

# Log2Intent: Towards Interpretable User Modeling via Recurrent Semantics Memory Unit

Zhiqiang Tao\*  
Northeastern University  
zqtao@ece.neu.edu

Sheng Li  
University of Georgia  
sheng.li@uga.edu

Zhaowen Wang  
Adobe Research  
zhawang@adobe.com

Chen Fang  
ByteDance AI Lab  
fangchen@bytedance.com

Longqi Yang  
Cornell University  
ly283@cornell.edu

Handong Zhao  
Adobe Research  
hazhao@adobe.com

Yun Fu  
Northeastern University  
yunfu@ece.neu.edu

## ABSTRACT

Modeling user behavior from unstructured software log-trace data is critical in providing personalized service (e.g., cross-platform recommendation). Existing user modeling approaches cannot well handle the long-term temporal information in log data, or produce semantically meaningful results for interpreting user logs. To address these challenges, we propose a Log2Intent framework for interpretable user modeling in this paper. Log2Intent adopts a deep sequential modeling framework that contains a temporal encoder, a semantic encoder and a log action decoder, and it fully captures the long-term temporal information in user sessions. Moreover, to bridge the semantic gap between log-trace data and human language, a recurrent semantics memory unit (RSMU) is proposed to encode the annotation sentences from an auxiliary software tutorial dataset, and the output of RSMU is fed into the semantic encoder of Log2Intent. Comprehensive experiments on a real-world Photoshop log-trace dataset with an auxiliary Photoshop tutorial dataset demonstrate the effectiveness of the proposed Log2Intent framework over the state-of-the-art log-trace user modeling method in three different tasks, including log annotation retrieval, user interest detection and user next action prediction.

## KEYWORDS

User Modeling; Log-trace Data; Sequential Modeling; Recurrent Memory Network; Semantics Attention

### ACM Reference Format:

Zhiqiang Tao, Sheng Li, Zhaowen Wang, Chen Fang, Longqi Yang, Handong Zhao, and Yun Fu. 2019. Log2Intent: Towards Interpretable User Modeling via Recurrent Semantics Memory Unit. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19)*, August 4–8, 2019, Anchorage, AK, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3292500.3330889>

\*Work partially done while intern at Adobe Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

KDD '19, August 4–8, 2019, Anchorage, AK, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6201-6/19/08...\$15.00

<https://doi.org/10.1145/3292500.3330889>

## 1 INTRODUCTION

User modeling [11, 27, 40] enables personalized experience of software and web service, with a wide range of applications in recommender system, personalized social web, online advertising, email marketing, intelligent user interfaces, etc. Most existing user modeling approaches have benefited largely from the *structured* user behavior data, such as texts, images, ratings, etc. For example, collaborative filtering methods decompose the incomplete user-movie rating matrix into latent factors and predict whether a user will be interested in a movie she/he hasn't watched before; email marketing systems achieve behavioral targeted advertising, by segmenting users into multiple groups based on the structured data like demographics and online purchase history. However, user modeling from *unstructured* user behavior data has not been extensively studied before. One representative type of unstructured user behavior data is the software log-trace data, which can be observed in many complex software, such as the graphic design software Photoshop, computer-aided design and manufacturing software CAD/CAM, enterprise resource planning software SAP ERP, etc. This paper focuses on modeling user behaviors, predicting and interpreting user intents from the unstructured software log-trace data, in order to improve the user experience.

The major challenges in modeling the unstructured user behavior log-trace data are two-fold. First, *how to encode the long-term temporal information to user modeling?* Util2Vec [38] is the most relevant work to ours, which extracts discriminative embeddings from the log-trace data. However, it neglects to model the long-term temporal context in user logs. We propose to use deep sequential modeling based on recurrent neural networks (RNN) to address this issue. Although many RNN based models have been introduced to sequential recommendation [14], they haven't been applied to analyzing software log-trace data yet. Second, *how to achieve interpretable results for user modeling?* As the software log-trace data are usually unstructured and hard to understand, there is a big semantic gap between the log data and human language. To bridge such a semantic gap, we propose to leverage auxiliary text datasets (e.g., software tutorials) and further interpret user logs in a semantically meaningful way.

In this paper, we propose a Log2Intent framework for interpretable user modeling. By formulating the user log modeling as a representation learning problem, our Log2Intent framework extracts low-dimensional user embedding from unstructured software log-trace data and auxiliary tutorial data. In particular, the deep sequential modeling is employed to capture the long-term temporal

information, in which a memory unit is designed to exploit the auxiliary tutorial data and generate interpretable results. The sequential modeling part accounts for the temporal information from each user session, which is implemented by a temporal encoder, a semantic encoder, and a log action decoder. The temporal encoder models the contextual information between log action sequences. Inspired by the end-to-end MemNet [29] and the recurrent memory network [31], we propose a recurrent semantics memory unit (RSMU) that captures the input from tutorial data. Then, the semantic encoder fuses the memory output of RSMU and the hidden state of temporal encoder. Finally, the log action decoder is fed with the previous action and it predicts the next action conditioning on the last hidden state. The framework of Log2Intent is shown in Fig. 2. Extensive experiments on a real-world Photoshop log-trace dataset with an auxiliary Photoshop tutorial dataset demonstrate the effectiveness of our framework in the following three tasks: (1) interpreting log-trace data by retrieving relevant annotations from tutorials; (2) exploring user interests by predicting the associated tags; (3) predicting the next user action. As the proposed Log2Intent framework is indeed a general solution to sequential modeling of log data, it could be applied to many other log analysis tasks (e.g., web browsing analytics) beyond the software log-trace data.

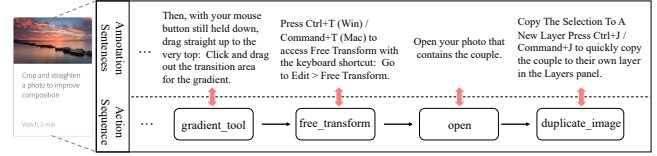
**Contributions.** In summary, we highlight the major contributions of this work as follows.

- We propose the Log2Intent framework for user behavior modeling, by developing a novel deep sequential model with a temporal encoder, a semantic encoder and a log action decoder network.
- We design a recurrent semantics memory unit (RSMU) to dynamically capture the semantic information provided by an auxiliary tutorial dataset, conditioning on temporal context. The semantics attention mechanism inside RSMU enables our model to interpret the user log data with the attended annotations (*i.e.* memory slots) from tutorials.
- We perform comprehensive evaluations of our model and baselines on the real-world software log-trace dataset. Our Log2Intent approach obtains remarkable improvements over baselines in three tasks including log annotation retrieval, user interest detection and next action prediction.

## 2 DATASET

We provide a general method that is applicable to any software as long as it records 1) user log trace and has 2) text tutorials. In this paper, we implement our idea upon the Adobe Photoshop platform with following reasons. First, Photoshop is one of the most representative software for users to complete complex and long-period tasks, which provides plenty of long action sequences, thus sufficient temporal context, to our study. Second, there are many online Photoshop tutorials that allow us to obtain text annotation/explanation for action sequence. Third, Photoshop users cover a wide range of occupations such as graphic designer, advertiser and artist, who share different user interests and professional habits. Hence, it meets the assumption of our approach for personalized learning. We give the details of our data collection as follows.

**User-Log Dataset.** Photoshop software records the actions conducted in the application for all the users who agree with the usage



**Figure 1: Illustration of action/annotation sequence in the tutorial dataset, where each user action is described by an annotation sentence.**

reporting. These user actions include the buttons clicked, tools selected and features applied, such as [open], [undo], [move]; [horizontal\_type\_tool], [brush\_tool], [stamp\_pickup\_tool]; [deselect], [clone\_stamp] and [stepbackward]. Hence, the log-trace record is a sequence of user actions. We collected such log sequence data from U.S. Photoshop users in February 2018, resulting in 611,628 users with totally 1,865,636 sessions of around 0.41 billion actions. Each session is corresponding to user activity per time, and has an average length of 220 actions. All these logs share with a vocabulary of 940 unique actions after filtering out the low frequency ones.

**Tutorial Dataset.** We use the tutorial dataset [37] collected from 2,022 online Photoshop tutorials. After preprocessing this dataset with the same vocabulary in our User-Log dataset, we remain 1,781 tutorials of 49,763 actions covering 426 high frequency actions in the vocabulary. Each action is corresponding to one human annotation per tutorial. Thus, we also get 49,763 sentences with a vocabulary of 11,223 words. It is worth noting that, the same software action is usually annotated by different sentences according to different action context. One tutorial example is shown in Fig. 1, where each tutorial contains two sequences: 1) software actions performed in this tutorial and 2) a sequence of annotation sentences along with time steps for each action. We collect all the annotated sentences for each action as its specific auxiliary knowledge.

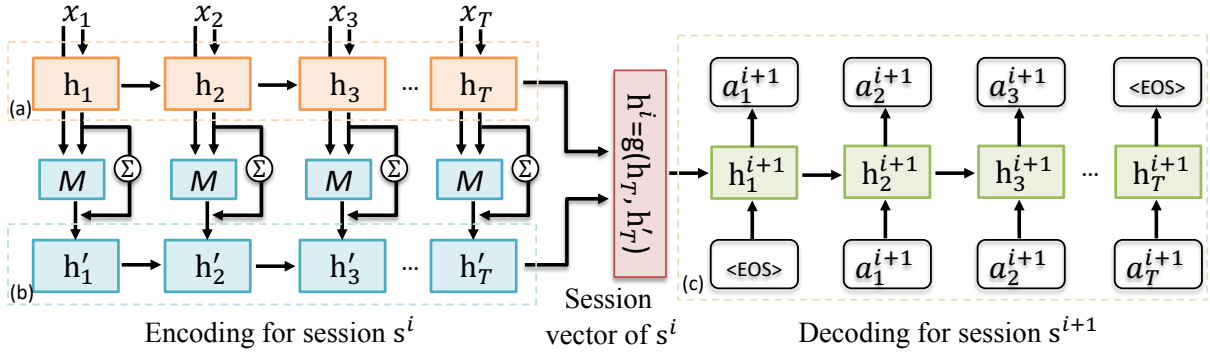
## 3 METHODOLOGY

### 3.1 Problem Formulation

Software log-trace data provide large amount of personalized information collected from daily life, which is useful to enrich user profile and describe user behavior. In this work, we target to mine user’s intent from log data. Specifically, we denote  $\{u\}$  as a set of users, each of which has a long log-trace history consisting of a sequence of sessions, *i.e.*,  $u \leftarrow [s^1, s^2, \dots, s^N]$ , where  $s^i$  represents the  $i$ -th session and  $N$  denotes the number of user sessions. Each session is a sequential data denoted by  $s^i = [a_1^i, a_2^i, \dots, a_T^i]$ , where  $a_t^i$  represents the  $t$ -th software action taken by user  $u$  in the  $i$ -th session. Let  $T$  be the length of  $s^i$  and  $V_a$  be the vocabulary for all the unique software actions. We focus on the following questions.

- (1) *Can we interpret log-trace data in a human readable way?*
- (2) *Can we learn a compact user representation from log history?*
- (3) *Can we predict next user action by giving the previous ones?*

To address the above challenges, we propose a *Log2Intent* model to incorporate semantics memory units into sequence to sequence learning. Moreover, we leverage auxiliary knowledge to enrich and interpret the log sequential data. For each user action (*i.e.*,  $\forall a \in V_a$ ), we collect log annotations from software tutorials as its own semantic “memory” slots. Thus, each action  $a$  is corresponding to one



**Figure 2: Illustration of the proposed Log2Intent model. (a) The temporal encoder network models the context information between action sequence; (b) the semantic encoder fuses the memory output as well as the hidden state in (a) at each time step; (c) the action decoder works as a “language” model, which is fed with the previous user action and predicts the next action conditioning on the last hidden state. Each action has its own memory unit (see Fig. 3) and attends to different memory slots upon the temporal context.**

memory unit denoted as  $M = \{m_k\}_{k=1 \dots K}$ , where  $m_k$  represents the  $k$ -th memory slot and  $K$  is the size of a memory unit. The memory slot  $m_k$  is a sentence of  $L$  words, denoted by  $[w_1^k, \dots, w_L^k]$ . We refer to  $V_w$  as the word vocabulary shared by all the memory units. **Notations.** We present the details of our model in the next. For convenience, we omit the superscript  $i$  for different sessions and actions when no confusion occurs, and always assume a user action  $a \in V_a$  is corresponding to its own memory unit  $M$  without specific instructions. Moreover, we suppress all the bias vectors in the projection layer for readability. Vectors and matrices are denoted as bold lowercase and capital letters, respectively.

**3.1.1 Embedding layers.** Embedding layer is widely used by recent deep recurrent neural networks to project discrete variables into continuous vector space, where the general idea is to adopt an embedding matrix (i.e., a look-up table) to index input variable. In our model, we also learn action and memory embeddings to enable gradient backpropagation for network training. In detail, let  $\mathbf{A} \in \mathbb{R}^{|V_a| \times d_a}$  be the embedding matrix for actions, then action  $a_t$  is represented by

$$\mathbf{x}_t = \mathbf{A}(a_t), \quad (1)$$

where  $d_a$  represents the dimension for action embedding,  $1 \leq a_t \leq |V_a|$  indexes the  $a_t$ -th row in matrix  $\mathbf{A}$ . In a similar way, we can also vectorize each memory slot (a sentence) in  $M$ . Following [29], we define two memory look-up tables for addressing and reading memories, respectively, such as  $\mathbf{M} \in \mathbb{R}^{|V_w| \times d_m}$  and  $\mathbf{O} \in \mathbb{R}^{|V_w| \times d_m}$ , where  $d_m$  refers to the embedding dimension for all the memories. Then, we represent the memory slot  $m_k$  by

$$\mathbf{m}_k = \sum_l \eta(l) \mathbf{M}(w_l^k), \quad (2)$$

$$\mathbf{o}_k = \sum_l \eta(l) \mathbf{O}(w_l^k), \quad (3)$$

where  $1 \leq l \leq L$  indexes the  $l$ -th word in  $m_k$  and  $\eta(l)$  is a weight function w.r.t the word position in one sentence as defined in [29]. As can be seen, Eqs. (2-3) give memory embedding as a weighted sum of word embeddings within the sentence  $m_k$ ,  $1 \leq k \leq K$ .

## 3.2 Session to Session

Log trace is in essence sequential data that contain the temporal context varying from different tasks and user habits. Hence, it will be useful to capture the temporal information from each user session. Inspired by recent deep sequential modeling [19, 30], we leverage a *session to session* strategy as  $s^i \rightarrow s^{i+1}$ . Fig. 2 shows the overall architecture of our model. We develop an encoder-decoder framework to model the user behaviors, which mainly involves two encoder networks and one decoder network as follows.

**3.2.1 Temporal Encoder.** Given one session  $s = [a_1, a_2, \dots, a_T]$ , we first design a temporal encoder network  $f_{enc-T}$  as

$$\mathbf{h}_t = f_{enc-T}(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta_{enc-T}), \quad (4)$$

where  $\mathbf{x}_t$  is the action embedding for  $a_t \in s$  and  $\theta_{enc-T}$  refers to all the learnable parameters for  $f_{enc-T}$ . By using Eq. (4), we generate a hidden representation  $\mathbf{h}_t$  for each time step with new arrival action and previous states. Thus, with such a recurrent model, the entire session  $s$  could be represented by the last hidden state  $\mathbf{h}_T$ .

Several popular recurrent neural networks (RNNs) can be used for  $f_{enc-T}$ , such as the Long Short-Term Memory (LSTM) [15] and Gated Recurrent Unit (GRU) [7, 8]. In our model, we adopt GRU as it empirically has a similar performance to LSTM for log sequence yet a more simple structure. In detail, GRU is defined by

$$\begin{aligned} \mathbf{r}_t &= \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1}), \\ \mathbf{z}_t &= \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1}), \\ \hat{\mathbf{h}}_t &= \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1})), \\ \mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \hat{\mathbf{h}}_t, \end{aligned} \quad (5)$$

where  $\mathbf{r}_t$  and  $\mathbf{z}_t$  respectively represent the reset gate and update gate,  $\odot$  denotes the Hadamard product, and  $\hat{\mathbf{h}}_t$  is the candidate state proposed at each time step. The final hidden state  $\mathbf{h}_t$  emitted at time  $t$  is a linear interpolation between the previous state  $\mathbf{h}_{t-1}$  and the candidate  $\hat{\mathbf{h}}_t$ . More specifically, let  $e$  be our temporal encoder, we encapsulate the temporal information within  $\forall s$  into a single hidden representation as

$$\mathbf{h}_T = e([\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T]), \quad (6)$$

where  $e$  is parameterized by  $f_{enc-T}$  defined in Eq. (4), and  $\theta_{enc-T}$  thus includes all the GRU parameters in Eq. (5) plus the action embedding matrix  $A$ .

**3.2.2 Semantic Encoder.** Our model jointly utilizes two-source information, *i.e.*, user action sequence from log data and the annotation sentences for each action collected from software tutorials. However, how to integrate the sentence information into the encoding process is quite challenging, especially when we aim to keep tracking sentences to obtain the interpretability, which means we cannot simply merge all the sentences as a semantics view for each action. To address this challenge, we propose a recurrent semantics memory unit (RSMU) to dynamically fetch the memory slot (*i.e.*, sentence) from each action's memories. We will introduce more about RSMU in Section 3.3. Let  $c_t$  be the output from RSMU for action  $a_t$ , then we have our semantic encoder function as

$$\mathbf{h}'_t = f_{enc-S}(\mathbf{v}_t, \mathbf{h}'_{t-1}; \theta_{enc-S}), \quad (7)$$

where  $\mathbf{v}_t = \mathbf{c}_t + \mathbf{h}_t$ ,  $\mathbf{h}'_t$  denotes the semantic hidden state at time  $t$ , and  $f_{enc-S}$  is also formulated by GRU with  $\theta_{enc-S}$  including all the related parameters in the semantics encoding pathway. We employ  $f_{enc-S}$  to build the connection between different RSMUs along with the time steps, which focuses on modeling the relationship between different user actions' memory context. A semantic representation for session  $s$  is obtained by

$$\mathbf{h}'_T = e'([\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_T]), \quad (8)$$

where  $e'$  represents the semantic encoder parameterized by  $f_{enc-S}$ .

**3.2.3 Session Vector.** By encoding both the temporal context and semantic information inside the action sequence, we eventually deliver a session vector for session  $s^i$  as

$$\mathbf{h}^i = g(\mathbf{h}_T, \mathbf{h}'_T), \quad (9)$$

where  $g(\cdot, \cdot)$  represents a common fusion function such as summation or concatenation. We refer to  $\mathbf{h}^i$  as the *session vector* to session  $s^i$ ,  $1 \leq i \leq N$ . Our session vector encapsulates the long-term sequential pattern from user session with the assistance of tutorial explanation, and therefore it gains the ability to remember personalized usage habit. In the next, we show how to predict next user actions upon the last session vector.

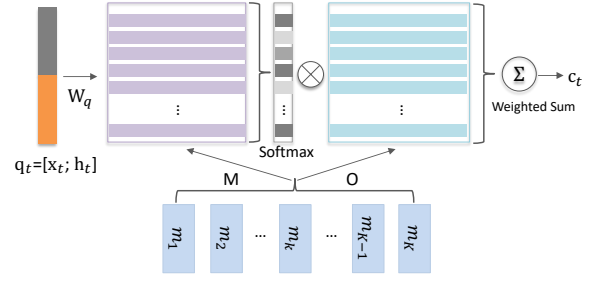
**3.2.4 Log Action Decoder.** Similar to some RNN based language models [7, 30], we develop our log action decoder by maximizing the following ordered conditional probabilities:

$$p(s^{i+1}) = \prod_{t=1}^T p(a_t^{i+1} | a_{t-1}^{i+1}, \mathbf{h}_{t-1}^{i+1}, \mathbf{h}^i), \quad (10)$$

which means the current action  $a_t^{i+1}$  in session  $s^{i+1}$  is conditioning on the previous action  $a_{t-1}^{i+1}$ , previous hidden state  $\mathbf{h}_{t-1}^{i+1}$ , as well as the last session vector  $\mathbf{h}^i$ . Specifically, the hidden state in decoding process is obtained by

$$\mathbf{h}_t^{i+1} = f_{dec}(\mathbf{x}_t^{i+1}, \mathbf{h}_{t-1}^{i+1}, \theta_{dec}), \quad (11)$$

where  $f_{dec}$  is parameterized by GRU and initialized with  $\mathbf{h}^i$ , *i.e.*,  $\mathbf{h}_0^{i+1} = \mathbf{h}^i$ , and  $\theta_{dec}$  represents all the parameters in the decoding function. The conditional probability is obtained through adding a



**Figure 3: Illustration for our recurrent semantics memory unit (RSMU), where each memory slot  $m_k$  is addressed and read by M and O, respectively.**

projection layer  $\mathbf{W}_a \in \mathbb{R}^{d_h \times |V_a|}$  on the top of  $f_{dec}$  followed by a softmax activation function, which is formulated as

$$\hat{\mathbf{y}}_t^{i+1} = \text{softmax}(\mathbf{W}_a^T \mathbf{h}_t^{i+1}), \quad (12)$$

where  $\hat{\mathbf{y}}_t^{i+1} \in \mathbb{R}^{|V_w|}$  gives the probability distribution over  $V_w$ .

### 3.3 Recurrent Semantics Memory Unit (RSMU)

The proposed recurrent semantics memory unit (RSMU) inherits from memory networks [26, 29, 34], and generally has two steps: 1) memory addressing and 2) memory reading. Recall that, our memory unit  $M$  consists of a set of sentences collected from human annotations for log sequence in the software tutorials. Though one action is corresponding to a sentence group, it should attend to different sentences (*i.e.*, memory slots) upon the action context, which is matched to different hidden states ( $[\mathbf{h}_1, \dots, \mathbf{h}_T]$ ) in the temporal encoding process. Thus, we recurrently address and read the memory unit as follows.

**3.3.1 Recurrent Memory Addressing.** Fig. 3 shows our RSMU block, which is inserted between the temporal encoder and semantic encoder (see Fig. 2). For each time step, we query the memory unit with a semantics attention mechanism by

$$\mathbf{q}_t = [\mathbf{x}_t; \mathbf{h}_t], \mathbf{e}_t = \mathbf{W}_q^T \mathbf{q}_t, \quad (13)$$

$$\alpha_k = \text{softmax}(\mathbf{e}_t^T \mathbf{m}_k) = \frac{\exp(\mathbf{e}_t^T \mathbf{m}_k)}{\sum_{k=1}^K \exp(\mathbf{e}_t^T \mathbf{m}_k)}, \quad (14)$$

where  $\mathbf{q}_t$  is the query vector,  $\mathbf{W}_q \in \mathbb{R}^{(d_a+d_h) \times d_m}$  projects  $\mathbf{q}_t$  into the memory embedding space as  $\mathbf{e}_t$ ,  $\mathbf{m}_k$  represents the memory embedding given by Eq. (2),  $1 \leq k \leq K$ , and  $\alpha_k$  works as the attention weights. Note that, we add the action embedding  $\mathbf{x}_t$  into the query vector, as it helps build a connection between different software actions and words; on the other hand,  $\mathbf{h}_t$  differs different sentences upon temporal context.

**3.3.2 Recurrent Memory Reading.** After obtaining the attentions weights  $\alpha_k$  for all the memory slots, we can write the semantic output for RSMU at each time step by

$$\mathbf{c}_t = \mathbf{W}_o^T \sum_{k=1}^K \alpha_k \mathbf{o}_k, \quad (15)$$

where  $\mathbf{c}_t$  in essence represents a semantic context vector, and  $\mathbf{W}_o \in \mathbb{R}^{d_m \times d_h}$  is used to align the dimension of memory embeddings and hidden states. It is worth noting that, though different actions  $a \in V_a$

**Table 1: Statistics Summary of User-Log Dataset**

dataset	# users ( $u$ )	# sessions ( $s$ )	avg. # session/ $u$	avg. # action/ $s$
User-Log-30	519,889	14,354,256	28	29
User-Log-100	454,881	5,018,158	11	82

have their own memory units, all the RSMUs share with the same parameter, which is denoted as  $\theta_{mem} = \{\mathbf{M}, \mathbf{O}, \mathbf{W}_q, \mathbf{W}_o\}$ .

### 3.4 Final Objective

Recall Section 2, we have collected two datasets to train our proposed Log2Intent model, which are (1) the user-log dataset containing plenty of sessions from user set  $\{u\}$ ; and (2) the tutorial dataset including human annotations for action sequence in all the tutorials. The user log data provide lots of sessions pairs  $(s^i, s^{i+1})$  to model the temporal patterns for each user, while tutorial sequences could supervise RSMU to attend to the corresponding memory slot of each action with different temporal context. Hence, we formulate two objectives as follows.

**3.4.1 Sequence Loss.** Given  $(s^i, s^{i+1})$  from user  $u$ , we learn the session vector  $\mathbf{h}^i$  for  $s^i$  with Eq. (9) and then predicts the following actions in  $s^{i+1}$  conditioning on  $\mathbf{h}^i$  with  $f_{dec}$ . Let  $\mathbf{y}_t^{i+1} \in \mathbb{R}^{|V_w|}$  be the one-hot vector of  $a_t^{i+1}$ , then we define the sequence loss for each user  $u$  as

$$\mathcal{L}_u = - \sum_{s^{i+1} \in u} \sum_t \mathbf{y}_t^{i+1} \cdot \log \hat{\mathbf{y}}_t^{i+1}, 1 \leq i \leq N, \quad (16)$$

where  $\cdot$  denotes element-wise product and  $\hat{\mathbf{y}}_t^{i+1}$  is given by Eq. (12).

**3.4.2 Attention Loss.** We treat our tutorial dataset as a “special user”  $u'$  and focus on modeling its semantic information. Given  $\forall s \in u'$ , we have the ground truth sentence annotation for  $\forall a_t \in s$ , which means we have memory labels  $\mathbf{y}'_t \in \mathbb{R}^K$  at each time step. Thus, an attention loss is designed to explicitly guide the RSMU training by

$$\mathcal{L}_{u'} = - \sum_{s \in u'} \sum_t \mathbf{y}'_t \cdot \log \hat{\mathbf{y}}'_t, \quad (17)$$

where  $\mathbf{y}'_t$  is a one-hot vector to index the correct memory slot in the corresponding memory unit  $M$  of  $a_t$ , and  $\hat{\mathbf{y}}'_t$  is directly given by the attention weights in Eq. (14), i.e.,  $\hat{\mathbf{y}}'_t = [\alpha_1, \dots, \alpha_K]^T$ .

By jointly considering Eq. (16) and Eq. (17), we eventually train our Log2Intent model by

$$\mathcal{L} = \sum_u \mathcal{L}_u + \lambda \mathcal{L}_{u'}, \quad (18)$$

where we sum over all the users’ sequence loss and concurrently optimize the attention loss, and  $\lambda > 0$  is used to balance the training frequency between these two parts. To handle large-scale user log data, we adopt stochastic gradient descent (SGD) optimization with mini-batch strategy to train our network. Specifically, we update  $\{\theta_{enc-T}, \theta_{enc-S}, \theta_{dec}, \theta_{mem}\}$  by minimizing  $\mathcal{L}_u$ ; while only backpropate the gradient of  $\mathcal{L}_{u'}$  w.r.t  $\{\theta_{enc-T}, \theta_{enc-S}, \theta_{mem}\}$ .

## 4 EXPERIMENT

### 4.1 Experimental Setting

**4.1.1 Datasets.** To train our model in a *sequence-to-sequence* fashion, we pre-process the User-Log dataset (in Section 2) as session

pairs. Specifically, we allow sequences with variable lengths not exceeding the maximum time step  $T$ . User sessions with larger size are divided into several consecutive sub-sessions by using non-overlapped sliding window of size  $T$ . We remove the users who only have an individual session of a smaller size than  $T$ . Hence, we generate datasets with different users upon the maximum time step. To explore the temporal impact of different lengths, we set  $T = 30$  and  $T = 100$  to obtain two datasets as summarized in Table 1, and termed as User-Log-30 and User-Log-100, respectively. We randomly held out 10% users from these two datasets for evaluation.

On the other side, we collect 49,763 action-sentence pairs from 1,781 tutorials from our Tutorial dataset. We pre-process these sentences by filtering out low frequency words and reduce the vocabulary to 3676 unique words. All these sentences are linked to 426 actions in our User-Log dataset, resulting each action has an average number of 116 sentences. To fully utilize the tutorial information, we slide over each tutorial by a window of size 10 with one step. By this means, we obtain 33,090 consecutive action sequences with ground-truth annotation sentences. We randomly select 5% tutorial sequences as a test set.

**4.1.2 Validation Criteria.** User intent is a general concept that covers a wide range of user choices. In the experiment, we employ *Hits@K* to evaluate the annotation retrieval and action prediction task; while employ *Recall@K* for the user interest detection task. In details, we give one *hit* if the desired annotation/action is in the top  $K$  predictions. Following [16, 33], the *Hits@K* is given by  $Hits@K = \frac{\#hit@K}{|\mathcal{D}^{test}|}$ , where  $\mathcal{D}^{test}$  represent the testing set. Note that, *Hits@K* is equivalent to top-K accuracy, since we have one exact annotation/action for each time step or the given sequence.

On the other side, *Recall@K* is computed by

$$Recall@K = \frac{|\text{GT} \cap \text{top } K \text{ predicted interests}|}{|\text{GT}|}, \quad (19)$$

where GT represents the set of ground-truth labels, as each user could have multiple interests. We average *Recall@K* for all the users in  $\mathcal{D}^{test}$ . We range  $K$  from 1 to 5 in the experiment.

**4.1.3 Implementation Details.** We implement our Log2Intent model with the Tensorflow framework, and employ the Adam optimizer [18] with an initial learning rate of 0.001. In our model, we have  $|V_a| = 940$  and  $|V_w| = 3676$ . We set  $d_a, d_m = 100$  as the embedding size for action embedding  $\mathbf{A}$  and memory embedding matrices  $\mathbf{M}, \mathbf{O}$ , respectively. We learn  $\mathbf{A}$  by training word2vec [25] model on the entire User-Log data, and fix it during our training procedure to avoid overfitting.  $\mathbf{M}$  and  $\mathbf{O}$  are initialized by Glove [28] word embeddings and further fine-tuned in training. We employ  $d_h = 64$  hidden units for all the GRUs in our encoding pathway, while the dimension of decoder GRU is upon the fusion function  $g(\cdot, \cdot)$  in Eq. (9): 64 for summation; 128 for concatenation. In the experiment, we employ the concatenation function without specification. For our memory unit, we set  $M = 50$  with  $L = 30$ , which means each action is linked to a set of 50 sentences and each sentence has a maximum length of 30. During training, we randomly select  $M = 50$  memory slots for actions that have more than 50 annotations, and pad empty memories for the actions have fewer annotations. We train our network with the batch size of 128. For simplicity, we alternately train our network by  $\mathcal{L}_u$  with batch from User-Log and  $\mathcal{L}_{u'}$  with batch

**Table 2: Log annotation retrieval on the tutorial test set.**

Methods	Hits@1	Hits@2	Hits@3	Hits@4	Hits@5
MemNet-Act	0.0436	0.0777	0.1103	0.1417	0.1714
MemNet-SeqAvg	0.2505	0.3636	0.4401	0.4995	0.5483
MemNet-u2v	0.1738	0.2660	0.3289	0.3812	0.4249
Log2Intent	<b>0.4697</b>	<b>0.6048</b>	<b>0.6755</b>	<b>0.7185</b>	<b>0.7512</b>

from tutorial training sequences. Layer normalization [3] is used to accelerate the training procedure. Our model is usually trained for four days on a GPU machine with NVIDIA Titan X cards.

## 4.2 Log Annotation Retrieval

Interpreting log history in a human readable way is crucial for understanding user behavior. However, it is quite challenging and time consuming, even for the professional software designer, to annotate the wild and long log history. In this paper, we utilize the Tutorial dataset as an auxiliary knowledge source to explain the action sequences. Specifically, we formulate log annotations as memory slots for each action, and expect to attend to meaningful sentences upon the temporal context among action sequence.

To validate the effectiveness of our semantics attention mechanism, we formulate log interpretation as a “annotation retrieval” task on the test set of our tutorial dataset. Specifically, given a tutorial action sequence  $[a_1, \dots, a_T] \in \mathcal{U}$ , each of which has its own memory block  $M$  ( $K$  annotation sentences) and the ground-truth annotation index  $y'_t \in \mathbb{R}^K$ , our goal is to retrieve the correct annotation from  $M$  queried by  $a_t$ , i.e., to predict  $y'_t$  at each time step (see Section 3.4.2 for more details). It is worth noting that, as the same software action has different annotations according to sequential context, this task works as a sanity check for our model, which should memorize the relationship between temporal patterns and annotation sentences.

**Baselines.** We implement three baseline methods upon the same memory network architecture (i.e.,  $\theta_{mem}$ ) in Section 3.3 with different query methods as follows.

- **MemNet-Act** ( $q_t = x_t$ ) uses an individual action as query without considering the temporal context in a log sequence.
- **MemNet-SeqAvg** ( $q_t = [x_t; \frac{1}{T} \sum_{i=1}^T x_i]$ ) conditions the average action embeddings in a log sequence on the query at each time step.
- **MemNet-u2v** ( $q_t = [x_t; f_{u2v}(s)], x_t \in s$ ) utilizes the inference model (i.e.,  $f_{u2v}$ ) given by util2vec (u2v) [38] to capture the temporal information within log sequence.

We train these three methods with the attention loss given by Eq. (17) until the model converges on the Tutorial dataset. The util2vec [38] model is pre-trained on the entire User-Log dataset (as training u2v does not require sequence pair). The proposed Log2Intent model is trained on the User-Log-30 and Tutorial datasets, and we formulate query as  $q_t = [x_t; h_t]$ , where  $h_t$  is given by our temporal encoder  $f_{enc-T}$ . For all the methods, the retrieval results are directly given by top  $K$  attention weights.

**Retrieval Results on Tutorials.** Table 2 shows the comparison results between our model and other methods on the Tutorial test set by *Hist@K*. As can be seen, MemNet-Act cannot attend to the correct annotation without using temporal context, since the

**Table 3: Log interpretation examples for log snippets in the User-Log dataset, where each block is one example with action sequence and its annotations given by our model.**

Action sequence	Interpretation by attended memory slots (annotations)
[open]	Step 33 Open the tourist photo.
[crop_tool]	You can fine-tune your crop with the arrow keys.
[layer_via_copy]	(Layer>Layer via Copy).
[new_curves_layer]	Go to Layer > New Adjustment > Curves and add Curves adjustment layer.
[new_levels_layer]	Add a Black and White adjustment layer, then add a Levels adjustment layer.
[zoom_in_tool]	Click repeatedly to zoom in closer.
Action sequence	Interpretation by attended memory slots (annotations)
[open]	Open your Tractor Image.
[rectangular_marquee]	With the Rectangular Marquee Tool (M) create a rectangular selection like the image below.
[rectangular_marquee_tool]	To do that, activate the Rectangular Marquee tool (M) and then select the rope of our swing.
[paste]	Press CTRL + V to paste in your texture.
[eraser]	Use the Eraser Tool (E) at any point to remove any harsh edges.
[brush_tool]	With a large, soft brush paint black on the bottom, and corners of the rectangle and white in the middle.
[eraser]	Use the Eraser Tool (E) to erase any blurred edges to bring back some sharpness around the face and body.
[brush_tool]	Choose a Soft Round brush at 40-80% Opacity.
[move]	Pick the Move Tool (V) and Control-click the part you want to color.

same action is corresponding to different memory slots (log annotations) upon different contexts. On the other side, MemNet-SeqAvg performs better than MemNet-u2v, which is mainly due to two reasons: 1) averaging actually considers longer-range temporal information than util2vec [38] (as u2v adopts a small sliding window); 2) util2vec [38] focuses on modeling global user representation from her/his entire log history, yet without explicitly considers the local temporal information. The proposed Log2Intent model leverages the RSMU block to recurrently query action’s memory at each time step (i.e.,  $q_t = [x_t; h_t]$ ), which effectively captures the temporal pattern in a log sequence. As shown in Table 2, our approach achieves a substantial improvement over baselines. This demonstrates the effectiveness of using attended sentence for the log interpretation task, conditioning on the temporal context.

**Interpret User-Log Data.** Table. 3 presents two interpretation results for the log snippets in the User-Log-30 dataset. Though we cannot validate these wild user logs, we have two interesting conclusions. (1) Despite the limitation of the topics covered by the Tutorial dataset, our model could explain the common actions with texts. For example, we can well describe the actions such as [crop\_tool], [zoom\_in\_tool] in the first example; and [paste], [move] in the second one by providing extra text information from attended memories. (2) The interpretation provided by our model is impacted by the action sequence. For instance, the [brush\_tool] of the second example in Fig. 3 is given by different attended sentences upon different contexts. All these attended sentences provide an alternatively view for the unstructured log history, which can help software designer to understand the user behavior.

## 4.3 User Interest Detection






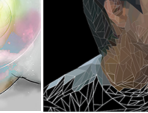

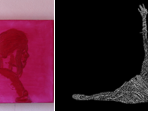

In this evaluation task, we aim to infer user’s interest only upon the log history. Our intuition is that users who share similar interest (or say occupation, self-disclosed tag, etc) may have similar temporal pattern in their software log history, as they usually work with similar contents and jobs. Thus, this requires our model could obtain a compact user representation. Given a user  $u$  with log history  $[s_1, \dots, s_T]$ , we obtain  $h_u$  by averaging all the session vectors




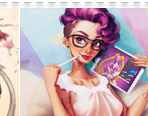
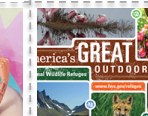


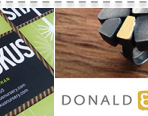



**Table 4: User interest detection evaluation in terms of Recall@K (%).**

	Methods	Recall@1	Recall@2	Recall@3	Recall@4	Recall@5
Detection Performance	Popular tags	15.41±0.34	24.35±0.39	33.00±0.41	38.65±0.41	43.50±0.46
	BoA + tf-idf (baseline)	17.15±0.57	26.78±0.59	33.93±0.61	39.62±0.69	44.55±0.78
	util2vec [38]	18.22±0.52	28.20±0.62	35.54±0.62	41.46±0.71	46.45±0.70
	Log2Intent	<b>19.00±0.51<sup>†</sup></b>	<b>29.25±0.58<sup>†</sup></b>	<b>36.62±0.60<sup>†</sup></b>	<b>42.68±0.56<sup>†</sup></b>	<b>47.87±0.60<sup>†</sup></b>
% of improvements	util2vec [38] over baseline	6.20%	5.32%	4.77%	4.65%	4.27%
	Log2Intent over baseline	10.80%	9.24%	7.93%	7.73%	7.45%

<sup>†</sup>: p-value < 0.01 under statistical significance test.

											
Method	Top1	Top2	Top3	Top4	Top5	Method	Top1	Top2	Top3	Top4	Top5
Pop tags	Graphic Design	Illustration	Photography	Branding	Art Direction	Pop tags	Graphic Design	Illustration	Photography	Branding	Art Direction
Log2Intent	Photography	Digital Photography	Fine Arts	Graphic Design	Performing Arts	Log2Intent	Graphic Design	Illustration	Fine Arts	Painting	Drawing
User interests (Ground truth)	Fine Arts, Photography, Digital Photography					User interests (Ground truth)	Illustration, Painting				

											
Method	Top1	Top2	Top3	Top4	Top5	Method	Top1	Top2	Top3	Top4	Top5
Pop tags	Graphic Design	Illustration	Photography	Branding	Art Direction	Pop tags	Graphic Design	Illustration	Photography	Branding	Art Direction
Log2Intent	Illustration	Digital Art	Graphic Design	Drawing	Character Design	Log2Intent	Graphic Design	Print Design	Advertising	Art Direction	Branding
User interests (Ground truth)	Illustration, Digital Art, Character Design					User interests (Ground truth)	Print Design, Graphic Design, Branding				

**Figure 4: Four user interest detection examples. For each user, we show her uploaded project images, popular user tags, user interest detection results by Log2Intent and her self-disclosed interests. We highlight the correct detections and the highly related ones with orange and green color, respectively.**

obtained by Eq. (9), i.e., we have  $\mathbf{h}_u = \frac{1}{|u|} \sum_{s^i \in u} \mathbf{h}^i$ . We employ our user representation to explore the possible user interests, which is formulated as a multi-label classification problem on the Behance dataset (see details below). To purely validate the effectiveness of  $\mathbf{h}_u$ , we only train a linear classifier as  $\mathbf{y}_u = \mathbf{W}_u \mathbf{h}_u$ , where  $\mathbf{W}_u$  directly projects user representation as a multi-label vector  $\mathbf{y}_u$ . We train  $\mathbf{W}_u$  by the sigmoid cross-entropy loss with limited-memory BFGS [22]. Moreover,  $\mathbf{h}_u$  is obtained by our Log2Intent model trained on the User-Log-100 dataset with the summation function.

**Behance Dataset.** We use the user data collected from Behance<sup>1</sup>, which is a social platform maintained by Adobe creative cloud. It allows users to upload their projects upon self-disclosed interests, such as *Photograph*, *Web Design*, *Fashion* etc. We summarize the majority user interests as 67 tags as following [38], and collect 9,439 users who are also the Photoshop users in our User-Log dataset. Hence, for each user in the Behance dataset, we have its user log record in February 2018, as well as a multi-hot label vector  $\mathbf{y}_u \in \mathbb{R}^{67}$ . We use a random training/testing split of size 7439/2000.

**Baselines.** Three strong baselines as used as follows. (1) **Popular tags.** We employ the 5 most popular user interest labels as the prediction results, i.e., *Graphic Design*, *Illustration*, *Photography*, *Branding* and *Art Direction*. (2) **Bag-of-Actions (BoA) + tf-idf.**

BoA computes the times of each action performed by the user. We adopt tf-idf to lower the weights of actions with high session frequency, such as [open] and [save\_as]. (3) **util2vec (u2v)** [38]. u2v is the state-of-the-art user representation learning method for log-trace data. It focuses on capturing the short-term temporal pattern exiting in the user’s log history. We train a multi-label classifier with BoA and u2v user representations, respectively, under the same setting to our approach.

**Detection Results.** Table 4 summarizes the user interest detection results of our approach and three compared methods. We do random split on the Behance dataset 20 times, and report the average Recall@K values with standard deviation. We also do a significance test between our result and util2vec [38]. The p-value for each Recall@K is less than 0.01, indicating the differences are statistically significant. Moreover, we also compare the relative improvement between Log2Intent and util2vec [38] over the baseline of BoA+tf-idf. As can be seen, our approach consistently outperforms util2vec from Recall@1 to Recall@5. This demonstrates the effectiveness of our Log2Intent model in learning user representations. It is worth noting that, though we simply average all the session vectors among user’s log history, we still have a better performance than util2vec [38], which is specifically designed for learning a global user representation based on the short-term temporal context from log history. This fully verifies our assumption

<sup>1</sup><https://www.behance.net/>

**Table 5: Next action prediction on User-Log-30 and User-Log-100 datasets by Hits@K.**

Methods	sequence 30 $\rightarrow$ 1 action					sequence 100 $\rightarrow$ 1 action				
	Hits@1	Hits@2	Hits@3	Hits@4	Hits@5	Hits@1	Hits@2	Hits@3	Hits@4	Hits@5
Bag-of-Actions	0.3104	0.4798	0.5644	0.6206	0.6623	0.3814	0.5356	0.6029	0.6498	0.6841
Average Action Embeddings	0.2989	0.4653	0.5513	0.6078	0.6494	0.3427	0.4898	0.5625	0.6188	0.6606
util2vec [38]	0.2001	0.3252	0.3938	0.4447	0.4835	0.2949	0.4241	0.4996	0.5518	0.5919
Log2Intent w/o RSMU	0.5421	0.6735	0.7313	0.7666	0.7935	0.5837	0.7184	0.7662	0.7981	0.8209
Log2Intent	<b>0.5548</b>	<b>0.6906</b>	<b>0.7495</b>	<b>0.7864</b>	<b>0.8124</b>	<b>0.5992</b>	<b>0.7377</b>	<b>0.7885</b>	<b>0.8195</b>	<b>0.8412</b>

that users’ long-term temporal pattern is more representative and thus more effective for the personalized user modeling task.

Some visualization results of user interest detection are shown in Fig. 4, where several important observations could be made. (1) With only around 7k training samples, our user representation is still able to explore diverse user interests. As can be seen, we successfully detect many less popular interest labels, such as *fine arts*, *character design* and *painting*. This justifies the ability of our approach for mining personalized usage habits. (2) Our approach can recommend relevant interests to user, as the examples highlighting with green color in Fig. 4, which shows the great potential of applying our approach to cross-platform recommendation.

#### 4.4 User Action Prediction

Predicting user’s next action is another related task to “leak” user intent. In this section, we train our model on the User-Log-30 and User-Log-100 datasets, respectively, and directly decode the following immediate action conditioning on the session vector obtained from the last session.

**Baselines.** Four baseline methods are employed in this evaluation. (1) **Bag-of-Actions (BoA)**. BoA counts the action frequency within each session, which is a standard user modeling approach for service providers [21]. (2) **Average Action Embeddings**. We average all the action embeddings in one session as the feature vector. (3) **util2vec (u2v)** [38]. We employ the pre-trained util2vec model on the entire User-Log dataset to infer a feature representation for each given session. We train a one-layer network by softmax cross-entropy loss with BoA, average action embeddings and u2v features, respectively. (4) **Log2Intent w/o RSMU**. Log2Intent w/o RSMU is an implementation of the proposed model without using RSMU. We compare with Log2Intent w/o RSMU to explore the impact of incorporating semantics information into the log encoding.

**Prediction Results.** Table 5 shows the action prediction performance of our approach and baselines by Hits@K. As can be seen, our Log2Intent model significantly improves the performance over BoA, average action embeddings and util2vec [38], which again demonstrates the effectiveness of leveraging long-term temporal information for the user modeling task. Moreover, our approach slightly outperforms Log2Intent w/o RSMU. This justifies the intuition behind our prediction task, *i.e.*, the semantic memories could also help to infer user’s following actions to some extent.

## 5 RELATED WORK

**Software User Modeling.** User modeling is an important task for personalized recommendation and is widely used by online social and E-commerce platforms. The goal of such modeling is to

summarize user preferences, habits and profiles to improve user experience, which however, could be roughly summarized as to explore user’s intent. Lots of research efforts have been made for this task by considering user data such as rating [5], contents [2, 17, 32], and reviews [39]. However, log-trace data, which contain rich user information, is still under explored.

Some previous works [1, 10, 21, 24] working with software actions are mainly designed to build a command recommendation system to assist users to complete complex task and learn software usage, yet neglect to learn a generic user representation. The most related method to this paper is one recent work termed as util2vec [38], which learns user representation by utilizing short-term temporal context within user’s log history in a doc2vec [20] framework. Different from util2vec [38], we explicitly leverage the long-term sequential information from user logs, and provide an efficient way to interpret the user behavior.

**Sequential Modeling.** Recurrent neural network (RNN) has been widely used for modeling sequential data in a wide range of applications, such as neural machine translation [4, 30] and speech recognition [12]. Among RNNs, LSTM [15] and GRU [8] are two important structures, and temporal attention mechanism [4, 23] is usually adopted by sequence to sequence model. Recently, recurrent sequential recommendation has attracted a lot of attentions, such as session-based recommendation [14] and recurrent recommender networks [35], which mainly adopt a RNN encoder followed by a ranking loss and are specifically designed for user recommendation. Different from these methods, our Log2Intent model serves with more general and broader purposes. Specifically, we develop our model by an encoder-decoder framework with semantics attention mechanism, and target to mine user’s intent from her log history.

**Memory Network.** The neural memory network (memNet) proposed by Weston *et al.* in [34] generally consists of two components: 1) a memory matrix to save information (*i.e.*, memory slots) and 2) a neural network controller to address/read/write memories. The memNet has shown better performance than traditional RNNs in several tasks, such as question answering [26, 29] and machine translation [13]. This is mainly due to its advantage over tackling long-term relationship and more storage units than a single hidden state. Following [34], several popular memory networks have been proposed as end-to-end memNet [29], Key-Value memNet [26], and dynamic memNet [36]. Our method is mainly based on the end-to-end memNet [29], which employs two memory embedding matrices to implement memory addressing and reading. In particular, we incorporate memory network into an RNN encoder, and recurrently query and write the memory unit through the network update. Similar to us, the recurrent memNet [31] also combines RNN network with a memory block. However, there are two main



differences between our Log2Intent model and [31]. First, Ref [31] utilizes memory network for language modeling, while Log2Intent adopts a sequence-to-sequence model for the user modeling task. Second, memory units in [31] are designed with last words to help prediction; however, we develop our memory with semantic sentences for interpretation. Most recently, the memory network has also drawn attentions from user recommendation tasks [6, 9, 33]. Compared with these methods, we not only implement a semantic recurrent memNet, but also provide a general framework that could cover a wide range of user modeling tasks.

## 6 CONCLUSION

An interpretable user modeling method termed as Log2Intent has been proposed in this paper, which mines user intent from sequential log-trace data. We implemented Log2Intent by incorporating auxiliary knowledge with memory network into a sequence to sequence model. Specifically, a recurrent semantics memory unit (RSMU) was developed to link the temporal encoder with the semantic one, which are jointly trained with a log action decoder network. We collected memory slots from software tutorials, and further supervised RSMU with annotated action sequence in the tutorial dataset. A session vector encapsulating long-term temporal patterns and attended memories was eventually delivered as user representations. Three evaluation tasks including log annotation retrieval, user interest detection and next action prediction were conducted on the Photoshop Tutorial and User-Log dataset. Experimental results compared with the state-of-the-art log-trace user modeling method demonstrated the effectiveness of the proposed Log2Intent model, as well as the great potentials of mining user intent from software log-trace history.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful comments. This work is supported by Adobe Research gift funding and SMILE Lab (<https://web.northeastern.edu/smilelab/>). The first author conducted part of this research at Adobe Research as a summer intern and is further supported by the SMILE lab at Northeastern University.

## REFERENCES

- [1] Eytan Adar, Mira Dontcheva, and Gierad Laput. 2014. CommandSpace: modeling the relationships between tasks, descriptions and features. In *UIST*. 167–176.
- [2] Deepak Agarwal, Bee-Chung Chen, Qi He, Zhenhao Hua, Guy Lebanon, Yiming Ma, Pannagadatta Shivaswamy, Hsiao-Ping Tseng, Jaewon Yang, and Liang Zhang. 2015. Personalizing LinkedIn Feed. In *SIGKDD*. 1651–1660.
- [3] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450 (2016).
- [4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
- [5] James Bennett, Stan Lanning, and Netflix Netflix. 2007. The Netflix Prize. In *In KDD Cup and Workshop in conjunction with KDD*.
- [6] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential Recommendation with User Memory Networks. In *WSDM*. 108–116.
- [7] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *EMNLP*. 1724–1734.
- [8] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS 2014 Workshop on Deep Learning, December 2014*.
- [9] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative Memory Network for Recommendation Systems. In *SIGIR*. 515–524.
- [10] Michael Ekstrand, Wei Li, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2011. Searching for Software Learning Resources Using Application Context. In *UIST*. 195–204.
- [11] Lin Gong and Hongning Wang. 2018. When Sentiment Analysis Meets Social Network: A Holistic User Behavior Modeling in Opinionated Data. In *SIGKDD*. ACM, 1455–1464.
- [12] Alex Graves, Abdel rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech Recognition with Deep Recurrent Neural Networks. *CoRR* abs/1303.5778 (2013).
- [13] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to Transduce with Unbounded Memory. In *NeurIPS*. 1828–1836.
- [14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
- [15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
- [16] Bo Hu and Martin Ester. 2013. Spatial Topic Modeling in Online Social Media for Location Recommendation. In *RecSys*. 25–32.
- [17] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *ICDM*. 263–272.
- [18] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).
- [19] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-Thought Vectors. In *NeurIPS*. 3294–3302.
- [20] Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *ICML*.
- [21] Wei Li, Justin Matejka, Tovi Grossman, Joseph A. Konstan, and George W. Fitzmaurice. 2011. Design and evaluation of a command recommendation system for software applications. *ACM Trans. Comput.-Hum. Interact.* 18, 2 (2011), 6:1–6:35.
- [22] Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *MATHEMATICAL PROGRAMMING* 45 (1989), 503–528.
- [23] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *EMNLP*. 1412–1421.
- [24] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. 2009. CommunityCommands: Command Recommendations for Software Applications. In *UIST*. 193–202.
- [25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).
- [26] Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-Value Memory Networks for Directly Reading Documents. In *EMNLP*. 1400–1409.
- [27] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. 2018. Perceive Your Users in Depth: Learning Universal User Representations from Multiple E-commerce Tasks. In *SIGKDD*. ACM, 596–605.
- [28] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*, Vol. 14. 1532–1543.
- [29] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. 2015. End-To-End Memory Networks. In *NeurIPS*. 2440–2448.
- [30] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *NeurIPS*. 3104–3112.
- [31] Ke M. Tran, Arianna Bisazza, and Christof Monz. 2016. Recurrent Memory Networks for Language Modeling. In *NAACL HLT*. 321–331.
- [32] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *NeurIPS*. 2643–2651.
- [33] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural Memory Streaming Recommender Networks with Adversarial Training. In *SIGKDD*. 2467–2475.
- [34] Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory Networks. *CoRR* abs/1410.3916 (2014).
- [35] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In *WSDM*. 495–503.
- [36] Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic Memory Networks for Visual and Textual Question Answering. In *ICML*. 2397–2406.
- [37] Longqi Yang, Chen Fang, Hailin Jin, Walter Chang, and Deborah Estrin. 2019. Creative Procedural-Knowledge Extraction From Web Design Tutorials. *CoRR* abs/1904.08587 (2019).
- [38] Longqi Yang, Chen Fang, Hailin Jin, Matthew D. Hoffman, and Deborah Estrin. 2017. Personalizing Software and Web Services by Integrating Unstructured Application Usage Traces. In *WWW*. 485–493.
- [39] Fuzheng Zhang, Nicholas Jing Yuan, Kai Zheng, Defu Lian, Xing Xie, and Yong Rui. 2016. Exploiting Dining Preference for Restaurant Recommendation. In *WWW*. 725–735.
- [40] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiuli Chen, and Jun Gao. 2018. ATRank: An Attention-Based User Behavior Modeling Framework for Recommendation. In *AAAI*.