Making Rewards More Rewarding: Sequential Learnable Environments for Deep Reinforcement Learning-based Sponsored Ranking

Chen Wang[*]
Rutgers University
New Brunswick, United States
chen.wang.cs@rutgers.edu

Aidan Finn
Overstock.com
Midvale, United States
afinn@overstock.com

Nishan Subedi
Overstock.com
Midvale, United States
nsubedi@overstock.com

ABSTRACT

Reinforcement Learning (RL) methods have risen in popularity among general ranking systems. However, despite having properties suitable for sponsored ranking problems, Reinforcement Learning methods remain under explored in this area. A major reason behind this gap is the *dilemma of exploration*: random exploration is prohibitively expensive in sponsored search ranking, with the potential to cause significant revenue loss. To address this concern, we study properties of a *simulated environment* for Reinforcement Learning in sponsored ranking. We demonstrate that by augmenting a learnable simulated environment based on intuitive design principles, we can significantly improve RL performances and boost the explainability of the model. We test our method with a Deep Deterministic Policy Gradient agent, and experimental results show our learned simulated environment outperforms existing methods. Furthermore, since our method is agent agnostic, it paves the way to a wide range of Reinforcement Learning applications to the sponsored ranking problem.

CCS CONCEPTS

• **Information systems → Sponsored search advertising**; **Electronic commerce**; • **Computing methodologies → Reinforcement learning**.

KEYWORDS

reinforcement learning, sponsored search, ranking

ACM Reference Format:
Chen Wang, Aidan Finn, and Nishan Subedi. 2021. Making Rewards More Rewarding: Sequential Learnable Environments for Deep Reinforcement Learning-based Sponsored Ranking. In *AdKDD '21: Workshops of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, August 14-18, 2021, Singapore.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

[*]Most of the work was done when the author was at Overstock.com.

All authors contributed equally to the work.

1 INTRODUCTION

Reinforcement Learning (RL) is a class of machine learning methods that optimize their performance by exploring and receiving rewards from their environment [19]. In recent years, due to the expressive power of Deep Reinforcement Learning (DRL), RL has become widely adopted for various prediction and decision-making problems. In the field of ranking, RL has been extensively studied under various contexts like recommendation systems [12, 22], web search results [2], document filtering [21], among others. RL methods are especially effective when the ranking is dynamic and interactive [1, 7, 15] (i.e. the items and the user's demand evolve over time).

In sponsored search ranking, a marketplace platform is presented with a list of advertisements and the goal is to determine the order of the advertisements to be displayed. A common approach is to combine advertiser bid and expected Click-Through-Rate (CTR) in a second-price auction, where the winning advertisements are charged the minimum amount needed to keep their position. In general, higher ranking ads should have both a higher bid and a higher probability of being clicked. While the former metric is easy to examine, the latter one depends largely on the advertisement's relevance to the search and the user's preference. Furthermore, the participants in an auction are determined by the goal of the advertisers, which may vary with each campaign, and budget available for each item at the time the auction changes. Given this dynamic nature of sponsored search ranking, RL-based approaches are attractive.

There is existing work that utilizes novel ideas to make RL-based sponsored search ranking feasible. He et al. [4] designs a mechanism to compute the rank score as a sum of the scores for the platform, user and advertiser (see also [5]). From the generalized second price auction mechanism, one can use the score of the advertisement ranked below the current item to compute the reward. In this way, the 'environment' could provide rewards to *any* action, enabling exploration in the sponsored search ranking context.

He et al. [4] describes a ranking strategy formulation that captures the general goal of sponsored ranking systems. An actor-critic deep reinforcement learner runs on a simulated environment to allow exploration which is deployed in an online setting using an evolution strategy to update the parameters of the policy model. However, when we tried to implement the offline Deep Deterministic Policy Gradient (DDPG) Learning algorithm based on our internal sponsored ranking logs, we found the performance of the model to be unstable [1]. This led us to investigate ways to improve

[1]When evaluated on sponsored search ranking data from Overstock.com.

the environment model that can facilitate better effectiveness and consistency between the RL goal and improved ranking quality.

In this paper we show that the environment and ranking score model of [4] can be improved in two ways: 1. by increasing the expressive power of the model; and 2. by introducing rule-induced consistency in reward design. We introduce a new mechanism such that the ranking function is trained by a sequential learnable environment based on user engagement. Our contributions are generally applicable since the sequential learnable environment model is agent agnostic, and the changes introduced in our model require no domain specific parameters to be tuned.

We test our model on real-world sponsored search ranking data from Overstock.com. To draw comparisons with existing approaches and show the effectiveness of our design, we also test the performance of other models, including the original model proposed in [4] and the classical CTR-based model, on the same dataset and with the same type of the DDPG agent. Experimental results show that the agent with the environment proposed in this paper outperforms other methods and offers more consistent learning signals.

2 RELATED WORK

The idea of applying RL to ranking system has received considerable attention in the era of deep reinforcement learning [19]. RL has been successfully deployed on search engine optimization [7], recommendation systems [8], advertisement displaying [11], and advertisement retrieval and pre-ranking [20]. In particular, the problem [20] studied falls into the earlier stages of sponsored search ranking, and it built a sequential and learnable neural network to produce the results. Note that capturing the sequential information is often crucial in the application of advertisement click data (see [10]). Therefore, our approach extends the idea of [20]: we similarly build a sequential and learnable neural network as a generic part of the environment; the difference is that the neural network in our method is adopted to produce the *embedding* of the advertisements.

3 OUR METHOD

In standard RL applications, the goal is often to design and implement the agent, while the environment information is obtained via exploration. However, exploration becomes prohibitively expensive in sponsored search ranking. This makes the often-trivial environment model a major design challenge. To address this challenge we build our method based on the idea of the environment model in [4]. Our contribution is to provide a thorough treatment to the understanding of the environment model, and suggest improvements to the model based on our findings.

3.1 Review of the Simulated Environment Model

The central idea of the environment model in [4] is to use the inverse of the ranking function to generate the click prices and the rewards. The ranking function used in [4] follows:

$$\phi(ad, \boldsymbol{a}) = bid \cdot f_{a_1}(CTR) + a_2 \cdot f_{a_3}(CVR,\ CTR) + a_4 \cdot f_{a_5}(CVR,\ price), \tag{1}$$

where $\boldsymbol{a} = [a_1, \cdots, a_5]$ parameterize the output of the learning agent (*actions*), and $f_a(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ are monotonic non-linear functions. As standard in the sponsored search ranking literature, CTR and CVR stand for the Click Through Rate and the Conversion Rate. The values of a_i are non-negative and bounded by some positive number. By solving the inverse of Eq (1), one can compute the *click price*, which is proportional to the reward. With the second price auction mechanism, the prediction of the click price is

$$\begin{aligned} &\psi(ad, \boldsymbol{a}) = a_2 \cdot f_{a_3}(CVR,\ CTR) + a_4 \cdot f_{a_5}(CVR,\ price); \\ &\text{click-price}_{pred} = \frac{\phi(ad', \boldsymbol{a}) - \psi(ad, \boldsymbol{a})}{f_{a_1}(CTR)}. \end{aligned} \tag{2}$$

In the above equation, ad' stands for the advertisement that is ranked second to the clicked advertisement (*ad*), and the click price is the minimum value required to maintain position in the auction. A natural notion of reward can therefore be defined as

$$r = \text{click-price}_{pred} \cdot CTR. \tag{3}$$

Note that in the model [4], the reward of Eq (3) is 'smoothed' as $r = \left(\text{click-price}_{pred} \cdot CTR + \delta \cdot CTR\right)$, where δ is a hyper-parameter to be specified. We believe that although the parameter helps the performance, the choice of its value is not transferable to a general context and does not help in terms of understanding the environment model. We therefore omit the δ factor in this work.

3.2 Our Model: The Improved Simulated Environment

We now address the core problem posed and addressed by this paper: what makes a *good* simulated environment for Reinforcement Learning on sponsored search applications, and how can we *improve* the environment model? We present design decisions based on intuitive principles that we found [4] was unable to consistently address.

The proposed model has two major distinct components: the *sequential trainable embedding* for the environment and the *sample-adaptive click price calculation mechanism.* Specifically, the first component enables the environment to capture the sequential information between clicks, and the second component introduces the separation between positive and negative examples. Noticeably, although the designs can be viewed as 'revisions' of [4], we demonstrate that these changes lead to significant gains.

We begin with introducing the sequential trainable model. The idea of this environment is to two-fold: for the sequential part, we adopt a Recurrent Neural Network (RNN) structure (with Gated Recurrent Unit (GRU) cells); for the trainable part, the prediction of the click price is a function of the output of the RNN. Therefore, by setting a loss function between predicted and the real click price, one can train the parameters of the RNN by standard back-propagation. We note that the idea has been explored in [20] for sponsored search ranking applications. Moreover, we remark that previous work has shown that a learned environment is particularly well suited to ranking tasks since sequence models gain statistical power by aggregating across multiple individual engagements [6, 14]. An illustration of the model we adopted can be shown as Figure 1.

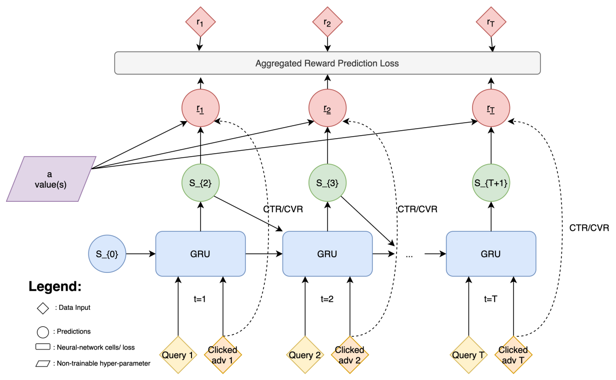

Figure 1: The framework of the trainable environment.

We now describe the details of the method. For a given session of consecutive clicks, at each step, the state transitions of the RNN are computed as

$$\boldsymbol{s}_t = \mathrm{GRU}(\boldsymbol{s}_{t-1}; \boldsymbol{q}_t; \boldsymbol{z}_t; \mathrm{CTR}; \mathrm{CVR}; bid). \tag{4}$$

Note that we slightly abuse the notation to use CTR, CVR and bid to denote the corresponding values in the t-th step. In Eq (4), $\boldsymbol{s}_{t-1}$ is the internal state of the GRU-RNN model from the previous step; $\boldsymbol{q}_t$ is the embeddings of the query; $\boldsymbol{z}_t$ is the feature embedding of the advertisement, and the last 3 terms are the meta-information. When calculating the reward with Eq (2), the RNN model provides the hidden state embedding of the t-th click as an additional feature (denoted as $\boldsymbol{s}_t$). The computation of the ranking function will therefore be updated as

$$\begin{aligned}\hat{\phi}(ad, \boldsymbol{a}) =& bid \cdot f_{a_1}(CTR, \boldsymbol{s}_t, \boldsymbol{z}_t) + a_2 \cdot f_{a_3}(CVR, CTR, \boldsymbol{s}_t, \boldsymbol{z}_t) \\ &+ a_4 \cdot f_{a_5}(CVR, price, \boldsymbol{s}_t, \boldsymbol{z}_t),\end{aligned} \tag{5}$$

which means, in addition to the meta information like the CTR and CVR, the features of each product, embedded as the hidden states of the RNN, are also taken into account to compute the predicted click price. Eq (5) inevitably increases the dimension of the input space to $\mathbb{R}^{>1}$, which makes it incompatible with the non-linear functions $f_a : \mathbb{R} \to \mathbb{R}$. This issue can be easily addressed by projecting the input to a scalar with a projection vector $\boldsymbol{w}$. Note that $\boldsymbol{w}$ can also be part of the learnable parameters by the standard back-propagation.

We then compute the click price as the inverse of the ranking function (as in [4]), and on the top of the original formula, we further introduce a scaling parameter as a non-linear projection from the hidden state $\boldsymbol{s}_t$. This parameter helps 'calibrate' the range of the click price, and the prediction of the click price becomes

$$\begin{aligned}&\hat{\psi}(ad, \boldsymbol{a}) = a_2 \cdot f_{a_3}(CVR, CTR, \boldsymbol{s}_t, \boldsymbol{z}_t) + a_4 \cdot f_{a_5}(CVR, price, \boldsymbol{s}_t, \boldsymbol{z}_t); \\ &\overline{\text{price}} = \frac{\hat{\phi}(ad', \boldsymbol{a}) - \hat{\psi}(ad, \boldsymbol{a})}{f_{a_1}(CTR, \boldsymbol{s}_t, \boldsymbol{z}_t)}; \\ &\text{click-price}_{pred} = \mathtt{sigmoid}(\boldsymbol{s}_t^T \boldsymbol{w}_{\text{scale}}) \cdot \overline{\text{price}}.\end{aligned} \tag{6}$$

where $\boldsymbol{w}_{\text{scale}}$ is the parameter (linear weights) for the transformation. One can therefore train the parameter $\boldsymbol{w}_{\text{scale}}$ together with the RNN model by regressing the prediction of the clicked price to the real click prices using square loss.

$$\mathcal{L}_{RNN} = \|\text{click-price}_{pred} - \text{click-price}\|_2^2. \tag{7}$$

With the above steps, we have introduced the ability to capture sequential information to the simulated environment, and the learning signal is more precise since the model can update itself by learning from the real click prices. In order to further improve the environment model, the model has to further satisfy the following properties: the ability to separate positive and negative samples, ensure consistency of the learning signal, and the interpretability of the model. In the rest of this section, we introduce changes to the model to satisfy these requirements, and present our model in its entirety.

3.2.1 *Separating positive and negative signals.*

We first ensure that learning signals for positive and negative samples are clearly distinguishable. Intuitively, a desired environment should be able to provide distinct signals between advertisements that are more likely to be clicked and those that are not given a particular context. We can thus leverage information on whether an advertisement was clicked or not to determine if a sample is positive or negative. This information is readily available for offline training based on interaction logs from the production sponsored ranking system; furthermore, during the online prediction, we do *not* require the status of click as an input.

We introduce the following difference in behavior for positive and negative samples based on whether a click occurred to the model:

- We first clip the prediction of click price of the positive and negative samples into different ranges. For negative samples, we allow prediction of the click price to become negative; however, for positive samples, it is not desired to have negative click prices (and rewards). Therefore, for the positive samples, we clip the click price to no lower than 0.
- We then encourage $\boldsymbol{a}$ values to increase for the positive samples. Since $f_{\boldsymbol{a}}(\cdot)$ is monotonic, this separation helps the environment to provide ranking values that favor the clicked advertisements. We observe that since the reward can be negative, to avoid negative signals, the agent tends to make *all* the $\boldsymbol{a}$ values close to 0. To address this, we introduce a normalization term to explicitly encourage increment of $\boldsymbol{a}$ values for the positive samples. With this term, the reward for the positive samples becomes

$$r = \text{click-price}_{pred} \cdot CTR + \lambda \frac{\|\boldsymbol{a}\|}{\|\boldsymbol{a}^{max}\|}, \tag{8}$$

  where λ is an interpolation factor, and $\boldsymbol{a}^{max}$ is the upper-bound of the values of the vector (recall that the value of $\boldsymbol{a}_i$ is bounded between 0 and a positive number).

3.2.2 *Improving the consistency of the learning signal.*

Another important aspect of our goals is to increase the consistency between the learning signal and the goal of learning. We observe that the reward computed by Eq (2) can be negative; and when this happens, the reward signal cannot keep consistent with the learning goal. To see this, consider the case when $\text{click-price}_{pred} < 0$. To keep the learning goal consistent, the monotonic nature nature of $f_{\boldsymbol{a}}(\cdot)$ requires that we *decrease* the $\boldsymbol{a}$ values when the advertisement is negative. However, the prediction of the reward is controlled by Eq (6) which means that the agent can increase the reward by *increasing* the value of a_1. Learning this behavior undermines the agent's ability to reward consistently.

To remedy this issue, we provide a further treatment for the negative samples with a negative reward (note that the rewards of the positive samples are already clipped to be non-negative). We define `raw-click-price` as $\hat{\phi}(ad', \boldsymbol{a}) - \big(a_2 f_{a_3}(CVR, CTR, \boldsymbol{s}_t, \boldsymbol{z}_t) + a_4 f_{a_5}(CVR, price, \boldsymbol{s}_t), \boldsymbol{z}_t\big)$, and when it is less than 0, the prediction of the click price is computed as

$$\mathrm{NM}(a_1) = \frac{f_{a_1}(CTR, \boldsymbol{s}_t, \boldsymbol{z}_t)}{f_{a_1^{min}}(CTR, \boldsymbol{s}_t, \boldsymbol{z}_t)}$$

$$\text{click-price}_{pred} = \texttt{sigmoid}(\boldsymbol{s}_t^T \boldsymbol{w}_{\text{scale}}) \cdot \frac{\big(\texttt{raw-click-price} - \mathrm{NM}(a_1)\big)}{f_{a_1^{max}}(CTR, \boldsymbol{s}_t, \boldsymbol{z}_t)}, \quad (9)$$

Note that in the new form, the denominator is fixed, and the increment of a_1 value *decreases* the reward. In this case, the agent will learn to decrease a_1, which is consistent with the goal of learning. Besides, note that in Eq (9), we divide the value of $f_{a_1}(CTR, \boldsymbol{s}_t)$ by $f_{a_1^{min}}(CTR, \boldsymbol{s}_t)$ to make sure the contribution of f_{a_1} does not vanish when its numerical scale is small; and we put the denominator as $f_{a_1^{max}}(CTR, \boldsymbol{s}_t)$ to make sure the quantity of the click price prediction lies in the same scale with other samples.

3.2.3 Connections with the CTR-based model as a base case. The ranking function should ideally be able to connect to the CTR-based model for the purpose of interpretability. To this end, the following approaches are feasible: 1. pick a non-linear function such that $f_{a_1=0}(CTR, \boldsymbol{s}_t, \boldsymbol{z}_t) = CTR$,; and 2. pick a non-linear function such that $f_{a_1=0}(CTR, \boldsymbol{s}_t, \boldsymbol{z}_t) = 1.0$, and put a CTR multiplicative factor ahead of the first term of Eq (5). Note that by either of the above strategies, Eq (5) returns the CTR value when all the a values are 0. That is to say, if the agent *does not learn* at all, then the rewards of the model is determined by $CTR \cdot bid$. This allows for environment model to be rolled out in a production context under the same conditions as the CTR-based policy. If the agent learns anything, it can be viewed as *adjustments* on the top of the CTR-based method. In practice, we adopt the swish activation function as the non-linear function of $f_{a_1}(\cdot)$ [16], which follows the first type of approaches as above.

Finally, apart from the above major features, we also design a mechanism to balance the numerical terms in the ranking function. Note that the ranking function of Eq (5) is a summation of 3 terms. In practice, we found it is important to bound the price input for $f_{a_5}(.)$, as the scale of such a term can overwhelm other terms. Hence, we use logarithmic scale for input price. We remark that in [4], a similar form of calibration was adopted.

The overall algorithm. Taking these together, the algorithm for the environment model can be represented as Algorithm 1.

Algorithm 1: The algorithm for the environment model.

```
Initialize the RNN-GRU model GRU($\cdot$) randomly;
// training of the environment
for each epoch do
    for each batch do
        Compute the click price with Eq (9) with fixed $\boldsymbol{a} = \boldsymbol{0}$
          or random $\boldsymbol{a}$ vectors;
        Train the GRU model with loss Eq (7);
// Reinforcement Learning environment
for each action $\boldsymbol{a}$ by the RL agent do
    Return click price by Eq (9) with given $\boldsymbol{a}$ and the state $\boldsymbol{s}_t$;
    Clip the click price by ranges;
    Compute the reward $r$ with Eq (8), and retrieve $\boldsymbol{s}_{t+1}$
      from the GRU model;
    Return $r$ and $\boldsymbol{s}_{t+1}$ to the agent;
// Reinforcement Learning training
Train the Deep Deterministic Policy Gradient
  Reinforcement Learning in the standard manner ([18]);
// Prediction and Production phase
for each session do
    Initialize $\boldsymbol{s}_t = \boldsymbol{0}$ ;
    for each advertisement item do
        Obtain $\boldsymbol{a}$ by inputting $\boldsymbol{s}_t$, $\boldsymbol{z}_t$, and the
          meta-information to the RL agent ;
        Compute the ranking score for the item by Eq (5) ;
        Update $\boldsymbol{s}_t$ by Eq (4) ;
    Rank the items by the ranking scores;
```

3.3 The DDPG Reinforcement Learning Agent

We now briefly introduce the DDPG model, which is the RL Agent adopted in this paper (also in [4]). The section is self-contained, and we assume the readers already have in-depth knowledge of RL. Due to space limit, we only include the outline and the justification of the method. For an in-depth introduction to RL, keen readers can refer to [19]; and [13] contains a more detailed treatment to the DDPG model.

Similar to many policy gradient methods, the DDPG method adopts the *actor-critic structure*, and it outputs actions as *deterministic values*, and optimize the loss function

$$\mathcal{L} := \frac{1}{N} \sum_{t=1}^{N} \left(Q_\phi(\boldsymbol{s}_i, \boldsymbol{a}_i) - (r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \gamma \max_\theta Q_\phi(\boldsymbol{s}_{i+1}, \mu_\theta(\boldsymbol{s}_{i+1}))) \right)^2. \quad (10)$$

Note that when $i = N$, $\boldsymbol{s}_{i+1}$ is not well defined, so we simply use $(Q_\phi(\boldsymbol{s}_i, \boldsymbol{a}_i) - r(\boldsymbol{s}_i, \boldsymbol{a}_i))^2$ for this specific step.

We note that the learning method in [4] also adopts multiple asynchronous agents to jointly learn the DDPG target. In our method, we did not implement this part as it is much more expensive than the single-agent scenario. We also omit the online learning part of the agent in [4] since it is based on coordinate-wise heuristic and it does not provide insights for competent RL environments.

We also remark that the DDPG method is appropriate for this task. We can first rule out any model-based approach, since the complexity of the environment for sponsored search ranking renders the model-based approaches ineffective. Among the model-free RL algorithms, the available options are the value-based (e.g. Q-learning) and the policy gradient-based methods. Since the action vector $\boldsymbol{a}$ in our case is continuous, it is unclear how to optimize it (see [17] for more insights). Therefore, the most realistic path is to find our method from the policy gradient-based algorithms like DDPG.

4 EXPERIMENT

The main contribution and novelty of our paper lie presenting a set of principles for the design of the environment. Our experiments focus on showing the advantages of these design principles on ranking performance. By comparing our *environment* model against the original one in [4] we can examine the contribution of the design in Section 3.2 to performance.

We adopt the DDPG method to ensure a fair comparison. Note that compared to the agent used in [4], we omit many enhancements to the learning agents such as multi-agent learning and dueling under *both* of the environment models. Therefore, the fairness of comparison is *not* negatively affected since the same type of agent (without the enhancements) is applied across the environments. Furthermore, the enhancements of the agent in [4] are independent from the environment model, which means it should contribute equally to the agent under different environments. In addition, He et al. [4] outlines the primary need for an online update to the inconsistency between the real online environment and the offline one due to dynamic distribution and sequential correlation between the continuous user behavior. Our model remedies these issues without an online learning component. Therefore, we determine the online learning characteristics of our model an important future area of study to determine if further work is required.

Data and evaluation metrics. We use real sponsored search auction data from the Overstock.com to perform experiments. We train on 7 days of sponsored search auctions and evaluate on 1 day. We train the model with the policy network as a 25-layer ResNet (see [3]) and the value network as the 2-layer fully-connected MLP. Ranking during the testing phase is performed in the following way: for any advertisement items, we can obtain the action vector $\boldsymbol{a}$ by passing the item to the environment model; we then compute the ranking score with Eq (5), and use this score to rank items in each auction.

We use NDCG to evaluate our rankings [9]. To get results that are as unbiased as possible, we use *simulated clicks* as the ground truth. For evaluation we rerank all auctions from one day and simulate clicks on the reranked auction results using historical CTR at each position. We then calculate NDCG using the simulated clicks. A better comparison with the *online* click results can be an interesting direction to explore in the future.

Results. We test our environment model against the original one (as in [4] and without any components in Section 3.2) and a randomly initialized model. The latter comparison is included to justify the validity of *learning* in the environment model. Furthermore, we compare our model against several other ranking methods, including Bid, smoothed Purchase Rate, AdRank (Bid * CTR) and smoothed CTR. The results are summarized in Table 1 (numerical precision rounded to 10^{-4}). The features of each advertisement include smoothed hierarchical engagement rates for clicks, cart adds and purchases across categories and partners, along with query and product embeddings built from Overstock historical click data.

From the table, it can be found that the model with the trained new environment provides the most competitive performance. We also observe that the randomly-initialized environment performs almost identically to the original environment, while the trained environment considerably outperforms the latter. This highlights the crucial role of learning in the environmental model.

The importance and effectiveness of learning in the environmental model can also be shown by the progress made by training. Specifically, we show in Figure 2 the loss curve for training the GRU model with the loss function Eq (7). From the curve, it can be found that the ℓ_2 loss declines drastically, indicating that the quality of the reward prediction before and after training are considerably different. Moreover, as we can see in Table 2, the ranking quality gradually improves as the number of training epochs of the environment increases. This also validates the claim that a trainable environment model achieves superior performances.

Table 1: Ranking performance for different methods.

Ranking Method	NDCG Click
Bid	0.0458
Purchase Rate	0.0485
AdRank	0.0530
CTR	0.0534
RL + original env ([4])	0.0530
RL + random new env (no training)	0.0529
RL + new env	**0.0558**

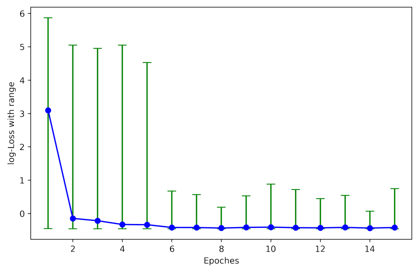

Figure 2: The curve for the loss function Eq (7) of the environment model. The data is obtained with 50 runs (expect a few unconverged records), and the y-axis is re-scaled by logarithm.

Table 2: Ranking performance for the environment model trained at different epochs. The most relevant set of features are used.

Training Epochs	NDCG Click	percentage change
0	0.0529	–
1	0.0523	-1.13%
2	0.0530	+0.19%
5	0.0531	+0.38%
9	0.0536	+1.32%
14	0.0541	+2.27%
15	0.0558	+5.48%

Apart from the ranking performance, we can also examine the improvement of the environment model from the perspective of the training of the Reinforcement Learning agent. We can illustrate the reward curve of the same DDPG agent under different environments as in Figure 3. From the figure, it can be observed that the agent under the original environment suffers from negative rewards; in contrast, the agent under the trained environment model can achieve relatively stable increments on the rewards. Note that Figure 3 is a good reference for both the *difficulty* for the agent to learn and the *consistency* of the reward signals.

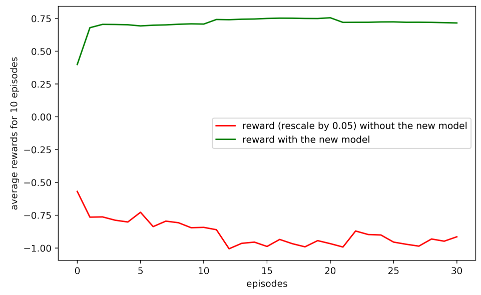

Figure 3: The average rewards of the agent for every 10 episodes. Note that the reward for original method is rescaled for illustration purpose.

We conclude this section by noting that the experimental results in this section do *not* invalidate the effectiveness of the original environment model in [4]. Compared to the work of [4], our data is significantly different. More importantly, the agent in [4] is far more complicated (although still under the DDPG framework) and many effective heuristic methods are not adopted in our agent. Again, the purpose of the experiments is to show the effectiveness of the modifications we introduced to the environment model.

5 CONCLUSION

In this paper, we study the challenges and strategies for RL when deployed on sponsored search ranking problems. We highlight the key challenge of environment exploration in this setting and the necessity of using simulated environment models. Subsequently, we adopt the idea of [4] for the simulated environment models, and propose several important changes, which lead to a sequential and learnable environment model with enhanced properties. We test the proposed model on real-life production data from Overstock.com, and empirical results validate the improved performance of the new model.

In future work, we will focus on real-world deployment of the RL model. In an online production setting, re-training the sequential learnable model can be done in batch periodically to keep the environment and actor-critic models up-to-date.

In conclusion, the paper makes two significant contributions to the area of sponsored search ranking. Firstly, the proposed new environment model offers better performance and higher-quality learning signals for RL agents, and it can be widely adopted in this area. Secondly, it proposes several principles for RL-based sponsored ranking environments, which contributes to our general understanding of the ways RL is used in the area.

REFERENCES

[1] Maarten de Rijke. 2019. Reinforcement learning to rank. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 5–5.
[2] Vali Derhami, Elahe Khodadadian, Mohammad Ghasemzadeh, and Ali Mohammad Zareh Bidoki. 2013. Applying reinforcement learning for web pages ranking algorithms. *Applied Soft Computing* 13, 4 (2013), 1686–1692.
[3] 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.
[4] Li He, Liang Wang, Kaipeng Liu, Bo Wu, and Weinan Zhang. 2018. Optimizing Sponsored Search Ranking Strategy by Deep Reinforcement Learning. *arXiv preprint arXiv:1803.07347* (2018).
[5] Li He, Liang Wang, Kaipeng Liu, and Weinan Zhang. 2018. Deep Policy Optimization for E-commerce Sponsored Search Ranking Strategy. In *Proceedings of the 2018 AdKDD and TargetAd*.
[6] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
[7] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 368–377.
[8] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Morgane Lustman, Vince Gatto, Paul Covington, et al. 2019. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767* (2019).
[9] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.
[10] Shihao Ji, Ke Zhou, Ciya Liao, Zhaohui Zheng, Gui-Rong Xue, Olivier Chapelle, Gordon Sun, and Hongyuan Zha. 2009. Global ranking by exploiting user clicks. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. 35–42.
[11] Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. 2018. Real-time bidding with multi-agent reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2193–2201.
[12] Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. 2013. Learning to rank for recommender systems. In *Proceedings of the 7th ACM conference on Recommender systems*. 493–494.
[13] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1509.02971
[14] Yifei Ma, Balakrishnan Narayanaswamy, Haibin Lin, and Hao Ding. 2020. Temporal-Contextual Recommendation in Real-Time. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2291–2299.
[15] Harrie Oosterhuis and Maarten de Rijke. 2020. Unifying Online and Counterfactual Learning to Rank. *arXiv preprint arXiv:2012.04426* (2020).
[16] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2018. Searching for Activation Functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=Hkuq2EkPf
[17] Moonkyung Ryu, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja, and Craig Boutilier. 2019. Caql: Continuous action q-learning. *arXiv preprint arXiv:1909.12397* (2019).
[18] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014 (JMLR Workshop and Conference Proceedings, Vol. 32)*. JMLR.org, 387–395. http://proceedings.mlr.press/v32/silver14.html
[19] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[20] Wenjin Wu, Guojun Liu, Hui Ye, Chenshuang Zhang, Tianshu Wu, Daorui Xiao, Wei Lin, and Xiaoyu Zhu. 2018. EENMF: An End-to-End Neural Matching Framework for E-Commerce Sponsored Search. *CoRR* abs/1812.01190 (2018). arXiv:1812.01190 http://arxiv.org/abs/1812.01190
[21] Byoung-Tak Zhang and Young-Woo Seo. 2001. Personalized web-document filtering using reinforcement learning. *Applied Artificial Intelligence* 15, 7 (2001), 665–685.
[22] Jianghong Zhou and Eugene Agichtein. 2020. RLIRank: Learning to Rank with Reinforcement Learning for Dynamic Search. In *Proceedings of The Web Conference 2020*. 2842–2848.