ATRank: An Attention-Based User Behavior Modeling Framework for Recommendation

Abstract

A user can be represented as what he/she does along the history. A common way to deal with the user modeling problem is to manually extract all kinds of aggregated features over the heterogeneous behaviors, which may fail to capture the inner-relation of the data that goes beyond human instincts. Recent works usually use RNN-based methods to give an overall embedding of a behavior sequence, which then could be exploited by the downstream applications. However, this can only preserve very limited information, or aggregated memories of a person. When a downstream application requires to facilitate the modeled user features, it may lose the integrity of the specific highly correlated behavior of the user, and introduce noises derived from unrelated behaviors. This paper proposes an attention based user behavior modeling framework called ATRank, which we mainly use for recommendation tasks. Heterogeneous user behaviors are considered in our model that we project all types of behaviors into multiple latent semantic spaces, where influence can be made among the behaviors via self-attention. Downstream applications then can use the user behavior vectors via vanilla attention. Experiments show that ATRank can achieve better performance and faster training process. We further explore ATRank to use one unified model to predict different types of user behaviors at the same time, showing a comparable performance with the highly optimized individual models.

Introduction

As a word can be represented by the surrounding context (Mikolov et al. 2013), a user can be represented by his/her behaviors along the history. For downstream tasks like ranking in recommendation system, traditional ways to represent a user is to extract all kinds of hand-crafted features aggregated over different types of user behaviors. This feature engineering procedure may fail to capture the inner-relation of the data that goes beyond human instincts and it requires too much laboring work. Besides, the aggregated features also lose information of any individual behavior that could be precisely related with the object that needs to be predicted in the downstream application.

As the user behaviors naturally form a sequence over the timeline, RNN/CNN structures are usually exploited to encode the behavior sequence for downstream applications as in the encoder-decoder framework. However, for RNN based methods, long-term dependancies is still very hard to preserve even using the advanced memory cell structures like LSTM and GRU (Hochreiter and Schmidhuber 1997; Chung et al. 2014). RNN also has a disadvantage that both the offline training and the online prediction process are time-consuming, due to its recursive natural which is hard to parallelize. Recently, it's shown that CNN-based encoding methods can also achieve comparable performance with RNN in many sequence prediction tasks. Though it's highly parallelizable, the longest length of the interaction paths between any two positions in the CNN network is $log_k(n)$, where n is the number of user behaviors and k is the kernel width.

Both the basic RNN and CNN encoders suffer from the problem that the fixed-size encoding vector may not support both short and long sequences well. The attention mechanism is then introduced to provide the ability to reference specific records dynamically in the decoder, which has already achieved great successes in fields like Machine Translation, Image Caption in recent years. Downstream applications of behavior modeling like recommendation can also utilize the attention mechanism, since the item to be ranked may only be related to very small parts of the user behaviors.

However, we show that the one-dimensional attention score between any two vectors may neutralize their relationships in different semantic spaces. The attention-based pooling can be formularized as $C = \sum_{i=1}^{n} a_i \vec{v_i}$, where a_i is a scalar. We can see that, each element in $\vec{v_i}$ will multiply by the same a_i, such that different semantics of $\vec{v_i}$ can only compromise to use this unique multiplier, which makes it hard to preserve only the highly related semantics while discard those unrelated parts in the weighted average vector.

Another important issue is that the user behaviors are naturally heterogeneous, highly flexible, and thus hard to model. For large companies today, various kinds of user behaviors could be collected, which makes it increasingly important to utilize those user behaviors to provide better personalized services. Take the e-business recommendation service as an example, a user may browse/buy/mark items, receive/use coupons, click ads, search keywords, write down reviews, or even watch videos and live shows offered by a shop, each of which reveals some aspects of the user that would be helpful to build more comprehensive user models, providing a better understanding of the user intents. Though heterogeneous data representation can be learnt by minimizing the distances in the projected latent spaces (Bordes et al. 2013; Lin et al. 2015; Chang et al. 2015), there are no explicit supervisions like mapping or inferencing between any pair of user behaviors that could help build the individual behavior representations.

In this paper, we propose an attention-based user behavior modeling framework called ATRank, which we currently use for recommendation tasks. We first project variable-length heterogenous behavior representations into multiple latent spaces, where behavior interactions can be made under common semantics. Then we exploit the power of self-attention mechanism to model each user behavior after considering the influences brought by other behaviors. We perform vanilla attention between these attention vectors and the ranking item vector, whose outputs are fed into a ranking neural network. It's observed that self-attention with time encoding can be a replacement for complex RNN and CNN structures in the sequential behavior encoding, which is fast in both training and prediction phase. Experiments also show that ATRank achieves better performance while is faster in training. We further explore the model to predict multiple types of user behaviors at the same time using only one unified model, showing a comparable performance with the highly optimized individual models.

Related Works

Context Aware Recommendation. An industrial recommendation system usually has varieties of models to extract different aspects of the user features, e.g., user gender, income, affordance, categorial preference, etc, all from the user behaviors, trying to capture the context information of the user intents. Then it builds different models for each recommendation scenario using those extracted features, either continuous or categorial (Covington, Adams, and Sargin 2016; Cheng et al. 2016). These are multi-step jobs and it's hard to optimize jointly.

RNN based methods are studied for recommendation in academic fields in recent years, which builds the recommendation context directly from the behaviors for each user (Rendle et al. 2011; Hariri, Mobasher, and Burke 2012; Wu et al. 2016). Though it's an more elegant way to encode user context, it still suffers from several difficulties. First, RNN is hard to parallelize in prediction phase, which makes it not easy to ensure the response time to be low enough for a commercial recommendation system. Second, the RNN embedding of the user behaviors is fix-sized, which is not well suited for modeling both long and short behavior lists. It also plays a role of an aggregated status for a user history, which may not preserve specific behavior information when enrolled in the downstream applications.

Attention and Self-Attention. Attention is introduced by (Bahdanau, Cho, and Bengio 2014) firstly in the encoder-decoder framework, to provide more accurate alignment for each position in the machine translation task. Instead of preserving only one single vector representation for the whole object in the encoder, it keeps the vectors for each element as well, so that the decoder can reference these vectors at any decoding step. Recently, attention based methods are widely applied in many other tasks like reading comprehension (Cui et al. 2016; Cheng, Dong, and Lapata 2016; Lin et al. 2017), ads recommendation (Zhai et al. 2016), computer vision (Xu et al. 2015), etc.

Self-attentions are also studied in different mechanisms (Vaswani et al. 2017; Lin et al. 2017; Cui et al. 2016), in which inner-relations of the data at the encoder side are considered. Note that, both work of (Vaswani et al. 2017; Lin et al. 2017) show that project each word onto multiple spaces could improve the performance of their own tasks.

Heterogeneous Behavior Modeling. Heterogeneous data representation is heavily studied in domains like knowledge graph completion and multimodal learning. In the domain of knowledge graph completion, lots of works have been proposed to learn heterogeneous entity and relation representations by minimizing the distances of the linear projected entities in the relation-type semantic subspace (Bordes et al. 2013; Lin et al. 2015; Wang et al. 2014). Similar idea is adopted in multimodal learning tasks like image caption (Vinyals et al. 2015) and audio-visual speech classification (Ngiam et al. 2011).

Self-Attention Based Behavior Modeling Framework

This paper focuses on general user behaviors that can be interpreted using the binary relation between a user and an entity object. We formulate a user behavior as a tuple $\{a, o, t\}$, where a stands for the behavior type describing the action a user takes, o is the object that the behavior acts on, and t is the timestamp when the behavior happens. Note that, while a and t are the atomic features representing the behavior type and action time, o is represented as all its belonging features. So that, a user can be represented as all his/her behaviors $U = \{(a_j, o_j, t_j)|j = 1, 2, ..., m\}$.

We illustrate the overall attention-based heterogeneous user behavior modeling framework in Figure 1. We divide the framework into several blocks, namely raw feature spaces, behavior embedding spaces, latent semantic spaces, behavior interaction layers and downstream application network. Next, we discuss all these blocks in detail.

Raw Feature Spaces

We first partition the user behavior tuples $U = \{(a_j, o_j, t_j)|j = 1, 2, ..., m\}$ into different behavior groups $G = \{bg_1, bg_2, ..., bg_n\}$ according to the target object types, where $bg_i \bigcap bg_j = \emptyset$ and $U = \bigcup_{i=1}^{n} bg_i$. Within each bg_i, the raw feature spaces of the objects are the same, which may contain both the continuous features and the categorial features depending on how we model the target object. Then we can use group-specific neural nets to build up the behavior embeddings.

Behavior Embedding Spaces

In each behavior group bg_i, we embed the raw features of any behavior tuple (a_j, o_j, t_j) using the same embedding

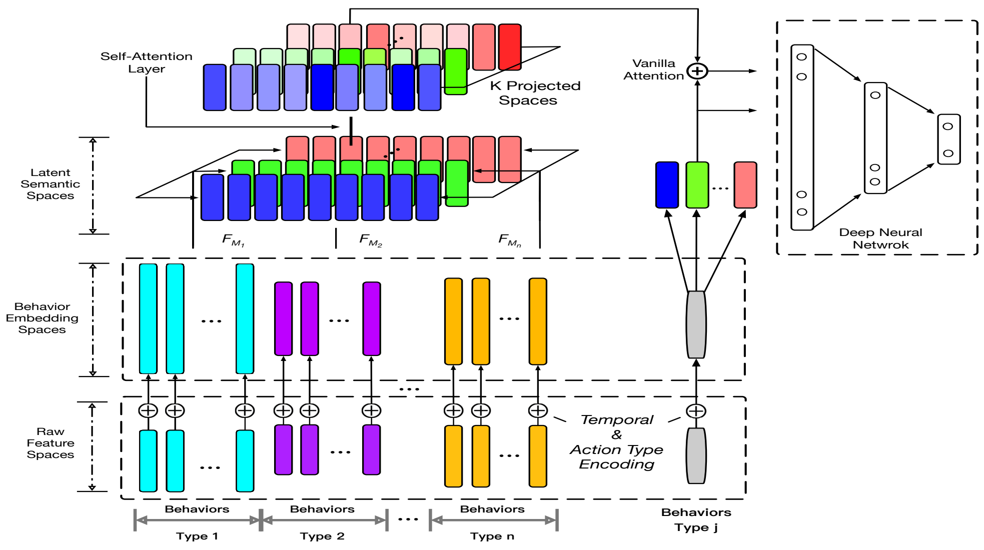


Figure 1: Attention-Based Heterogeneous Behaviors Modeling Framework

building block, $u_{ij} = f_i(a_j, o_j, t_j)$. The raw features of the target object o_j are concatenated and fed into the feedforward neural network, while a_j and t_j are encoded separately that will be discussed below.

Temporal & Behavior Type Encoding. Each behavior may have different action time, which is a very important continuous feature, providing a good opportunity to cope with sequence data without the enrollment of any CNN or RNN-structures. In order to preserve the temporal information, we use a temporal encoding methods similar as the positional embedding mentioned in (Gehring et al. 2017), with the embedding content to be the concatenation of both time and action type. However, we found it very difficult in learning a good embedding directly on this continuous time feature using embedding concatenation or addition. The better way we found is to bucketize the time feature into multiple granularities and perform categorial feature lookups. For example, in the recommend task over the amazon dataset, we slice the elasped time w.r.t the ranking time into intervals whose gap length grow exponentially, e.g., we map the time in range $[0, 1), [1, 2), [2, 4), ..., [2^k, 2^{k+1}), ...$ to categorial feature of 0, 1, 2, ..., k+1, ... Different behavior groups may have different granularities of time slicing. Similar as the temporal encoding, we perform a direct lookup on the behavior type, trying to capture the impacts of different behavior types.

Then we can encode any user behavior $u_{ij} \in bg_i$ as

$$\boxed{u_{ij} = emb_i(o_j) + lookup_i^t(bucketize(t_j)) + lookup_i^a(a_j)}$$

where emb_i is a the embedding concatenation for any object $o \in bg_i$. The outputs of the behavior embedding space are the list of vectors in all behavior groups,

$$B = \{u_{bg_1}, u_{bg_2}, ..., u_{bg_n}\}$$

where u_{bg_i} is the set of all behavior embeddings belonging to behavior group i, $u_{bg_i} = concat_j(u_{ij})$. Note that, the shape of the embeddings for each group may be different, since 1) the numbers of behaviors in each group vary from user to user, 2) the information carried for each type of behavior could be in different volume. E.g., an item behavior may contain much more information than a keyword search behavior, since the item may reveal user preference towards brand, price, style, tastes, etc, while one short query keyword can only show limited intents of a user.

Embedding Sharing. We enable the embedding sharing when multiple groups of objects have features in common, e.g, shop id, category id that can be both shared by objects like items and coupons in an e-business recommendation system, referring to the same entities. Note that we don't share the temporal encoding lookups across behavior groups, since the time impact for each type of behavior may vary a lot. For example, in the coupon receiving scene, the time impact can dominate the behavior showing that this type of behavior is quite of short-term, while buying shoes of a certain brand may be of long-term interest, where the temporal information could be less important.

Latent Semantic Spaces

Lots of works have shown that linear projections over multi-subspaces could improve the performance in various tasks. We argue that heterogeneous behaviors could have very different expressive power, thus their embedding space could be in both different sizes and meanings. Linear projection here plays a role to put them into the same semantic space, where connections or comparisons can be made. So we map each behavior into K latent semantic spaces. We explain this idea in Figure 1 that, each behavior group is represented as a composite color bar, which is then factorized into RGB colors where further operations can be taken within the comparable atomic color spaces.

Given the user behaviors' representation B, we first project the variable-length behaviors in different groups into fix-length encoding vectors

$$S = concat^{(0)}(\mathcal{F}_{M_1}(u_{bg_1}), \mathcal{F}_{M_2}(u_{bg_2}), ..., \mathcal{F}_{M_n}(u_{bg_n})) \tag{1}$$

where the symbol $\mathcal{F}_{M_i}$ represents the projection function which is parameterized by M_i. $\mathcal{F}_{M_i}$ maps a behavior in group i onto the overall space of dimension size s_{all}. Then the projected behavior embeddings in each spaces are

$$S_k = \mathcal{F}_{P_k}(S) \tag{2}$$

where $\mathcal{F}_{P_k}$ is a projection function for the k-th semantic space. In this paper, all the projection function $\mathcal{F}_\theta$ is set to be a single layer perceptron parametrized by θ with $relu$ as the activation function.

Let the number of all behaviors of a user be n_{all}, the number of behaviors in group i be n_i, the dimension of the feature embedding space for that group be b_i, the dimension of the k-th semantic space be s_k. Then u_{bg_i} is in shape of n_i-by-b_i, S_k is in shape of n_{all}-by-s_k. The superscript on $concat$ function indicates which dimension the concatenation is performed on.

Self-Attention Layer

This layer tries to capture the inner-relationships among each semantic space. The intense of each behavior could be affected by others, such that attentions for some behaviors could be distracted and the others could be strengthened. This is done by the self-attention mechanism. The outputs of this layer could be regarded as the behavior representation sequence taking considerations of the impacts of the others in each latent space.

We exploit similar self-attention structure mentioned in (Vaswani et al. 2017) for Machine Translation task, with some customized settings. We calculate each attention score matrix A_k in the k-th semantic space as

$$A_k = softmax(a(S_k, S; \theta_k)) \tag{3}$$

where each row of A_k is the score vector w.r.t all the attention vectors in that subspace, the softmax function guarantees that all scores for a behavior sums to one. The score function $a(S_k, S; \theta_k)$ measures the impacts among all behaviors in the k-th semantic space, and in this paper we choose the bilinear scoring function as in (Luong, Pham, and Manning 2015)

$$a(S_k, S; \theta_k) = S_k W_k S^T \tag{4}$$

Then the attention vectors of space k are

$$C_k = A_k \mathcal{F}_{Q_k}(S) \tag{5}$$

where $\mathcal{F}_{Q_k}$ is another projection function that maps S onto the k-th semantic space, which in this paper is a single layer perceptron with $relu$ activation fucntion. Then C_k is in shape of n_{all}-by-s_k.

These vectors from different subspaces are concatenated together and then reorganize by

$$C = \mathscr{F}_{self}(concat^{(1)}(C_1, C_2, ..., C_K)) \tag{6}$$

where $\mathscr{F}$ is a feedforward network with one hidden layer, aiming to provide non-linear transformation over each behavior vector after attention-based pooling. The output of $\mathscr{F}$ keeps the same shape as the input, and we successively perform drop out, residual connections and layer normalization on the outputs.

Downstream Application Network

With the generated user behavior models, we can ensemble various kinds of neural networks according to the downstream task requirement. In this paper, we focus on evaluating the recommendation tasks, and we set the downstream application network to be a point-wise or a pair-wise fully connected neural network.

Vanilla Attention. For both the point-wise and the pair-wise model, a vanilla attention is performed to produce the final context vector e_u^t for user u w.r.t. the embedding vector q_t to be predicted. This follows the similar procedure in the self-attention phase,

$$\begin{aligned} \vec{h_t} &= \mathcal{F}_{M_{g(t)}}(\vec{q_t}), \quad \vec{s_k} = \mathcal{F}_{P_k}(\vec{h_t}) \\ \vec{c_k} &= softmax(a(\vec{s_k}, C; \theta_k))\mathcal{F}_{Q_k}(C) \\ \vec{e_u^t} &= \mathscr{F}_{vanilla}(concat^{(1)}((\vec{c_1}, \vec{c_2}, ..., \vec{c_K}))) \end{aligned} \tag{7}$$

where $g(t)$ maps behavior t to its behavior group number, and $\mathscr{F}_{vanilla}$ has different parameters with $\mathscr{F}_{self}$.

For a point-wise prediction model, the probability of each behavior t that will be taken is predicted. We extract the raw feature of t, map these features into the behavior embedding space according to the building block of its behavior group q_t. Then we perform a vanilla attention as shown above. The final loss function is the sigmoid cross entropy loss

$$-\sum_{t,u} y_t \log \sigma(f(h_t, e_u^t)) + (1 - y_t) \log (1 - \sigma(f(h_t, e_u^t))) \tag{8}$$

where f is a ranking function whose input is the ranking behavior embedding q_t and the user encoding e_u^t w.r.t behavior t computed by vanilla-attention. f can be either a dot-product function or a more complexed deep neural network.

For a pair-wise prediction model, a set of partial order is defined as

$$D_S = \{(u, i, j) | i \in I_u^+ \wedge j \in I \backslash I_u^+\} \tag{9}$$

where I is the set of all behaviors and I_u^+ is the behavior subset that user u has performed. We compute the embedding difference of behavior i and j in each subspace as $\Delta = \{h_i - h_j\}$, and then replace the term h_t with Δ in equation 7 to calculate $e_u^{i>j}$. The loss function in pair-wise prediction model is defined as

$$L = -\sum_{(u,i,j) \in D_S} \log \sigma(f(\Delta, e_u^{i>j})) \tag{10}$$

Note that, the user behaviors above need not to be restricted to single type or even group, meaning that we can build a unified model to perform multi-task that predicts different types of behaviors at the same time.

Experiment

In this section, we first evaluate our method in context-aware recommendation tasks with only one type of behavior, trying to show the benefits of using self-attention to encode user behaviors. Then we collect a multi-behavior dataset from an online recommendation service, and evaluate how the heterogeneous behavior model performs in recommendation tasks. We further explore the model to support multi-task using one unified model, which can predict all kinds of behaviors at the same time.

Dataset

Amazon Dataset. We collect several subsets of amazon product data as in (McAuley et al. 2015), which have already been reduced to satisfy the 5-core property, such that each of the remaining users and items have 5 reviews each [1]. The feature we use contains user id, item id, cate id and timestamps. The statistics of the dataset is show in table 1. Let all the user behaviors for u be $(b_1, b_2, ..., b_k, ..., b_n)$, we make the first k behaviors of u to predict the k+1-th behavior in the train set, where $k = 1, 2, ..., n$-2, and we use the first n-1 behaviors to predict the last one in the test set.

Dataset	# Users	# Items	# Cates	# Samples
Electro.	192,403	63,001	801	1,689,188
Clothing.	39,387	23,033	484	278,677

Table 1: Statistics of Amazon DataSets

Taobao Dataset. We collect various kinds of user behaviors in a commercial e-business website [2]. We extract three groups of behaviors, namely item behavior group, search behavior group and coupon receiving behavior group. Item behaviors list all the user action types on items, such as browse, mark, buy, etc. The feature set of an item contains item id, shop id, brand id, category id. Search behavior is expressed as the search tokens, and has only one type of action. In coupon receiving behavior group, coupon features are consist of coupon id, shop id and coupon type. This gourp also has only one action type. All above behaviors have a time feature that represent the timestamp when that behavior happens. The statics of each categorial is shown in table 2. This dataset is guaranteed that each user has at least 3 behaviors in each behavior group.

Competitors

- BPR-MF: Bayesian Personalized Ranking (Rendle et al. 2009) is a pairwise ranking framework. It trains on pairs of a user's positive and negative examples, maximizing the posterior probability of the difference given the same user. The item vector is a concatenation of item embedding and category embedding, each of which has a dimension of 64, the user vector has a dimension of 128.
- Bi-LSTM: We implement a Bi-LSTM method to encode the user behavior history, whose difference with the work (Zhang et al. 2014) is that we use the bidirectional encoder for LSTM instead of unidirectional because of better performance in our experiments. Then we concatenate the two final hidden state to user embedding. The stacked LSTM depth is set to be 1.

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

[2]http://www.taobao.com

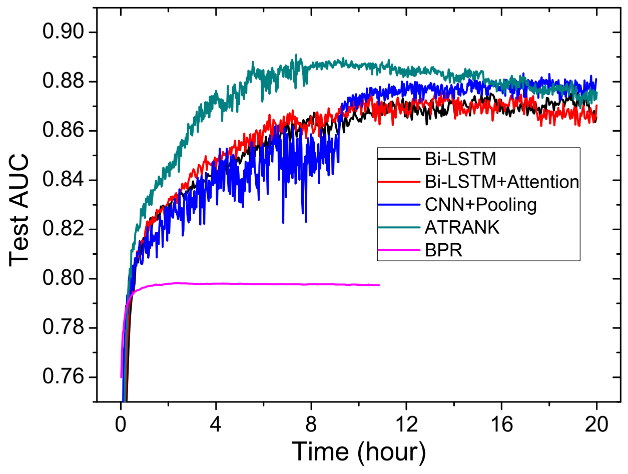


Figure 2: AUC Progress in Amazon Electro Test Set

- Bi-LSTM+Attention: We add a vanilla attention on top of the Bi-LSTM method mentioned above to see the difference.
- CNN+Pooling: We use a CNN structure with max pooling to encode the user history as in (Zheng, Noroozi, and Yu 2017; Kim 2014). Especially, we use ten kinds of window sizes from one to ten for extracting different features, and all of the feature map have the same kernel size with 32. Then we apply a max-over-time pooling operation over the feature map, and pass all pooled features from different maps to a fully connected layer to produce final user embedding.

Hyperparameters

Our method and all competitors use the common hyperparameters as follows.

- Network Shape. We set the dimension size of each categorial feature embedding to be 64, and we concat these embeddings as the behavior representation in the behavior embedding space. The hidden size of all layers is set to be 128. The ranking function f is simply set to be the dot product in these tasks. As we observe better performance with point-wise ranking model, we omit the results of pair-wise models here for simplicity. For ATRank, we set the number of the latent semantic spaces to be 8, whose dimension sizes sum to be the same as the size of the hidden layer.
- Batch Size. The batch size is set to be 32 for all methods.
- Regularization. The l2-loss weight is set to be 5e-5.
- Optimizer. We use SGD as the optimizer and apply exponential decay which learning rate starts at 1.0 and decay rate is set to 0.1.

Evaluation Metrics

We evaluate the average user AUC as following:

$$AUC = \frac{1}{|U^{Test}|} \sum_{u \in U^{Test}} \frac{1}{|I_u^+||I_u^-|} \sum_{i \in I_u^+} \sum_{j \in I_u^-} \delta(\hat{p_{u,i}} > \hat{p_{u,j}})$$

Dataset	#Users	#Items	#Cates	#Shops	#Brands	#Queries	#Coupons	#Records	Avg Length
Multi-Behavior.	30,358	447,878	4,704	109,665	49,859	111,646	64,388	247,313	19.8

Table 2: Statistics of Multi-Behavior DataSets

where $\hat{p_{u,i}}$ is the predicted probability that a user $u \in U^{Test}$ may act on i in the test set and $\delta(\cdot)$ is the indicator function.

Results on Single-type Behavior Dataset

We first illustrate the average user AUC of all the methods for the amazon dataset in table 3. We can see that ATRank performs better than the competitors especially when the user behavior becomes denser, which shows the superior of the self-attention based user behavior models.

Table 4 illustrates the average vanilla-attention score for different time buckets over the whole amaon dataset, which can be inferred that time encoding via self-attention can also incorporate the time information as a replacement of RNN/CNN.

Then we show how the average user AUC for the test set evolves along with the training procedure in Figure 2. We can see that ATRank converges much faster than RNN or CNN-based methods in training. It's because ATRank does not incur any less-parallelizable operations like RNN, and any behavior could be affected by the other in just one layer of self attention, which leads to less computational dependency and better performance. Though BPR is faster in training, it has a very pool performance.

Results on Multi-type Behavior Dataset

We train the general model to evaluate the probability of the next user action, including all types of behaviors. We evaluate 7 tasks regarding different parts of the dataset as the train set and the test set.

One2one model. First we train three models using only one type of the behaviors and predict on the same type of behavior, namely item2item, coupon2coupon, query2query as baseline, we refer these type of model as one2one model.

All2one model. Then we train another three models using all types of the behaviors as user history and predict each type of behavior separately, namely all2item, all2coupon, all2query, we refer these models as all2one model.

All2all model. Finally we train a unified model for multi-task evaluation, using all types of user behaviors to predict any type of behavior at the same time, and we refer this model as all2all model.

Sample generation. In the multi-type behavior dataset we build, the item samples are the click actions in one of the inner-shop recommendation scenarios, which are collected from the online click log of that scenario containing both positive and negative samples. The user behaviors, however, are not limited to that scenario. We build the coupon samples by using the behaviors before the coupon receiving action, and the negative sample is randomly sampled in the rest of the coupons. The query samples are made similar with the coupon samples.

Then we show the average user AUC of all three models in table 5 respecitvely. We can see that, all2one model utilizes all types of user behaviors so that it performs better than the one2one models which only use the same type of behavior. The all2all model actually performs three different tasks using only one model, which shows that it can benefit from these mix-typed training process and achieve comparable performance with the highly optimized individual models.

Dataset	Electro.	Clothe.
BPR	0.7982	0.7061
Bi-LSTM	0.8757	0.7869
Bi-LSTM + Attention	0.8769	0.7835
CNN + Max Pooling	0.8804	0.7786
ATRank	**0.8921**	**0.7905**

Table 3: AUC on Amazon Dataset

Case Study

We try to explain the effects of self-attention and vanilla-attention in our model separately by case study. We visualize a user buying sequence from amazon dataset in Figure 4(a), which are one bag for woman, one ring, one hat for men, five jewelry of all kinds, one women's legging and women's shows respectively, the ranking item is a women's hoodie.

We first illustrate how the self-attention scores look like among those behaviors in different semantic spaces in Figure 3. Here the items that a user has bought form a timeline placed repeatedly as coordinate, and the heat map matrix represent the self-attention scores between any two behaviors in several different semantic space. Note that, the score matrix is normalized through row-oriented softmax so that it is asymmetric.

Then we can observe several interesting phenomenons from the Figure 3. First, different semantic space may focus on different behaviors, since the heat distributions vary a lot in all the 8 latent spaces. Second, in some spaces, e.g., space I, II, III and VIII, the relative trends of the attention score vectors keep the same for each row's behavior, while the strength is varied. The higher scores tend to gather around some specific columns. These spaces may consider more about the overall importance of a behavior among them. Third, in some other spaces, the higher scores tend to form a dense square. Take the space VI as an example, behavior 2, 4, 5, 6, 7, 8 may form a group of jewelries, among which user behaviors have very high inner attention scores. Those spaces probably consider more about relationships of category similarities. It then can be concluded that self-attention models the impacts among the user behaviors in different semantic spaces.

Then we study the effects of vanilla-attention in ATRank in Figure 4. We can see that the attention score for each latent space also varies a lot in Figure 4(b). Some latent

Time Range(#Day)	[0, 2)	[2, 4)	[4, 8)	[8, 16)	[16, 32)	[32, 64)	[64, 128)
Avg Att-Score	0.2494	0.1655	0.1737	0.1770	0.1584	0.1259	0.1188

Table 4: Average Attention Score for Different Time Bucket over Amazon Dataset

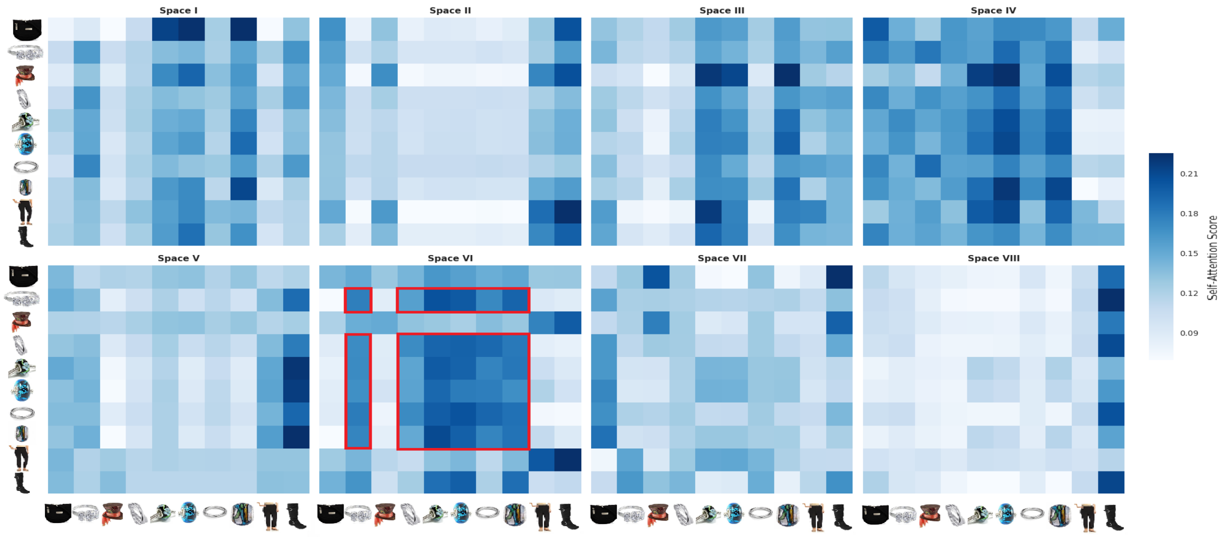
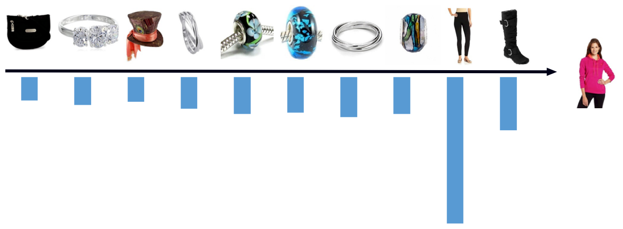
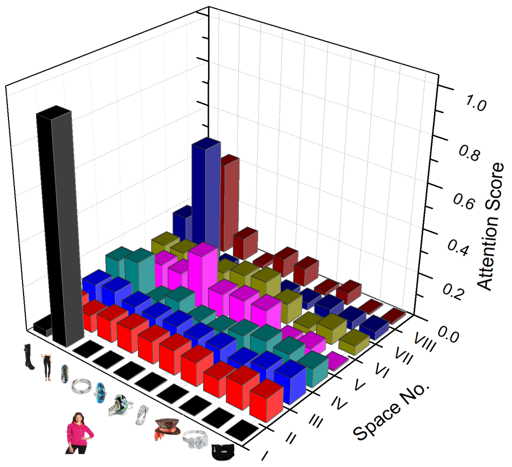


Figure 3: Case Study: Heatmap of Self-Attention in ATRank



(a) Average Vanilla-Attention Score over All Latent Space



(b) Vanilla-Attention Score in Different Latent Spaces

Figure 4: Case Study on Vanilla-Attention in ATRank

Predict Target	Item	Query	Coupon
Bi-LSTM	0.6779	0.6019	0.8500
Bi-LSTM + Attention	0.6754	0.5999	0.8413
CNN + Max Pooling	0.6762	0.6100	0.8611
ATRank-one2one	0.6785	0.6132	0.8601
ATRank-all2one	**0.6825**	**0.6297**	**0.8725**
ATRank-all2all	0.6759	0.6199	0.8587

Table 5: AUC on Ali Multi Behavior Dataset

spaces, e.g., space I, VII, VIII, highlight only the most correlated behaviors, while others like II, III show the average aggregation of all the behaviors. Note that, the space number of vanilla attention does not have a necessarily correspondence with that of self-attention, because there is a feedforward neural network that performs another non-linear projections on the concatenated self-attention vectors.

Conclusion

This paper proposes an attention-based behavior modeling framework called ATRank, which is evaluated on recommendation tasks. ATRank can model with heterogeneous user behaviors using only the attention model. Behaviors interactions are captured using self-attention in multiple semantic spaces. The model can also perform multi-task that predict all types of user actions using one unified model, which shows comparable performance with the highly optimized individual models. In the experiment, we show that our method achieves faster convergence while obtains better performance. We also give a case study that shows the insight of the how attention works in ATRank.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bordes, A.; Usunier, N.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *International Conference on Neural Information Processing Systems*, 2787–2795.

Chang, S.; Han, W.; Tang, J.; Qi, G.-J.; Aggarwal, C. C.; and Huang, T. S. 2015. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 119–128. ACM.

Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; Anil, R.; Haque, Z.; Hong, L.; Jain, V.; Liu, X.; and Shah, H. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, 7–10. New York, NY, USA: ACM.

Cheng, J.; Dong, L.; and Lapata, M. 2016. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Covington, P.; Adams, J.; and Sargin, E. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 191–198. ACM.

Cui, Y.; Chen, Z.; Wei, S.; Wang, S.; Liu, T.; and Hu, G. 2016. Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423*.

Gehring, J.; Auli, M.; Grangier, D.; Yarats, D.; and Dauphin, Y. N. 2017. Convolutional sequence to sequence learning. *ArXiv e-prints*.

Hariri, N.; Mobasher, B.; and Burke, R. 2012. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems*, 131–138. ACM.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kim, Y. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Lin, Y.; Liu, Z.; Zhu, X.; Zhu, X.; and Zhu, X. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2181–2187.

Lin, Z.; Feng, M.; Santos, C. N. d.; Yu, M.; Xiang, B.; Zhou, B.; and Bengio, Y. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

McAuley, J.; Targett, C.; Shi, Q.; and Van Den Hengel, A. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 43–52. ACM.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; and Ng, A. Y. 2011. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, 689–696.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, 452–461. AUAI Press.

Rendle, S.; Gantner, Z.; Freudenthaler, C.; and Schmidt-Thieme, L. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 635–644. ACM.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Vinyals, O.; Toshev, A.; Bengio, S.; and Erhan, D. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3156–3164.

Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge graph embedding by translating on hyperplanes. *AAAI - Association for the Advancement of Artificial Intelligence*.

Wu, S.; Ren, W.; Yu, C.; Chen, G.; Zhang, D.; and Zhu, J. 2016. Personal recommendation using deep recurrent neural networks in netease. In *IEEE International Conference on Data Engineering*, 1218–1229.

Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; and Bengio, Y. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, 2048–2057.

Zhai, S.; Chang, K.-h.; Zhang, R.; and Zhang, Z. M. 2016. Deepintent: Learning attentions for online advertising with recurrent neural networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1295–1304. ACM.

Zhang, Y.; Dai, H.; Xu, C.; Feng, J.; Wang, T.; Bian, J.; Wang, B.; and Liu, T.-Y. 2014. Sequential click prediction for sponsored search with recurrent neural networks. In *AAAI*, 1369–1375.

Zheng, L.; Noroozi, V.; and Yu, P. S. 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*, 425–434. ACM.