Reinforcement Learning for User Intent Prediction in Customer Service Bots

Cen Chen[1], Chilin Fu[1], Xu Hu[2], Xiaolu Zhang[1], Jun Zhou[1], Xiaolong Li[1], Forrest Sheng Bao[3]
[1]Ant Financial Services Group; [2]Alibaba Group; [3]Iowa State University
{chencen.cc,chilin.fcl,yueyin.zxl,jun.zhoujun,xl.li}@antfin.com
huxu.hx@alibaba-inc.com;forrest.bao@gmail.com

ABSTRACT

A customer service bot is now a necessary component of an e-commerce platform. As a core module of the customer service bot, user intent prediction can help predict user questions before they ask. A typical solution is to find top candidate questions that a user will be interested in. Such solution ignores the inter-relationship between questions and often aims to maximize the immediate reward such as clicks, which may not be ideal in practice. Hence, we propose to view the problem as a sequential decision making process to better capture the long-term effects of each recommendation in the list. Intuitively, we formulate the problem as a Markov decision process and consider using reinforcement learning for the problem. With this approach, questions presented to users are both relevant and diverse. Experiments on offline real-world dataset and online system demonstrate the effectiveness of our proposed approach.

CCS CONCEPTS

• **Theory of computation → Reinforcement learning**; • **Computing methodologies → Ranking**;

KEYWORDS

User Intent Prediction, Reinforcement Learning

ACM Reference Format:
Cen Chen, Chilin Fu, Xu Hu, Xiaolu Zhang, Jun Zhou, Xiaolong Li, and Forrest Sheng Bao. 2019. Reinforcement Learning for User Intent Prediction in Customer Service Bots. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3331184.3331370

1 INTRODUCTION

Traditional customer service is manpower/resource-intensive and time-consuming. It is thus of great importance to build intelligent assistants that automatically answers questions faced or asked by a user. Recently, there is an increasing interest in building such an intelligent assistant using machine learning techniques in industry, which goes beyond customer service, such as Amazon Alexa, Microsoft Cortana, and AliMe [8].

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331370

As a core function of a customer service bot, user intent prediction aims to automatically predict questions that a user might want to ask, and present candidate questions to the user for his/her selection to alleviate user's cognitive burden. More specifically, such a user intent prediction task can be treated as a top-N recommendation task [3], where each predefined question corresponds to a unique intent class. Most of existing approaches [2, 5, 16] solve the problem as an instantaneous classification problem, predicting a list of items that a user will be most likely interested in, given the current state of data. Such approaches aim at maximizing the immediate reward, such as the click, and ignore the long-term effects of each recommendation in the list, which may not be ideal in practice [19]. A more desirable way is to model the problem as an N-step sequential decision-making process to better capture the inter-relationship among the recommended questions and dynamics of the users [14, 19].

In this paper, we formulate the user intent prediction as a Markov decision process (MDP) and leverage Reinforcement Learning (RL) to find the best recommendation strategies. At each step, our RL-based ranking model finds a suitable question to be added to the recommendation list. Hence, it takes N steps to perform top-N recommendation, i.e., the episode length is N. Meanwhile, the complex nature of top-N recommendation requires the RL module to be light-weight so as to update its strategy in an online fashion. We build a set of ranking models beforehand and use their ranking results as our ranking signals. The RL module adapts the weights of the ranking signals to perform top-N recommendation. Our proposed RL-based approach brings two folds of benefits. First, it optimizes a mixture of long-term accumulated rewards, such as click, diversity, and user satisfaction. Second, the strategy can be continuously updated and adjusted along with changing question popularity and user behavior patterns.

To the best of our knowledge, our study is the first to consider a deep reinforcement learning based method for user intent prediction. Experiments on offline real-world dataset and online system in Alipay demonstrate the effectiveness of our proposed approach.

2 RELATED WORK

User Intent Prediction. We treat the task of user intent prediction as a task to recommend suitable questions that users are interested in. This task is close to Click-Through Rate (CTR) prediction task, as the core problem is to evaluate the click or purchase probability of an item (a question) for a given user query (user historical behaviors). The task is a large-scale problem in the industry as usually user and item sizes are huge. Due to the large feature space and data size, a number of methods are developed to avoid feature engineering, e.g. FM [12]. With the popularity of deep neural networks

(DNNs), there is a growing number of studies working on using DNNs to learn high-degree feature interactions [2, 5]. Empirical studies show DNN-based methods have a good performance, thus we adopt DNN-based methods as our base model.

Reinforcement Learning (RL). In our study, we consider RL for top-N recommendation in user intent prediction. The concept of RL dates back to several decades ago [1, 15]. Reinforcement learning algorithms can be categorized into two types: value-based methods and policy-based methods. Value-based methods estimate the expected total return given a state. This type of method includes SARSA [13] and the Deep Q Network [11]. Policy-based methods try to find a policy directly instead of maintaining a value function, such as the REINFORCE algorithm [18]. Finally, it is also possible to combine the value-based and policy-based approaches for a hybrid solution, such as the actor-critic algorithm [7]. This approach employs a learned value estimator to provide a baseline for the policy network for variance reduction. We experiment with a hybrid method named DDPG in this study. To the best of our knowledge, our study is the first to use RL for user intent prediction.

RL has been applied to recommendation tasks in previous studies [4, 17]. These models do not directly optimize the combination of base ranking signals such as CTR scores, diversity etc. In our RL method, we model the actions as the weights for the base ranking signals and study the importance of different ranking signals at different positions in the top-N recommendation. In this way, our model is lightweight and can be updated in an online fashion.

3 PROBLEM FORMULATION

The problem can defined as a multi-class classification problem. Formally, given user intent labels $L = \{l_1, l_2, ..., l_i\}$, the goal is to predict the most likely intent label of this user, subject to his/her user profile and the trace of historical behaviors. Essentially, in the customer service setting, each intent label l_i corresponds to a predefined question q_i to be predicted. A user will be presented with a list of N mostly likely questions, i.e., top-N question recommendation [3].

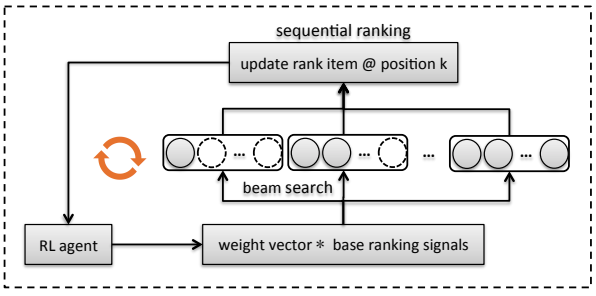

Figure 1: An overview of our RL-based method. Solid circles represent the selected items, while the dash ones are to be selected.

As illustrated in Fig. 1, our method consists of the ranking signal generation and the RL-based question ranking.

3.1 Base Ranking Signals

We focus on three types of ranking signals: Click-Through-Rate (CTR) score, question popularity, and question diversity. The first signal measures user interests, the second signal captures real-time question popularity, and the last signal gauges the diversity of recommended questions.

3.1.1 CTR score. Essentially, the task here is to evaluate the click probability of an item (i.e., a question) for a given user query (i.e., user historical behaviors), where we can build a CTR model to estimate question CTR [2, 5].

We consider using neural network based models for the task. We incorporate two types of features as input: user click sequence and user profile. For the former, we use CNN [6] to get a feature representation. For the latter, we use a fully-connected layer to learn a feature representation. The two types of feature representations are concatenated and fed into two fully-connected layers to generate final prediction scores.

3.1.2 Question popularity. Besides CTR scores, we consider question popularity as another ranking signal. The reason is that many questions are common to all users. For example, during a deal season, e.g., the Black Friday in the US or Double 11 in China, shoppers are overwhelmingly interested in how to redeem coupons. Specifically, we consider the real-time question popularity metrics, such as clicks and exposure, at different granularity, e.g., the recent 1-hour, 6-hour, one-day and etc.

3.1.3 Question diversity. The diversity of questions plays an important role in question ranking. Ideally, we hope the list of candidate questions to cover a wide range of question types. We define the diversity signal as the similarity of the current question to all previous questions.

3.2 RL for Question Ranking

We use RL to rank questions based on a richer set of ranking signals. The RL approach assigns weights to the ranking signals and using the weighted scores to rank questions at each step. In Fig. 2, we present an illustration of our RL-based method. Here we model the problem of top-N recommendation as an N-step Markov decision process, finding and adding a new question to the selected question list in each step.

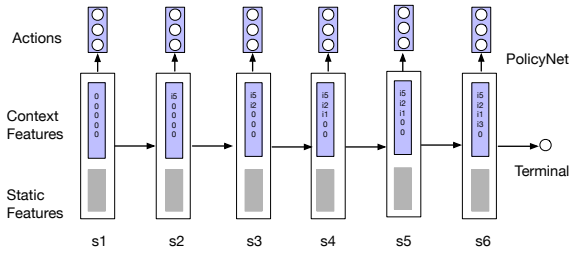

Figure 2: The Markov decision process for top-N recommendation ($N = 6$).

3.2.1 ***State.*** The state s_i of step i has static features such as user profile, and context or dynamic features as follows.

Static features: user features such as age, education, location etc.

Context features: item features such as the popularity of the selected questions; the step index (equal to or less than N), and the set of selected question indexes.

3.2.2 ***Action.*** An action at step i is $a_i = \{w_{i0}, w_{i1}, \cdots, w_{im}\}$, where each $w_{ij} \in [-1, 1]$ indicates the importance of the ranking signal f_{ij}. Here w_{ij} is generated by a learned policy function $\pi(\mathbf{S}_i)$. The policy network $\pi(\mathbf{S}_i)$ is approximated with two fully-connected layers. Formally, $\pi(\mathbf{S}_i)$ is defined as:

$$\pi(\mathbf{S}_i) = \tanh(\mathbf{W}_2\mathbf{H}_i + \mathbf{b}_2) \quad (1)$$

$$\mathbf{H}_i = \tanh(\mathbf{W}_1\mathbf{S}_i + \mathbf{b}_1), \quad (2)$$

where $\mathbf{W}_{(\cdot)}$ and $\mathbf{b}_{(\cdot)}$ are the weight matrix and bias vector respectively for a layer in the policy network, and $\mathbf{H}_{(\cdot)}$ is an intermediate hidden state. With the tanh activation function, the output action weights are bounded in the interval $[-1, 1]$.

Note that in this stage, our goal is to train an RL module that automatically combines these ranking signals for a better result. Hence the raw signal can be pre-trained in an offline fashion with a large dataset. While the RL module can be trained in online fashion to optimize the combination of these signals.

3.2.3 ***Question Ranking.*** Note that each question has static features that are consistent at different steps. For step i, let f_i^k be the set of question-level ranking signal features correspond to question k, where f_{ij}^k is the j-th ranking signal for question k. The ranking score of question k at step i is thus defined as:

$$c_i^k = \sum_{j=0}^{m} f_{ij}^k w_{ij}. \quad (3)$$

The candidate questions are then ranked based on the scores $c_i^{(\cdot)}$ to produce an output at step i, using one of the following approaches:
Greedy search: At each step i, we find the question with the maximum score to output, i.e. $\arg\max_k c_i^k$.
Beam search: At each step i, we find top b ($b = 3$ in our model) questions with the top score to generate a candidate output. At the last step, the sequence of questions that lead to the top score will be used as the output. Unlike greedy search, this approach keeps a subset at each step to avoid choosing a sub-optimal question.

3.2.4 ***Reward.*** With the above output question sequence, we can compute the reward for the RL method. For each step $i \le N$, the reward is set as α^p if the user clicks the p-th and 0 otherwise. Here α is a penalty factor ($\alpha = 0.9$ in our study). Note that we also incorporate user satisfaction rate as an additional reward r', where we set $r' = 1.0$ if the user clicks satisfaction in our system, otherwise $r' = 0$. In all, we compute the reward r_i for step i as:

$$r_i = \begin{cases} \mathbb{1}^{p==i}\alpha^p & \text{if i is not terminal,} \\ \mathbb{1}^{p==i}\alpha^p + r' & \text{otherwise,} \end{cases}$$

where $\mathbb{1}^s$ is an indicator function that returns 1 when the statement s is true and 0 otherwise.

3.2.5 ***Optimization.*** Our model is based on the actor-critic framework since it can help to reduce the variance so that the learning is more stable [7]. As our actions are deterministic, we adapt Deep Deterministic Policy Gradients (DDPG) [9] for our model.

In DDPG, we have target networks for both the actor and the critic, parameterized as Θ_{tgt} and Ω_{tgt} respectively. For each episode, the actor chooses actions according to: $a_i \leftarrow \pi(S_i) + \epsilon$, where ϵ is a noise process which can be an Ornstein-Uhlenbech process to generate temporally correlated exploration. With the generated tuples (s_i, a_i, s_{i+1}, r_i), we can then update the network parameters.

For each tuple, compute the target as:

$$y(r_i, s_{i+1}) = \begin{cases} r_i & \text{if } i+1 \text{ is terminal,} \\ r_i + \gamma Q_{\Omega_{tgt}}(s_{i+1}, \pi_{\Theta_{tgt}}(s_{i+1})) & \text{otherwise,} \end{cases}$$

where γ is a discount factor, empirically set as 0.95 in our study.

Let B be a batch of tuples (s_i, a_i, s_{i+1}, r_i), we update the value function as:

$$\Omega \leftarrow \Omega - \nabla_\Omega \frac{1}{|B|} \sum_{(s_i, a_i, s_{i+1}, r_i) \in B} \left(Q_\Omega(s_i, a_i) - y(r_i, s_{i+1})\right)^2. \quad (4)$$

We update the critic network for each batch B as:

$$\Theta \leftarrow \Theta + \frac{1}{|B|} \sum_{s_i \in B} \nabla_a Q_\Omega(s, a)|_{s=s_i, a=\pi(s_i)} \nabla_\Theta \pi_\Theta(s_i). \quad (5)$$

The target networks can be updated as follows:

$$\begin{aligned} \Omega_{tgt} &\leftarrow (1-\rho)\Omega_{tgt} + \rho\Omega, \\ \Theta_{tgt} &\leftarrow (1-\rho)\Theta_{tgt} + \rho\Theta, \end{aligned} \quad (6)$$

where ρ is a hyperparameter between 0 and 1 (usually choose 0.001).

4 EXPERIMENTS

4.1 Data and settings

The experiment data is sampled from Alipay's customer service bot. Each request instance has three types of features as elaborated in § 3.1. In the dataset, each label represents a distinct question that may be asked by or presented to the user and in total there are 200 unique question classes.

To evaluate the quality of the top-N recommendation, we measured the Hit-Rate (HR), Mean Average Precision (MAP), and Average Click Position (AvgClickPos) of the recommended top-N list. Hit-rate is calculated as $HR = \#hits/M$, where M is the total number of user request instance and an instance is considered as hit when question clicked is presented in the top-N list. MAP is defined as mean average precision on all the positions, while AvgClickPos is defined as the average position of all clicked questions in the list. A simulation environment is constructed to evaluate the feedback/reward with beam search. In the offline experiment, we trained the model on a one-week data and tested the model performance on a subsequent day, while for the online evaluation, we conducted a two-week A/B test online.

4.2 Offline Evaluation

To demonstrate the effectiveness of the base CTR signals learned by the DNN presented in § 3.1.1, we compare it against those by the Logistic Regression (LR) in Table 1. The DNN-based method outperforms the vanilla LR-based method by up to 6% relatively on all evaluation metrics. This is because DNNs, which usually perform better than vanilla linear models, are able to capture high-level interactions between features.

Table 1: Base CTR model performance

	HR@top3	HR@top6	AvgClickPos	MAP
LR	70.97%	90.24%	2.6863	0.5649
DNN	**72.65%**	**91.74%**	**2.5408**	**0.5993**

To examine the performance of RL-based method, we compare two variants of the proposed model, i.e., RL-greedy and RL-beam, against the following Learning-To-Rank (LTR) [10] baselines:

- **LTR-LR**: classical point-wise LTR approach that approximates a relevance rank using LR.
- **LTR-DNN**: similar to LTR-LR, where DNN is adopted as the ranking model.

Table 2: Offline Evaluation on model performance.

	HR@top3	HR@top6	AvgClickPos	MAP
LTR-LR	74.80%	89.84%	2.6646	0.5869
LTR-DNN	75.88%	92.60%	2.5395	0.6110
RL-greedy	75.30%	95.58%	2.6705	0.6246
RL-beam	**75.95%**	**95.69%**	**2.3533**	**0.6329**

We set N=6 for all the experiments, i.e., the top-6 recommendations[1]. Hence our RL-based approaches are optimized with an episode length of 6. The results are given in Table 2. First, LTR-DNN is better than LTR-LR because DNN has better expressive power than LR. Second, RL-beam is better than RL-greedy showing that a beam search is more beneficial than a greedy search in our task. Finally, RL-beam has better performance than all competing methods, improving the best baseline by up to 3.6% relatively, which demonstrates the effectiveness of our proposed method.

4.3 Online Evaluation

We deployed our model online in Alipay's service bot. We compare our RL-beam with a baseline model which is a variant of our model that does not consider RL reranking. Since our RL based method updates the model on a frequent basis, it can adjust quickly to sudden changes with respect to the question popularity and user behaviour patterns. A 14-day A/B testing was conducted on Feb 4-17, 2019. On average, our model brings a *relative improvement* of 12.29% in terms of CTR on the recommended list (top 6), which is considered to be significant for the platform.

4.4 Case Study

We perform a case study of the learned weights for the three types of ranking signals [2] in § 3.1. As shown in Table 3, the learned weights are generally different at different steps. Interestingly, we observe larger CTR weights at the early steps and smaller at later steps. This is intuitive as the top positions attract more user attention which will be better to present the questions with the highest click probability. And for diversity, its weight is relatively larger at later steps which shows it is important to promote diversity at later steps. In all, the learned weights are interpretable and provides insights on the importance of ranking signals at different steps.

[1]In our customer service bot, a page is best displayed with 6 questions.
[2]For popularity, we aggregate all the ranking features related to popularity

Table 3: A case study of the learned weights.

	Step 1	Step 2	Step 3	Step 5	Step 5	Step 6
CTR scores	0.122	0.049	0.103	0.034	0.041	-0.029
Diversity	0.342	0.281	0.305	0.384	0.389	0.399
Popularity	0.230	0.114	0.172	0.214	0.194	0.090

5 CONCLUSION

In this work, we propose to model the problem of user intent prediction as an N-step decision process to capture the inter-relationship between the recommended questions, so as to recommend questions that is relevant as well as diverse. We consider a reinforcement learning approach for the problem. Experiments on real-world datasets show our method generates better recommendations and improves the task of user intent prediction. We have deployed the method in a real-world industry bot and observed significant improvement.

REFERENCES

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* (2017).
[2] Heng-Tze Cheng and et al. 2016. Wide & deep learning for recommender systems. *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (2016).
[3] Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *TOIS* 22, 1 (2004), 143–177.
[4] Yue Feng, Jun Xu, Yanyan Lan, Jiafeng Guo, Wei Zeng, and Xueqi Cheng. 2018. From Greedy Selection to Exploratory Decision-Making: Diverse Ranking with Policy-Value Networks. In *SIGIR (SIGIR'18)*. 125–134.
[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *CoRR* abs/1703.04247 (2017).
[6] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 EMNLP*. 1746–1751.
[7] V. R. Konda and J. N. Tsitsiklis. 2003. On Actor-Critic Algorithms. *SIAM J. Control Optim.* (2003).
[8] F. Li, M. Qiu, H. Chen, X. Wang, X. Gao, J. Huang, J. Ren, Z. Zhao, W. Zhao, L. Wang, and G. Jin. 2017. AliMe Assist: An Intelligent Assistant for Creating an Innovative E-commerce Experience. In *CIKM '17*.
[9] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015).
[10] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.
[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* (2015).
[12] Steffen Rendle. 2010. Factorization Machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*. 995–1000.
[13] G. A. Rummery and M. Niranjan. 1994. *Online Q-Learning Using Connectionist Systems.* Technical Report. University of Cambridge.
[14] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *JMLR* 6 (2005), 1265–1295.
[15] R. S. Sutton and A. G. Barto. 1998. *Reinforcement Learning - An Introduction.* MIT Press.
[16] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proc. of the ADKDD'17*. 12:1–12:7.
[17] Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2017. Reinforcement Learning to Rank with Markov Decision Process. In *SIGIR17*. 945–948.
[18] R. J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* (1992).
[19] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. 2017. Deep reinforcement learning for list-wise recommendations. *arXiv preprint arXiv:1801.00209* (2017).