Attentive Sequential Models of Latent Intent for Next Item Recommendation

Md Mehrab Tanjim*
mtanjim@eng.ucsd.edu
UC San Diego

Congzhe Su, Ethan Benjamin, Diane Hu, Liangjie Hong
{csu,ebenjamin,dhu,lhong}@etsy.com
Etsy

Julian McAuley
jmcauley@eng.ucsd.edu
UC San Diego

ABSTRACT

Users exhibit different intents across e-commerce services (e.g. discovering items, purchasing gifts, etc.) which drives them to interact with a wide variety of items in multiple ways (e.g. click, add-to-cart, add-to-favorites, purchase). To give better recommendations, it is important to capture user intent, in addition to considering their historic interactions. However these intents are by definition latent, as we observe only a user's interactions, and not their underlying intent. To discover such latent intents, and use them effectively for recommendation, in this paper we propose an **A**ttentive **S**equential model of **L**atent **I**ntent (ASLI in short). Our model first learns item similarities from users' interaction histories via a self-attention layer, then uses a Temporal Convolutional Network layer to obtain a latent representation of the user's intent from her actions on a particular category. We use this representation to guide an attentive model to predict the next item. Results from our experiments show that our model can capture the dynamics of user behavior and preferences, leading to state-of-the-art performance across datasets from two major e-commerce platforms, namely Etsy and Alibaba.

KEYWORDS

Next Item Recommendation, User Modeling, Latent Intent

ACM Reference Format:
Md Mehrab Tanjim, Congzhe Su, Ethan Benjamin, Diane Hu, Liangjie Hong, and Julian McAuley. 2020. Attentive Sequential Models of Latent Intent for Next Item Recommendation. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3366423.3380002

1 INTRODUCTION

The basic goal of a recommender system is to recommend candidates from a large vocabulary of items a user might potentially interact with. To achieve this goal, various systems have been proposed which can learn users' preferences [7, 16, 23]. One popular category of techniques in industrial applications is Collaborative Filtering (CF), which leverages the observation that a user is most likely to interact with items that are similar to her historically interacted items [12]. To extend this idea, various models have further sought to capture the sequential dynamics of user feedback [10, 18]. Generally, these models are trained on a single interaction type (e.g. purchases or clicks), and do not further decompose predictions into different action or intent types.

In practice, user intent can change depending on context. Consider a typical setting from e-commerce: users can click items, add them to their favourites or carts, or purchase them if they fulfil their criteria. Depending on the intention and preferences of the user, she may be more likely to perform one type of interaction on an item at a particular point of time compared to others. For example, consider a case where the user intends to purchase a product; naturally, the items they will click, add to cart and eventually buy, will have similarities among them. So, to recommend the next purchase, it may be helpful to consider not only what the user has previously bought but also what products she has viewed or added to her cart in the past. Alternately, another user might simply browse products and add them to their cart with no immediate purchase intent; such items may share common properties (e.g. price, aesthetics) but may differ from those that are eventually purchased (e.g. cheaper alternatives). So, one intention (exploration) leads to different interactions compared to another (purchase). Thus we might try to recommend items that fulfill a user's current purpose. We argue that this is a potential limitation of current recommender systems that fail to differentiate between intent types (or even interaction types).

Based on the above observations, we seek a system that can find similarities among times, and also capture users' intentions. This task poses several challenges. First, although intentions and interactions are related, they are not the same; while we can directly observe a user's interactions, their intent is latent. Thus latent intent must be inferred from a user's interactions. Additionally, a user's intent might evolve gradually (long-term dependencies among interactions) or suddenly (short-term dependency), making it difficult to detect this drift from noisy action lists.

In this paper, we address these challenges and propose an **A**ttentive **S**equential **L**atent **I**ntent model (ASLI) which uses self-attention and temporal convolutional networks to find similarities among items, to capture users' hidden intents, and to attend on them. First, we apply a self-attention layer over all past items the user has interacted with to find item similarities from sequences; we then consider the interactions a user has performed on a given category and apply a temporal convolutional network to infer the user's latent intent. Finally we apply another attention layer to resolve both long-term and short-term dependencies between items and intentions. This proves to be effective at learning users' hidden preferences over an item. We further show the effectiveness of our model on real datasets from two e-commerce websites, namely Etsy and Alibaba. Our experiments show that the model is able to obtain state-of-the-art performance for sequential prediction tasks.

*Most work was done during an internship at Etsy

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380002

2 MOTIVATION

2.1 Actions on Categories as Interactions

In e-commerce settings, the number of items is typically significantly larger than the number of categories, such that user-item interactions are much sparser than user-category interactions. For example, for Etsy, the sparsity (the fraction of possible interactions that are observed) is only 0.02% at the user-item level but 1.68% at the user-category level. This means that inferring intentions from actions might be more difficult than inferring them from categories. To understand sparsity issues better, consider the case of a user who in 'discovery' mood and clicks items from 'Wedding Dress' category. As she finds interesting items, she adds them in the cart. But eventually she changes her intention to 'purchase' as she finds cheaper items from 'Gifts and Mementos' category. This is illustrated in Figure 1(a). It might be difficult to detect the subtle interest for a particular item from 'Gifts and Mementos' after the user interacted with a specific item from 'Wedding Dress' as they are only two instances out of many from their respective categories. However, the connection between 'Gifts and Mementos' and 'Wedding Dress' is much more succinct and as transition data from category to category is denser, we can learn the user's interest more easily. For these reasons, we consider users' category-wise interactions for the purpose of discovering latent intent.

2.2 Latent Factor Models of Intent

To better understand why intent is an important factor, we try to establish the dependency between the observed states, namely the items, and our defined interactions. If we wish to predict both, we would calculate their joint probability conditioned on the sequence of past items and interactions. Specifically, if we would like to calculate the probability of the next item p and next interaction i for a user u, then the joint probability can be expressed as $P(p, i|S_p^u, S_i^u)$. Here, S_p^u and S_i^u are the sequences of past items and interactions respectively. There are several ways we can factorize this probability based on assumption of interdependencies among the variables (e.g. items depend on interactions or vice versa). Based on the assumption, the factorization can become either simple or complex. For example, if we assume that items depend on interactions then the factorization becomes: $P(i|S_i^u)P(p|i, S_p^u, S_i^u)$. Similar constructions could be derived if we assumed interactions depend on items. We can simplify the factorization of the same joint probability if we assume that both items and interactions are conditionally independent given some other latent variable. Let us denote this variable as θ. Then, the joint probability would be factorized as follows:

$$P(p, i|\theta, S_p^u, S_i^u) = P(p|\theta, S_p^u) \times P(i|\theta, S_i^u)$$

Intuitively, the latent variable we introduce here can be defined as some representation of *intent* (shopping, browsing, discovering new items, etc.). We term this as the user's 'latent intent,' and design a model that can attend on it in order to give better recommendations.

3 PROPOSED MODEL

3.1 Problem Description

Notation is described in Table 1. We consider the problem of capturing users' hidden intent from their interacted items to obtain the most relevant next item recommendation. For that purpose, we define latent intent as a representation of users' interactions which can be used to predict (or 'explain') both the next item and the next interaction. Assume $\mathcal{P} = \{1, \ldots, p \ldots, |\mathcal{P}|\}$ is the set of all products, $\mathcal{C} = \{1, \ldots, c \ldots, |\mathcal{C}|\}$ is the set of all categories and $\mathcal{A} = \{1, \ldots, a \ldots, |\mathcal{A}|\}$ is the set of all actions. If a user takes an action a on an item p from a category c then $\mathbf{p}$, $\mathbf{c}$ and $\mathbf{a}$ are the corresponding d-dimensional embedding for product, category and action. As mentioned above, we have defined interactions as a combination of actions from a particular category, i.e., $\mathbf{i} = \mathbf{c} + \mathbf{a}$. Then we formulate our problem as follows: given a sequence of products and user interactions, $\mathcal{S}^u = \left(\langle \mathbf{p}^1, \mathbf{i}^1\rangle, \langle \mathbf{p}^2, \mathbf{i}^2\rangle, \ldots, \langle \mathbf{p}^{|\mathcal{S}^u|}, \mathbf{i}^{|\mathcal{S}^u|}\rangle\right)$, in each step, the goal is to capture hidden or latent intent and use it to predict both the next item and interaction, i.e., given $\left(\langle \mathbf{p}^1, \mathbf{i}^1\rangle, \ldots, \langle \mathbf{p}^{|\mathcal{S}^u|-1}, \mathbf{i}^{|\mathcal{S}^u|-1}\rangle\right)$, predict $\left(\langle \mathbf{p}^2, \mathbf{i}^2\rangle, \ldots, \langle \mathbf{p}^{|\mathcal{S}^u|}, \mathbf{i}^{|\mathcal{S}^u|}\rangle\right)$. Here, $|\mathcal{S}^u|$ is the length of the sequence for user u.

Table 1: Notation.

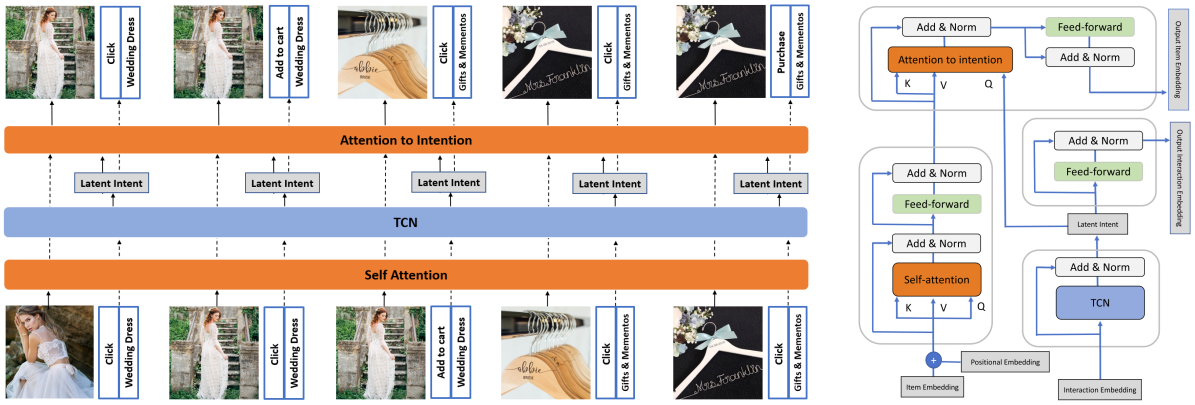
Notation	Explanation
$\mathcal{P}$	the set of products: $\{1, \ldots, p \ldots, \|\mathcal{P}\|\}$
$\mathcal{C}$	the set of categories: $\{1, \ldots, c \ldots, \|\mathcal{C}\|\}$
$\mathcal{A}$	the set of actions: $\{1, \ldots, a \ldots, \|\mathcal{A}\|\}$
$\mathbf{p}, \mathbf{c}, \mathbf{a} \in \mathbb{R}^d$	d-dimensional embedding for product p, category c and action a
$\mathcal{S}^u$	sequence of item and interaction tuples belonging to user u: $\left(\langle \mathbf{p}^1, \mathbf{i}^1\rangle, \ldots, \langle \mathbf{p}^{\|\mathcal{S}^u\|}, \mathbf{i}^{\|\mathcal{S}^u\|}\rangle\right)$
$\mathbf{o}_p, \mathbf{o}_i$	predicted embedding for product p and interaction i
r_{o_p}, r_{o_i}	scores for model prediction of product and interaction

3.2 Preliminaries

3.2.1 Self-Attentive Networks. The self-attention model is a recently proposed sequential model which achieved state-of-the art performance in various NLP tasks [21]. Self-attention first tries to calculate similarity scores between a query and a key, and use it as attention weights for a value. Here, queries, keys, and values can be the same objects (e.g. the sequence of items). Specifically, self-attention is defined as

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \tag{1}$$

where the $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the query, key, and value respectively. These are calculated by a linear projection of the input embedding. Specifically, $\mathbf{Q} = \mathbf{S}\mathbf{W}^Q$, $\mathbf{K} = \mathbf{S}\mathbf{W}^K$, $\mathbf{V} = \mathbf{S}\mathbf{W}^V$, where $\mathbf{S} \in \mathbb{R}^{n \times d}$ is a matrix describing a sequence of length n with d dimensional input embeddings, i.e., $\mathbf{S} = [\mathbf{e}^1; \ldots; \mathbf{e}^n]$, and $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times d}$ are projection matrices. The attention score between $\mathbf{Q}$ and $\mathbf{V}$ is divided by $\sqrt{d}$ to avoid large values of the dot product (especially with many dimensions). To maintain causality and to prevent information leaking from back to front, attention is modified by forbidding all links between $\mathbf{Q}_i$ and $\mathbf{K}_j$ ($j > i$). Note that as the self-attention operation is not aware of the order, each position is assigned a



(a) (b)

Figure 1: (a) Overview of our model. Here, an example use case is shown where the user initially has a 'discovery' intent which is reflected by interacting with aesthetic and expensive items. Eventually, she changes her intention toward cheaper and plainer looking items. (b) Detailed architecture

learned embedding which is added to the input embedding. This layer has a complexity of $O(n^2 d)$ because of Equation 1.

3.2.2 Temporal Convolutional Networks (TCN). In contrast to self-attention, convolution has a fixed context window. We can perform convolution in a number of different ways. In our task, we consider performing convolution in a 1D space (i.e., a sequence) based on a fixed kernel size which slides over the input sequence and determines the importance of context elements via a set of weights. The number of parameters can be reduced from $d^2 l$ to dl where l is the kernel width if we perform a depth-wise convolution with weight $\mathbf{W} \in \mathbb{R}^{d \times l}$. Specifically, the output for each element i in the input sequence $\mathbf{S}$ is calculated as follows:

$$\text{DepthwiseConv}(\mathbf{S}, \mathbf{W}_{c,:}, i, c) = \sum_{j=1}^{l} \mathbf{W}_{c,j} * \mathbf{S}_{(i+j-\lfloor (l+1)/2 \rfloor), c} \quad (2)$$

Here c is the current channel. Each dimension of the latent space can be considered as a channel, so usually $c = d$. The computational complexity of this layer is $O(ndl)$.

3.2.3 Feed-forward Network (FFN). Although both self-attention and convolutional models are able to aggregate sequential information through adaptive weights, they are still linear models. To introduce non linearity, the next step is to feed the outputs from these models to two layer feed-forward networks (FFNs). Specifically, if $\mathbf{o}^t$ is the output at step t (either from self-attention or TCN), then:

$$\text{FFN}(\mathbf{o}^t) = \mathbf{W}_2(\text{ReLU}(\mathbf{W}_1 \mathbf{o}^t + \mathbf{b}_1)) + \mathbf{b}_2$$

where ReLU is Rectified Linear Unit activation function [3], $\mathbf{W}_1$, and $\mathbf{W}_2$ are $d \times d$ weight matrices and $\mathbf{b}_1$ & $\mathbf{b}_2$ are d-dimensional bias vectors. We should note that FFN is applied point-wise, i.e. outputs from each step are taken as inputs separately. Therefore, there is no interaction between outputs from the two steps, and any leakage of information is prevented. As we are applying point-wise FFN, its computational complexity is $O(nd^2)$.

3.3 High-Level Overview of ASLI

We show a high level summary of ASLI in Figure 1(a). In ASLI, we first apply self-attention over the sequence of items to calculate similarities of items from all positions. We do not apply self-attention again to these outputs as we would like to probe which part of the item sequence is most relevant to users' hidden intent. To capture latent intent, intuitively speaking, we need a hidden representation from users' actions. As mentioned before, this poses a unique challenge as users' item-wise actions are sparse. To alleviate sparsity issue, one of our key modeling decisions is to treat category-wise actions as interactions. In ASLI, we choose TCN to get features from these interactions as it is relatively shallow and easily parallelizable. Later, to make sure this latent feature captures intent, we use it for dual prediction of both the next interaction (by a feedforward network) and the next item (by a co-attentional transformer layer). We take co-attention between the first layer's output (which calculates item similarities) and discover hidden intents as we would like to resolve both long-term and short-term dependencies between them and better learn items' relevance with respect to users' intent.

3.4 ASLI

3.4.1 Embedding Layer: We show the detailed architecture of ASLI in Figure 1(b). We first transform the training sequence $\mathcal{S}^u$ for user u into a fixed length (n) sequence where n represents the maximum number of steps that our model can process. For item p, category c and action a, we have corresponding embeddings $\mathbf{p}, \mathbf{c}, \mathbf{a}$. From $\mathbf{c}$ and $\mathbf{a}$, we construct the interaction embedding, i.e. $\mathbf{i} = \mathbf{c} + \mathbf{a}$. If the sequence length is greater than n, we consider the most recent n actions. If the sequence length is less than n, we pad on the left until the length is n. A constant zero vector $\mathbf{0}$ is used for padding.

3.4.2 Self-Attention Layer. The goal of this layer is to discover similarities among items based on users' sequential interactions. We can calculate similarity scores among items from different positions by applying self-attention on the item sequence. We consider a learnable positional embedding $\mathbf{t}$ for the current step or position t which we add to the current item embedding $\mathbf{p}$ [21]. This way we build a sequence matrix, $\mathbf{S}_p$ and use it to calculate the query, key and

value, i.e, $\mathbf{Q} = \mathbf{S}_p\mathbf{W}^Q$, $\mathbf{K} = \mathbf{S}_p\mathbf{W}^K$, $\mathbf{V} = \mathbf{S}_p\mathbf{W}^V$. Finally, we apply Equation 1 and get an output. Then, we apply residual connection to leverage any low-level information [5] and layer normalization [1] to stabilize and accelerate training. The bottom-left block in Fig. 1(b) shows its architecture.

3.4.3 TCN Layer. In this layer, we apply depth-wise convolution (Equation 2), DepthwiseConv($\mathbf{S}_i, \mathbf{W}$), followed by residual connection and layer normalization, to the sequence matrix of interactions $\mathbf{S}_i$. This layer (shown as the bottom-right block in Fig. 1(b)) gives us the latent representation of intent which we use as a query to predict the next item. We also use it as an input to a feedforward network which gives us an embedding, $\mathbf{o}_i^t$, to predict the next interaction (the middle block in Fig. 1(b)).

3.4.4 Attention to Intention Layer. Finally, to find the relevance of items with latent intent, we treat the outputs from the first self-attention layer as keys and values and the outputs of the TCN as queries for to another self-attention layer. As the query is a latent representation of intent we call this the Attention to Intention Layer. Output from this layer is taken as input to another feedforward network which outputs an embedding, $\mathbf{o}_p^t$ for predicting the next item. The top block in Fig. 1(b) shows its overall architecture.

3.4.5 Loss Function. For training the model, we adopt a point-wise loss where we consider one positive example and one negative example [9, 13]. As we are predicting both the next item and next interaction in each step, we have one ground truth for each, namely p and i. For predicting the next item, we randomly sample a negative item (an item which the user has not interacted with before) from the data. Then we calculate the dot product between the model output and positive/negative example, and obtain a score. This score is used to calculate a ranking loss, $\mathcal{L}_{\text{ranking}}$ for the next item in the following manner:

$$\mathcal{L}_{\text{ranking}} = -\sum_{S^u \in \mathcal{S}} \sum_{t \in [1,\ldots,n]} \left[\log(\sigma(r_{o_p^t})) + \log(\sigma(1 - r_{o_{p'}^t}))\right]$$

Here, $r_{o_p^t}$ is the dot product for the positive item p, and $r_{o_{p'}^t}$ is the score for the negative item p'. Similarly, for predicting the next interaction, we have the ground truth next interaction i. Then we randomly sample a combination of a category-action pair which is not observed for that user, and construct the negative interaction i'. We similarly calculate the scores and calculate the interaction loss. Specifically,

$$\mathcal{L}_{\text{interaction}} = -\sum_{S^u \in \mathcal{S}} \sum_{t \in [1,\ldots,n]} \left[\log(\sigma(r_{o_i^t})) + \log(\sigma(1 - r_{o_{i'}^t}))\right]$$

Here, $r_{o_i^t}$ and $r_{o_{i'}^t}$ are the scores for the positive and negative interaction. Our final loss is $\mathcal{L} = \mathcal{L}_{\text{ranking}} + \mathcal{L}_{\text{interaction}}$.

4 EXPERIMENTS

In this section, we present our experimental setup and empirical results. Our experiments are designed to answer the following research questions: **RQ1:** How does our model which captures latent intent perform compared to other recommendation models? **RQ2:** What is the influence of various components of our architecture? **RQ3:** What is the training efficiency and scalibility of our model?

	Tmall	Tmall-Small	Etsy
# users	9,883	6,280	6,690
# items	569,658	47,759	119,310
# categories	6,352	130	608
# actions	2.45M	0.44M	0.22M
avg. actions/user	248.10	71.98	34.17
avg. views/user	211.32	62.84	27.31
avg. add-to-favorites/user	11.66	3.01	3.37
avg. add-to-carts/user	17.15	4.22	2.47
avg. purchases/user	7.97	1.91	1.02

Table 2: Data statistics after pre-processing

4.1 Datasets

To demonstrate ASLI's performance, we consider datasets from two popular e-commerce websites: Alibaba and Etsy. Table 2 shows the statistics of these datasets after preprocessing. Following the settings of the next item recommendation [13, 24], for each user, we test on the last item, validate on the item before that, and train with the rest of the sequence.

Tmall[1]**.** Tmall is a publicly available dataset provided by Alibaba. Originally, it contains around 12 million actions from 10,000 user records. There are four types of actions the a user can take, namely, click, add-to-favorite, add-to-cart and purchase. For pre-processing, we followed the same procedure from [7, 8, 18] and removed any user or item with fewer than 5 interactions. Also, note that some items may be clicked many times by a user which may introduce bias toward certain actions. To remedy this we consider only the first such action the user has taken. After preprocessing we get 9,883 users, 569,658 items, 6,352 categories, and 2,454,115 actions.

Etsy. Etsy is an e-commerce platform focused on personalized and handmade items. The dataset contains the activity logs of users with accounts on the recommendation module collected between Oct. 15, 2018 to Dec. 15, 2018. The data were filtered following [4], such that any items or users with fewer than 20 actions are removed. After processing, it contains 6,690 users, 119,310 items, 608 categories, and 215,227 actions.

For fair comparison, we also prepare a version of Tmall, namely Tmall-Small, which is pre-processed in the same way as Etsy. It contains 6,280 users, 47,759 items, 130 categories, and 439,497 actions.

4.2 Evaluation Metrics

We report results for two popular top-k metrics in recommender systems, namely: HitRate@k and NDCG@k [7, 9]. HitRate@k is the fraction of times the ground-truth item appears among the top-k predicted items, while NDCG@k (Normalized Discounted Cumulative Gain) is a position-aware metric which assigns larger weights to higher positions. Specifically, if the ranking of the ground truth item for a user is rank_u and $\text{rank}_u \leq k$, then NDCG@k for that user is calculated as follows: $\frac{1}{\log_2(\text{rank}_u+1)}$. If $\text{rank}_u > k$, NDCG@k is 0. For ranking items, we consider 100 negative samples for each ground-truth item. For values of k, we choose 5 and 10.

4.3 Baselines

To show the effectiveness of our model, we compare ASLI with two groups of recommendation baselines. The first group contains general non-sequential recommender models:

[1] https://tianchi.aliyun.com/dataset/dataDetail?dataId=46

Dataset	Tmall				Tmall-Small				Etsy			
	NDCG		HitRate		NDCG		HitRate		NDCG		HitRate	
	k = 5	k = 10	k = 5	k = 10	k = 5	k = 10	k = 5	k = 10	k = 5	k = 10	k = 5	k = 10
Most-Pop	0.1688	0.2030	0.2459	0.3516	0.1271	0.1577	0.1898	0.2848	0.0539	0.0691	0.08116	0.1283
BPR-MF	0.3541	0.3862	0.4608	0.5601	0.2688	0.3124	0.3774	0.5127	0.2724	0.2800	0.2849	0.3086
NextItRec	0.4301	0.4514	0.5028	0.5686	0.3981	0.4235	0.4705	0.5493	0.3543	0.3671	0.3866	0.4265
SASRec	0.4771	0.5028	0.5436	0.6230	**0.4525**	**0.4773**	0.5269	0.6037	0.3765	0.3828	0.3927	0.4123
ASLI	**0.5133**	**0.5367**	**0.6051**	**0.6769**	0.4334	0.4678	**0.5367**	**0.6429**	**0.3946**	**0.4015**	**0.4310**	**0.4523**

Table 3: Performance of various models

Attention Type	Tmall		Etsy	
	NDCG@5	HR@5	NDCG@5	HR@5
Seq-Seq (SASRec)	0.4753	0.5434	0.3792	0.3976
Seq-Item	0.3641	0.4297	0.2542	0.2773
Seq-Action	0.3766	0.4644	0.3386	0.3658
Seq-Category	0.4787	0.5690	0.3788	0.4189
Seq-Latent Intent	**0.5101**	**0.5955**	**0.3940**	**0.4300**

Table 4: Impact of different attentions on both datasets

- **MostPop:** MostPop is a simple baseline which ranks items based on their popularity.
- **BPR-MF:** Bayesian personalized ranking factorizes the user-item interaction matrix using a ranking loss [17].

The second group contains two recently proposed sequential deep learning models:

- **NextItRec:** This is a convolutional model for next item recommendation proposed in [24]. NextItRec uses 1-D dilated convolutional networks to obtain the context of the past L items.
- **SASRec:** This is a sequential model based on self-attention for next item recommendation [13].

As other sequential models (such as FPMC [18], Fossil [8], improved GRU4Rec [10], CASER [20]) have been outperformed by the sequential models mentioned above, we omit comparisons against them.

4.4 Implementation Details

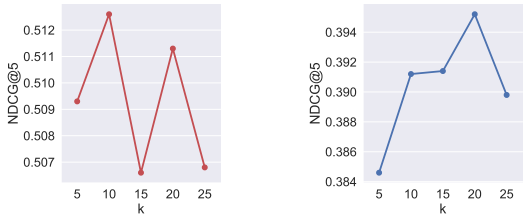
We implement our proposed model in TensorFlow, and conduct all experiments with a single GPU (NVidia 1080). We use an Adam optimizer [14] to update model parameters. Based on the average number of actions per user, we fix 50 as the maximum sequence length for Etsy, 100 for Tmall-Small, and 300 for Tmall. The batch size is 32 for NextItRec and 128 for the rest. For all models, 200 latent dimensions yielded satisfactory results. To tune other model specific hyper-parameters, we followed the strategies suggested by the methods' authors. For our model, we set the dropout rate to 0.3 after performing a grid search from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the learning rate to 0.001 from $\{0.1, 0.01, 0.001, 0.0001, 0.00001\}$. We also tune the kernel size for TCN layer which we discuss later on.

4.5 Performance

To answer **RQ1**, we report the performance of each of the models in Table 3. From the table, our first observation is that the group of all sequential deep-learning models outperform the non-sequential group. Between the two baseline models from the first group, BPR-MF performs significantly better than popularity-based model, MostPop. Although BPR-MF can learn static user preference, it cannot capture sequential dynamics. Hence, all deep sequential models which are capable of learning such dynamics, outperform BPR-MF. Among the sequential models, we can observe that NextItRec generally performs better in shorter sequence data (Etsy) than longer ones (both Tmall and Tmall-Small). Moreover, it achieves higher HitRate@20 (0.4265 vs 0.4123) for Etsy than the self-attention based model SASRec. Presumably, this owes to the fixed size of dilated kernels in NextItRec enabling it to capture sequential patterns for short-length sequences better than for longer ones. Apart from this, SASRec performed better than NexitItRec in general. Overall, ASLI achieves the best performance under all metrics for all datasets except Tmall-Small under NDCG where its performance was worse than SASRec (roughly 4% less NDCG for $k = 5$ and 2% less for $k = 10$). This result shows one of the most important dependencies of our model on the number of categories. This dependency can be better explained using an example of a dataset with only one category. In this case, our model will treat all category-wise action almost equally, and therefore, it will not be able to fully extract useful patterns for discovering intent. This example, however simple, provides an intuitive reasoning for explaining the performance gap for the Tmall-Small dataset. Of note, Tmall-Small was pre-processed using Etsy's scheme, and there are only 130 categories for this dataset which may be responsible for the performance drop in NDCG. In both Tmall and Etsy dataset, the number of categories is much higher (above 500) and our model achieves the best performance under all metrics. For example, the improvement under NDCG for these two datasets is at least **4.8**%. Under HitRate, it is **6**% or more. Next we discuss the impact of different attentions and the effect of kernel size l to answer **RQ2**.

4.6 Impact of Attention to Latent Intent

For our choice of query to the last attention layer, we could potentially use embeddings for items, actions or categories in addition to latent intent. Table 4 compares these choices of query. Here 'Seq' denotes the sequence output from the first self-attention layer. SASRec tries to find intrinsic properties of items from the first layer and uses 'Seq-Seq' attention to resolve any dependencies that are missed in the first layer. 'Seq-Item' denotes using item embeddings as queries in the second layer. Seq-Item performs worse than Seq-Seq, possibly because it redundantly calculates the similarity of



(a) Tmall (b) Etsy

Figure 2: Effect of kernel size l on performance.

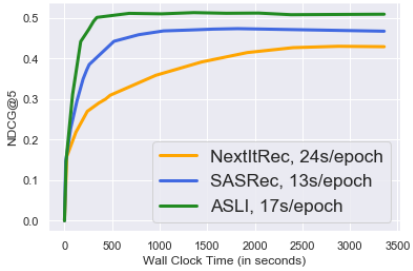


Figure 3: Training time on Tmall

items to sequences and may lose the resemblance found in the first layer. When actions or categories are used, performance improves significantly, showing that both are important factors and thus using both as interactions for discovering latent intent and paying attention to it leads to better recommendations.

4.7 Effect of kernel size l on performance

The kernel size l in TCN layer is a key hyper-parameter in our model. For analyzing its effect, we vary its size from 5 to 25 (by intervals of 5), and show NDCG@5 for both datasets in Fig. 2. If we increase l, we are working with a larger context size for extracting latent intent which may lead to a performance gain. Interestingly, from the figure, we make an important observation that increasing l arbitrarily may not lead to better performance. In our case, kernel sizes of 10 and 20 worked better for Tmall and Etsy. The following discusses experimental results for answering **RQ3**.

4.8 Training Efficiency and Scalibility

To emprically show training efficiency of our model, we test how fast our model converges compared to other sequential models. Figure 3 shows the convergence results for the Tmall dataset. We see that ASLI converges faster than the other two (within 20 epochs or roughly 350 seconds) although its per-epoch time is more than SASRec's. We think the main reason for this is that our joint optimization for both next item and next interaction helps the model to quickly learn about the relashionships of items. In terms of scalability, we should note that the overall computation complexity of our model is $O(n^2d + ndl + nd^2)$ (due to self-attention, TCN and FFN layer). As n is typically larger than d, the complexity is dominated by the n^2d term. However, both self-attention and the TCN layer are fully parallelizable in the GPU and can easily scale. For example, when we increased n from 50 to 600 for Tmall, the per epoch time increases from 7 sec. to 45 sec. (full results not shown for brevity).

5 RELATED WORK

Traditionally, recommender system focus on users and items via interaction matrices. These interactions can be explicit (e.g. ratings) or implicit (e.g. clicks, purchases, comments, etc.) [12, 17]. Popular approaches include Matrix Factorization (MF) methods which aim to uncover latent dimensions from interaction matrices [15, 19]. Modeling implicit behavior is challenging due to the ambiguous interpretation of 'non-observed' data (i.e., items the user did not interact with). To address this problem, point-wise [12] and pairwise [17] methods have been proposed.

Recommendations can be improved by accounting for temporal dynamics. For example, TimeSVD++ [2], achieved strong results on the Netflix challenge, by splitting time into several segments and modeling users and items separately in each. Such models are essential to understand datasets that exhibit temporal 'drift' [2, 23].

Sequential recommendation (or next-item recommendation) is slightly different from temporal recommendation in that it only considers the order of actions rather than timestamps. FPMC uses matrix factorization, augmented with an item-item transition term to capture long-term preferences and short-term transitions [18]. Since the previous item provides 'context' for the next action, first-order MC based methods show strong performance, especially on sparse datasets [7].

Some methods adopt high-order MCs that consider more previous items [6], [8]. For example, GRU4Rec uses Gated Recurrent Units (GRU) to model click sequences for session-based recommendation [11], and an improved version further boosts its Top-N recommendation performance [10]. In each time step, RNNs take the state from the last step and current action as its input. These dependencies make RNNs less efficient, though techniques like 'session parallelism' have been proposed to improve efficiency [11]. Convolutional networks have recently been applied in sequential recommendation settings. CASER [20] views the embedding matrix of the L previous items as an 'image' and applies convolutional operations to extract transitions while NextItRec [24] applies 1-D dilated convolutions to get the context. Self-attention based [13, 25] networks have also been proposed for sequential recommendation due to their tremendous success in various NLP tasks. Recently, [22] proposed a mixture-channel purpose routing network to model different purposes of items in anonymized sessions. Although these methods can model dynamic user preferences or item purposes, they do not consider the influence of user intent. We view recommendation as a joint task, i.e., predicting both the next interaction and the next item, and aim to model both via a unified framework to discover users' latent intent.

6 CONCLUSION

In this paper, we proposed ASLI, which achieves better recommendations by capturing users' hidden intents from their interactions. In addition to finding similarities among items, ASLI makes use of a TCN layer to obtain latent representation of users' intentions, which are used to query an attention layer to find which items are most relevant to users' intents. Through experiments, we find that our model outperforms the current state-of-the-art on two real-world datasets.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Koren Y Collaborative. 2010. filtering with temporal dynamics [J]. *Communications of the Acm* 53, 4 (2010), 89–97.

[3] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 315–323.

[4] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[6] Ruining He, Chen Fang, Zhaowen Wang, and Julian McAuley. 2016. Vista: A visually, socially, and temporally-aware model for artistic recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 309–316.

[7] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 161–169.

[8] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 191–200.

[9] 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.

[10] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 843–852.

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

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets.. In *ICDM*, Vol. 8. Citeseer, 263–272.

[13] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

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

[15] Yehuda Koren and Robert Bell. 2015. Advances in collaborative filtering. In *Recommender systems handbook*. Springer, 77–118.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[17] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[18] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 811–820.

[19] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

[20] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 565–573.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[22] Shoujin Wang, Liang Hu, Yang Wang, Quan Z Sheng, Mehmet Orgun, and Longbing Cao. 2019. Modeling multi-purpose sessions for nextitem recommendations via mixture-channel purpose routing networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 1–7.

[23] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. 2010. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM, 211–222.

[24] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 582–590.

[25] Shuai Zhang, Yi Tay, Lina Yao, Aixin Sun, and Jake An. 2019. Next item recommendation with self-attentive metric learning. In *Thirty-Third AAAI Conference on Artificial Intelligence*, Vol. 9.