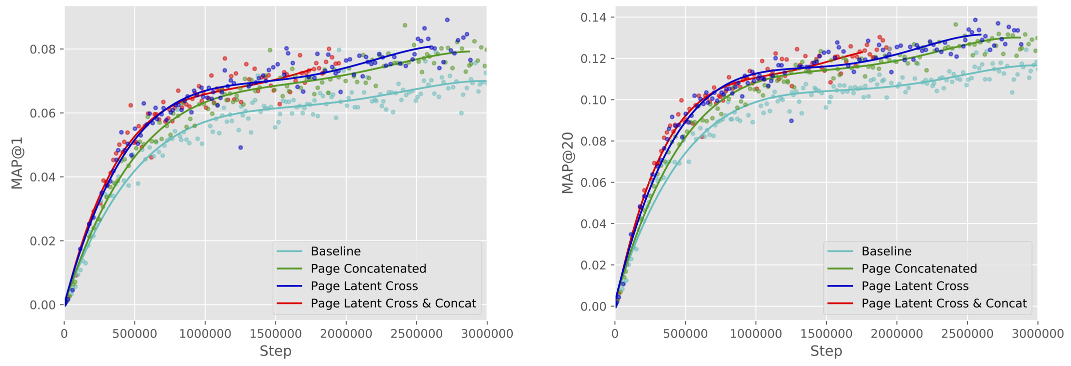

Figure 3: Accuracy from using page as either input or a latent cross feature (reported over training steps).

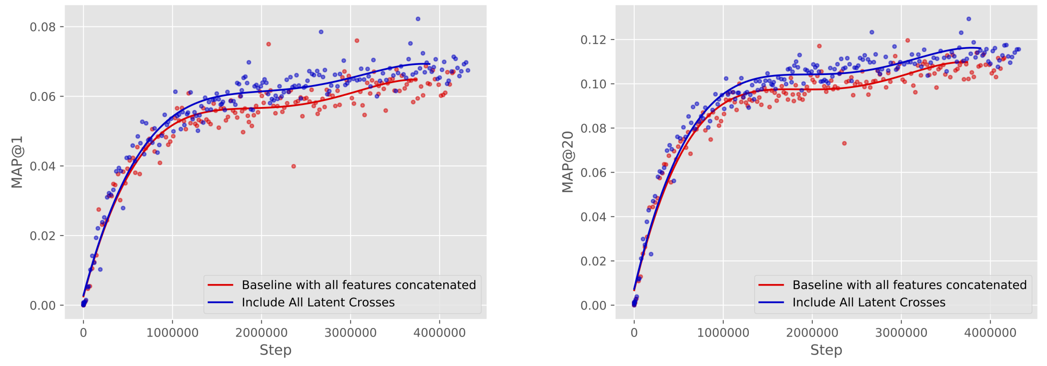

Figure 4: Accuracy from using all latent cross features against a baseline of using all possible concatenated input features.

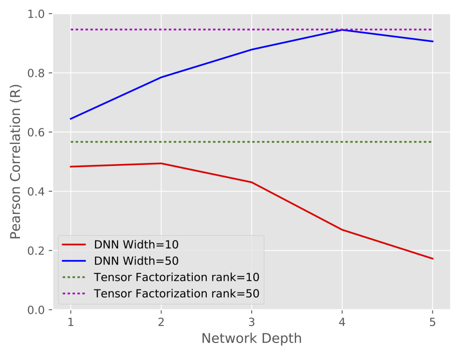

Figure 5: Sufficiently large DNNs can learn to approximate discrete interactions.

The growing trend throughout deep learning appears to be using more second-order interactions. For example, this is common in attention models and memory networks, as list above. While these are even more difficult to train, we believe this work shows the promise in that direction for neural recommender systems.

9 CONCLUSION

In this paper we have explored how to incorporate contextual data in a production recurrent recommender system at YouTube. In particular, this paper makes following contributions:

- **Challenges of First-Order DNNs:** We found feed-forward neural networks to be inefficient in modeling multiplicative relations (crosses) between features.
- **Production Model:** We offer a detailed description of our RNN-based recommender system used at YouTube.
- **Latent Cross:** We offer a simple technique for learning multiplicative relations in DNNs, including RNNs.
- **Empirical Results:** We demonstrate in multiple settings and with different context features that latent crosses improve recommendation accuracy, even on top of complex, state-of-the-art RNN recommenders.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, and others. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*.

[2] Amr Ahmed, Liangjie Hong, and Alexander J Smola. 2013. Hierarchical geographical modeling of user locations from social media posts. In *Proceedings of the 22nd international conference on World Wide Web (WWW)*. ACM, 25–36.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[4] James Bennett, Stan Lanning, and others. 2007. The netflix prize. In *Proceedings of KDD cup and workshop*, Vol. 2007. New York, NY, USA, 35.

[5] Alex Beutel, Ed H Chi, Zhiyuan Cheng, Hubert Pham, and John Anderson. 2017. Beyond Globally Optimal: Focused Learning for Improved Recommendations. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*. ACM.

[6] Pedro G Campos, Fernando Díez, and Iván Cantador. 2014. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction* 24, 1-2 (2014), 67–119.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, and others. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[8] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. 2010. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 759–768.

[9] Evangelia Christakopoulou and George Karypis. 2016. Local Item-Item Models For Top-N Recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*. ACM, 67–74.

[10] Junyoung Chung, Caglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2015. Gated Feedback Recurrent Neural Networks.. In *ICML*. 2067–2075.

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*. ACM, 191–198.

[12] Bin Cui, Anthony KH Tung, Ce Zhang, and Zhe Zhao. 2010. Multiple feature fusion for social media applications. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 435–446.

[13] Andrew M Dai, Christopher Olah, and Quoc V Le. 2015. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998* (2015).

[14] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083* (2016).

[15] Nan Du, Yichen Wang, Niao He, Jimeng Sun, and Le Song. 2015. Time-sensitive recommendation from recurrent user activities. In *Advances in Neural Information Processing Systems*. 3492–3500.

[16] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[17] Hancheng Ge, James Caverlee, and Haokai Lu. 2016. TAPER: A contextual tensor-based approach for personalized expert recommendation. (2016).

[18] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[19] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. ACM, New York, NY, USA, 355–364. DOI: https://doi.org/10.1145/3077136.3080777

[20] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[21] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[22] Balázs Hidasi and Domonkos Tikk. 2016. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery* 30, 2 (2016), 342–371.

[23] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[24] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *ICDM*.

[25] How Jing and Alexander J. Smola. 2017. Neural Survival Recommender. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM)*. 515–524.

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

[27] Ryan Kiros, Richard Zemel, and Ruslan R Salakhutdinov. 2014. A multiplicative model for learning distributed text-based attribute representations. In *Advances in neural information processing systems*. 2348–2356.

[28] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*. ACM, 426–434.

[29] Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. *Commun. ACM* 53, 4 (2010), 89–97.

[30] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37. DOI: https://doi.org/10.1109/MC.2009.263

[31] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2013. Local Low-Rank Matrix Approximation. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*. 82–90. http://jmlr.org/proceedings/papers/v28/lee13.html

[32] Haokai Lu and James Caverlee. 2015. Exploiting geo-spatial preference for personalized expert recommendation. In *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys)*. ACM, 67–74.

[33] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 1149–1154.

[34] Steffen Rendle. 2012. Factorization Machines with libFM. *ACM TIST* 3, 3, Article 57 (May 2012), 22 pages.

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

[36] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*. ACM, 111–112.

[37] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. 2012. TFMAP: optimizing MAP for top-n context-aware recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 155–164.

[38] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, and others. 2015. End-to-end memory networks. In *Advances in neural information processing systems*. 2440–2448.

[39] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 17–22.

[40] Duyu Tang, Bing Qin, Ting Liu, and Yuekui Yang. 2015. User Modeling with Neural Network for Review Rating Prediction.. In *IJCAI*. 1340–1346.

[41] Bartlomiej Twardowski. 2016. Modelling Contextual Information in Session-Aware Recommender Systems with Neural Networks.. In *RecSys*. 273–276.

[42] Manasi Vartak, Hugo Larochelle, and Arvind Thiagarajan. 2017. A Meta-Learning Perspective on Cold-Start Recommendations for Items. In *Advances in Neural Information Processing Systems*. 6888–6898.

[43] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM)*. 495–503.

[44] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan R Salakhutdinov. 2016. On multiplicative integration with recurrent neural networks. In *Advances in Neural Information Processing Systems*. 2856–2864.

[45] Chunfeng Yang, Huan Yan, Donghan Yu, Yong Li, and Dah Ming Chiu. 2017. Multi-site User Behavior Modeling and Its Application in Video Recommendation. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 175–184.

[46] Zhe Zhao, Zhiyuan Cheng, Lichan Hong, and Ed H Chi. 2015. Improving User Topic Interest Profiles by Behavior Factorization. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*. 1406–1416.

[47] Lei Zheng, Vahid Noroozi, and Philip S Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM)*. ACM, 425–434.

[48] Yong Zheng, Bamshad Mobasher, and Robin Burke. 2014. CSLIM: Contextual SLIM recommendation algorithms. In *Proceedings of the 8th ACM Conference on Recommender Systems*. ACM, 301–304.

[49] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to Do Next: Modeling User Behaviors by Time-LSTM. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 3602–3608. DOI: https://doi.org/10.24963/ijcai.2017/504