A Dynamic Bidding Strategy Based on Model-Free Reinforcement Learning in Display Advertising

MENGJUAN LIU, LI JIA XING, ZHENGNING HU, JINYU LIU, AND XUYUN NIE

Network and Data Security Key Laboratory of Sichuan Province, University of Electronic Science and Technology of China, Chengdu 610054, China

Corresponding author: Mengjuan Liu (mjliu@uestc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61202445 and Grant 61472064.

ABSTRACT Real-time bidding (RTB) is one of the most striking advances in online advertising, where the websites can sell each ad impression through a public auction, and the advertisers can participate in bidding the impression based on its estimated value. In RTB, the bidding strategy is an essential component for advertisers to maximize their revenues (e.g., clicks and conversions). However, most existing bidding strategies may not work well when the RTB environment changes dramatically between the historical and the new ad delivery periods since they regard the bidding decision as a static optimization problem and derive the bidding function only based on historical data. Thus, the latest research suggests using the reinforcement learning (RL) framework to learn the optimal bidding strategy suitable for the highly dynamic RTB environment. In this paper, we focus on using model-free reinforcement learning to optimize the bidding strategy. Specifically, we divide an ad delivery period into several time slots. The bidding agent decides each impression’s bidding price depending on its estimated value and the bidding factor of its arriving time slot. Therefore, the bidding strategy is simplified to solve each time slot’s optimal bidding factor, which can adapt dynamically to the RTB environment. We exploit the Twin Delayed Deep Deterministic policy gradient (TD3) algorithm to learn each time slot’s optimal bidding factor. Finally, the empirical study on a public dataset demonstrates the superior performance and high efficiency of the proposed bidding strategy compared with other state-of-the-art baselines.

INDEX TERMS Real-time bidding, bid optimization, model-free reinforcement learning.

I. INTRODUCTION

In recent years, online advertising has generated a multi-billion-dollar market share [1]. As one of the most striking advances in online advertising, real-time bidding (RTB) has attracted increasing attention from academia and industry. It improves the efficiency and transparency of the online advertising ecosystem [2]. In RTB, publishers (such as websites and mobile apps) sell the individual ad impressions via hosting real-time auctions, and advertisers are allowed to evaluate and bid for each impression.

Fig. 1 illustrates the typical process of ad delivery in RTB [3]. Specifically, when a user visits a web page, the script for the ad slot embedded on the page will initiate a bid request for the ad impression to the ad exchange (ADX). And then, the ADX forwards the bid request to the connected demand-side platforms (DSPs). With the help of bidding agents on DSPs, each advertiser can estimate the utility of the auctioned impression and make decision of the bidding price. Then, determined by ADX, the one who bids with the highest price wins the auction to show its advertisement and pay for this display. The entire process will be completed in 100 milliseconds. More details of RTB are given in [4].

For the advertisers in RTB, the goal is to maximize their ad campaigns’ revenues under the given budgets. It means that they should make efforts to spend the budget effectively and get more positive user responses, e.g., clicks or conversions [5]. In the RTB context, click refers to the event that occurs when a user interacts with an ad using a mouse click;
**conversion** is an action that the user takes after visiting the landing page of the advertiser, which is regarded as revenue for the advertiser. Typical examples of conversion actions include making a reservation or purchasing a product. Intuitively, conversion actions can better reflect the advertiser’s revenue. Unfortunately, attributing a conversion event to a specific ad display is very difficult because the conversion event can happen minutes, hours, or even days after an ad display. Besides, the number of conversion events is far less than that of click events, making it arduous to establish an excellent conversion rate estimator. As a result, both in academia and industry, the number of clicks is widely used as the advertiser’s revenue. The predicted click-through rate (pCTR) is used to measure the value of the ad impression to the advertiser [11], [18].

In RTB, DSP provides a bidding agent for each advertiser to maximize revenue (the click number) of an ad delivery period under a given budget. To achieve this goal, the bidding agent needs to realize two principal functions, shown as in Fig. 2. One estimates the value of the auctioned impression to the advertiser, and the other determines the bidding price based on the estimated value. The former has been well studied in [6]–[8], usually measured by the predicted click-through rate (pCTR). In this paper, we focus on the latter one, which is optimizing the bidding strategy.

The bidding strategy is closely related to the auction mechanism adopted by ADX. Currently, the majority of RTB platforms take the generalized second pricing (GSP) mechanism [9]. That is, the advertiser with the highest bidding price wins the auction and only pays the second-highest bidding price to the ADX. In the second-price auctions, there is truthfulness that the optimal bidding price for each auctioned item should be equal to its true value for the bidder [3]. Therefore, a straightforward bidding strategy in RTB is defined as formula (1), where \( pctr(i) \) is the predicted CTR of impression \( i \), \( vpc \) means the value per click, and \( bid(i) \) is the bidding price. This formula means that the bidding price equals to the click value multiplied by the impression’s predicted CTR.

\[
\text{bid}(i) = pctr(i) \times vpc \tag{1}
\]

However, this optimal bidding strategy may not be right in RTB since the auction results depend on the market competition, auction volume, and campaign budget [10]. Therefore, many researchers model the bidding decision as a static optimization problem under budget-constraint and use the optimization theory to solve it. For example, researchers in [11] have proposed to seek the key performance indicator (e.g., total clicks) to optimize bidding function, based on the static distribution of input data and market competition models. Unfortunately, such a static optimal strategy derived from historical data may not work well in a new ad delivery period because of the RTB market’s dynamic and unpredictable characteristics. Ideally, each bid is strategically correlated by the available budget, the overall effectiveness of the ad campaign (e.g., the rewards from generated clicks), and the dynamics of the RTB environment [12].

Fortunately, reinforcement learning (RL) [13], [14] may be a promising solution for the above problem. By RL, the bidding decision is modeled as a sequentially dynamic interactive process; the bidding agent determines the bidding price for each impression according to both the immediate and long-term future rewards, to realize budget allocation dynamically cross all impressions in the whole ad delivery period. Typically, Cai et al. [12] tried to generate the bidding price for each impression by using a model-based RL framework (called \( \text{RLB} \)). However, model-based RL approaches such as [12] require storing the state transition matrix and using dynamic programming algorithms, whose computational cost is unacceptable in real-world advertising platforms [15].

Therefore, the authors of [15] proposed using a model-free RL framework to learn the bidding strategy, called Deep Reinforcement Learning to Bid (DRLB). Explicitly, they redefined the bidding function as formula (2), where each impression’s bidding price depends on not only its estimated value but also the bidding factor of the time slot that the impression is arriving. So, the task of bidding strategy is simplified from generating the bidding price for each impression to selecting a set of optimal bidding factors for all time slots, recorded as \( \lambda (1), \lambda (2), \cdots, \lambda (T) \), where \( T \) means that an ad delivery period is divided into \( T \) time slots.

\[
\text{bid}(i, t) = pctr(i) \times \lambda(t), \quad t = 1, 2, \cdots, T \tag{2}
\]

Unfortunately, the authors of [15] found it very difficult to learn the optimal bidding factor of each time slot based on Deep Q learning algorithms (such as DQN) in RTB. The policy in DQN is to choose an action from a preset discrete action space as the optimal bidding factor, resulting in the performance of the bidding strategy heavily depending on the design of action space. However, it is very challenging to design such a discrete action space. If the action space is set small to support coarse-grained bidding factor selection, there is a big gap between the selected one and the real optimal one. If the action space is set huge to support fine-grained bidding factor selection, the computational cost will be significantly increased, and the convergence speed will be substantially decreased. Therefore, DRLB [15] gives up learning each time slot’s optimal bidding factor and selects the bidding factor’s regulating value from the discrete action space at every time slot. Specifically, the action space in DRLB is defined as \{-8%, -3%, -1%, 0%, 1%, 3%, 8%\}. The bidding factor of time slot \( t \) can be updated by the formula (3), where \( \beta_i(t) \) is...
the action (regulating value) selected by the optimal policy. So, the RL-based agent learns the optimal regulating value for each time slot in DRLB.

$$\lambda(t) = \lambda(t-1) \times (1 + \beta_t(t))$$  \hspace{1cm} (3)

where $\beta_t(t) \in \{-8\%, -3\%, -1\%, 0\%, 1\%, 3\%, 8\%\}$ Compared with model-based RL, the model-free RL algorithm, such as DRLB, can avoid calculating the state transition probability matrix and only observing the state from the environment. Thus, it can effectively reduce the computation cost without lowering the performance. However, regulating value selection is still coarse-grained in DRLB, resulting in gaps between the selected and the actual optimal ones.

In this paper, we follow the idea of DRLB and learn the optimal bidding factor generation policy through a model-free RL framework. Unlike DRLB, we exploit the latest Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [16] to learn each time slot’s optimal bidding factor, rather than its regulating value. Our scheme can adjust the bidding price more finely and adaptively than DRLB, so we call it as Fine-grained and Adaptive Bidding (FAB).

In FAB, we first model the auction process as a Markov Decision Process (MDP) [17]. The whole RTB market and Internet users are regarded as the environment. At the beginning of each time slot, the bidding agent first observes the environment and obtains the current state. The bidding agent executes an action (the bidding factor of this time slot) with such a state. And then, during this time slot, the agent calculates the bidding prices for all received bid requests according to the formula (7) (defined in Section III). At the end of this time slot, actual costs and user feedback (such as clicks) are used to generate the immediate reward. Finally, we use the TD3 algorithm to learn the optimal bidding factor generation policy to maximize the total cumulated reward of the whole ad delivery period. The contributions of this work can be summarized as follows:

- To the best of our knowledge, we are the first to use the policy-based model-free RL to optimize the bidding strategy, exactly, directly generating each time slot’s optimal bidding factor, which demonstrates a significant advantage over DRLB.
- To guide the TD3 algorithm toward the optimal bidding factor generation policy efficiently, we design two reward functions based on the cost and user feedback of a time slot. We then discuss the influences of the two reward functions on the bidding performance through experiments.
- Finally, we evaluate the proposed strategy through extensive experiments on a real-world dataset. The results demonstrate the superior performance and high efficiency of the proposed bidding strategy compared with other state-of-the-art baselines.

The rest of this paper is organized as follows. In Section II and III, we introduce the related work and detail the bidding functions in FAB. We formulate the real-time bidding as a MDP in Section IV and describe our TD3-based reinforcement learning solution in Section V. The experimental setup and results are presented and analyzed in Section VI and VII. Finally, we conclude our work and discuss future work in Section VIII.

II. RELATED WORK

In RTB, bidding optimization has always been the focus of research [11], [18], [19]. As described in the introduction, most of the existing solutions formulate the bidding decision as a static optimization problem. They derive the optimal bidding functions by the heuristic algorithms or the optimization methods based on a historical training dataset. This section introduces three representative static bidding strategies: two linear schemes and one nonlinear one. The first linear bidding strategy (called LIN) is defined as (4), where $pctr(i)$ is the predicted CTR of impression $i$. Here, $CPD$ and $avg\_pCTR$ mean the average cost of winning impressions and the average pCTR of impressions in the training set, respectively. Both $CPD$ and $avg\_pCTR$ are determined on the training set, so each impression’s bidding price in a new ad delivery period only depends on the impression’s predicted CTR.

$$bid_{LIN}(i) = pctr(i) \times \frac{CPD}{avg\_pCTR}$$ \hspace{1cm} (4)

As shown in formula (5), another representative linear bidding strategy is based on a heuristic algorithm, called HB. Here, base_bid is a fixed base bid and $avg\_pctr$ is the average value of all impressions’ pCTRs on the training set. In HB, we set the base bid from 1 to 300 (increased one by each time), and calculate each impression’s bidding price according to (5). The base bid that maximizes the total clicks on the training set is recorded as the optimal base bid and taken as the base bid in a new ad delivery period. Like LIN, in HB, each impression’s bidding price in a new ad delivery period only depends on the impression’s predicted CTR.

$$bid_{HB}(i) = pctr(i) \times \frac{base\_bid}{avg\_pctr}$$ \hspace{1cm} (5)

In the nonlinear bidding, ORTB is a typical strategy [11], defined as (6), where the parameters, $c$, and $\lambda$ are both learned on the training set. So, during a new ad delivery period, the bidding price in ORTB also only depends on the predicted CTR of the impression.

$$bid_{ORTB}(i) = \sqrt{\frac{c}{\lambda}pctr(i) + c^2 - c}$$ \hspace{1cm} (6)

The above three static bidding strategies have a common problem that their bidding functions in a new ad delivery period are only related to the estimated value (pCTR) of an individual impression. All parameters except for the impression’s estimated value have been determined based on the training set. Therefore, these static bidding strategies usually fail to adapt well to a new ad delivery period, when the RTB environment changes dramatically (such as different impression distributions and market competitions). Intuitively, in a highly dynamic RTB environment, each bid should consider not only the benefit of a single bid but its impact on future profits.
Recently, reinforcement learning (RL) is introduced into the RTB bidding decision to adapt to the dynamic changes of the auction environment. RLB is a typical RL-based bidding framework proposed in [12]. The bidding problem is taken as a MDP to choose the optimal bidding price for each impression sequentially. In RLB, at each time step (triggered by an impression arriving), the bidding agent first observes a state (about the current RTB environment) and selects an action from the action space under the state. Specifically, the action in RLB is the bidding price for the auctioned impression, and the action space is defined as \( \{0, 1, \ldots, 300\} \) (10^{-3} Chinese FEN). RLB adopts dynamic programming to solve the optimal action-selection policy based on a model-based RL model. By following the idea of RLB, the authors in [20] regard the budget-constrained bidding problem as a Constrained Markov Decision Process (CMDP) and use the linear programming method [21] to solve the CMDP. Unfortunately, such model-based RL methods require the explicit state transition probability matrix, challenging to represent due to the huge computational cost in the real world.

As an improvement, researchers seek to solve the MDP by using model-free RL algorithms. In [22], the authors formulate a robust MDP model at an hour-aggregation level and propose a control-by-model framework for RTB in sponsored search advertising. Similarly, the authors of DRLB [15] formulate the budget-constrained bidding problem as a bidding factor control problem based on the linear bidding function shown as formula (2). It does so by leveraging a value-based model-free RL algorithm. However, the value-based model-free RL algorithms, such as DQN in [15], [22], may have convergence issues in practice. All have shown to be unable to converge to any policy for simple MDP and simple function approximator [23], [24]. In contrast, policy gradient methods can theoretically guarantee better convergence and keep relatively high efficiency in high-dimensional or continuous action space [24]. Therefore, Actor-Critic architecture that combines the policy-based and the value-based methods is developed. Due to the excellent performance, some new model-free RL algorithms are presented based on Actor-Critic architecture, such as Asynchronous Advantage Actor-Critic (A3C), Deep Deterministic Policy Gradient (DDPG), and Twin Delayed Deep Deterministic policy gradient (TD3). As the latest algorithm based on Actor-Critic architecture, TD3 solves the problems of overestimation bias and high variance in Actor-Critic architecture [26], [27]. It also dramatically improves the speed and performance of generating continuous actions during training.

On the other hand, RLB and DRLB are both based on reinforcement learning with discrete action spaces. One problem with discrete action spaces is that we must extend it manually for tasks requiring fine control of actions. Besides, the authors in [25] point out that discrete action space throws away the action domain’s structure, which may be essential for solving many problems. Thus, in this paper, we optimize the bidding strategy by leveraging the TD3 algorithm [16], which supports the optimal policy to generate the continuous action values under a given range, as the time slot’s bidding factors. Our scheme is the first job to use the policy-based model-free RL to optimize the RTB bidding strategy to the best of our knowledge. Notably, our optimal action policy is to generate each time slot’s bidding factor rather than the bidding price for each impression. Table 1 analyzes these representative bidding strategies’ characteristics, and Fig. 3 illustrates the bidding strategies’ evolution.

### III. BIDDING FUNCTION

In this section, we discuss the bidding function in FAB. Before that, we first review the problems of DRLB. As mentioned in the introduction, DRLB uses the formula (3) to generate the bidding factor of the current time slot and the formula (2) to calculate the bidding price for each impression. Therefore, the optimal action-selection policy in DRLB only needs to choose a sequence of regulating values from a manually preset action space. Although this simplification makes RL’s computational cost significantly reduced, there are still two challenges in DRLB. One is to determine the bidding factor’s initial value \( \lambda(0) \), and the other is to set up the discrete action space manually. Apparently, improper \( \lambda(0) \) and action space will lead to a big gap between the updated \( \lambda(t) \) and its optimal value. Besides, when more fine-grained regulating values needed, the action space must be extended to very large, which will significantly increase the computing cost and weaken the convergence of DRLB.

To solve the above problems, we redefine the bidding function as the formula (7) in FAB. Here \( ecpc(B) \) is the expected cost per click under budget \( B \), calculated by the formula (8). It is determined by \( avg \cdot pctr \) and base bid\(^*\), where \( avg \cdot pctr \) is the average value of all impressions’ pCTRs on the training set, and base bid\(^*\) is the optimal base bid that maximizes the total clicks on the training set obtained by a heuristic algorithm (refers to HB). In FAB, the policy will directly generate the optimal bidding factor for each time slot, recorded as \( \{a_1, a_2, \ldots, a_T\} \). To produce the continuous bidding factors within a given range, we adopt the policy-based RL algorithm

---

| Strategy | Static/Dynamic | Method | Adaptiveness | Granularity | Action | Action Space |
|---------|----------------|--------|--------------|-------------|--------|--------------|
| LIN     | Static         | Linear/Statistical method | Not support | Impression | /     | /            |
| HB      | Static         | Linear/Heuristic method   | Not support | Impression | /     | /            |
| ORTB    | Static         | Non-linear/Heuristic method | Not support | Impression | /     | /            |
| RLB     | Dynamic        | Model-based RL/Dynamic programming | Support | Impression | Bidding price | Discrete value |
| DRLB    | Dynamic        | Model-free RL/DQN          | Support | Time slot | Bidding factor’s regulating value | Discrete value |
| FAB     | Dynamic        | Model-free RL/TD3           | Support | Time slot | Bidding factor | Continuous value |
Static Strategy can not adapt to the dynamic RTB environment of the new ad delivery period.

**FIGURE 3.** The evolution of the bidding strategies.

(i.e., TD3) instead of the value-based RL algorithm (such as DQN in DRLB).

The former has better robustness and convergence than the latter. Formula (7) also shows the transformation between $\lambda(t)$ and $a_t$, so the optimal $\lambda^*(t)$ can be achieved by using $\lambda^*(t) = \frac{ecpc(B)}{1 + a_t}$. It means the expected cost per click at each time slot should change over time to adapt to different RTB environments.

$$bid(i, t) = pctr(i) \times \lambda(t) = pctr(i) \times \frac{ecpc(B)}{1 + a_t}$$

$$ecpc(B) = \frac{base\_bid^*}{avg\_pctr}$$

**IV. PROBLEM AND FORMULATION**

Mathematically, we formulate the problem of real-time bidding as a Markov Decision Process (MDP). An episode comprises all the impressions received by a specific advertiser in an ad delivery period. Generally, a MDP is represented by a tuple $<S, A, P, R, \gamma>$, where $S$ is the state space $(s_t \in S)$, and $A$ is the action space $(a_t \in A)$. The state transition probability distribution matrix is denoted as $P$, where $p(s_{t+1} | s_t, a_t) \sim P$ represents the state transition probability from state $s_t$ to another state $s_{t+1}$ when taking action $a_t$. And $R$ is the reward function that decides the immediate reward received after taking action $a_t$ under the state $s_t$.

The interaction process between the bidding agent and the RTB environment can be displayed in Fig. 4. First, the agent divides the episode into $T$ time slots. And at the beginning of each time slot, it obtains a state $s_t$ by observing the current RTB environment, which is described by the statistics of the latest finished time slot $t-1$. Then the agent generates an action $a_t$ according to the deterministic action policy $a_t = \pi(s_t)$ to calculate the bidding price for each impression based on the formula (7). After time slot $t$ finished, the next state $s_{t+1}$ is observed. Meanwhile, the user feedback and budget cost are both considered as the reward signal $(r_t(s_t, a_t))$ of time slot $t$ to guide the agent toward the optimal policy.

In this paper, we adopt the TD3 algorithm to learn the optimal policy. The next state is directly generated by observing the environment without the requirement for the state transition matrix. The bidding agent’s goal is to maximize the cumulative discount reward. Formula (9) defines the cumulative discount reward, where $\gamma$ represents the discount factor, indicating the uncertainty in the future. As shown in the formula (10), the policy that maximizes the cumulative discount reward is optimal. It is noted that the bidding decision will stop once the budget of the delivery period is exhausted, namely that the total cost should not exceed the total budget in our RTB environment.

$$R = \sum_{t=1}^{T} \gamma^{t-1} r_t (s_t, a_t), \quad \text{where } \gamma \in [0, 1]$$

$$\pi^* = \arg\max_{\pi} E \left[ \sum_{t=1}^{T} \gamma^{t-1} r_t (s_t, a_t) | \pi \right]$$

More specifically, we describe the core elements of the MDP as follows:

- **State:** we regard the statistical information of the latest finished time slot $t-1$ as the observations of the environment at time slot $t$ and define the state $s_t$ as $(avbudget\_ratio(t-1), cost\_ratio(t-1), ctr(t-1), win\_rate(t-1))$, where each parameter is described as:

  - $avbudget\_ratio(t-1)$: the average available budget ratio for the remained $T-(t-1)$ time slots, defined by the formula (11), where $remain\_budget(t-1)$ denotes the available budget at the...
end of time slot $t-1$, and $B$ is the total budget;
\[
\text{avbudget\_ratio}(t-1) = \left( \frac{\text{remain\_budget}(t-1)}{T - (t-1)} \right) / \left( \frac{B}{T} \right)
\]

(11)

\[\text{cost\_ratio}(t-1) = \text{cost}(t-1)/B \]

(12)

$\text{ctr}(t-1)$: the click-through rate of time slot $t-1$, defined by the formula (13), where $\text{imps}(t-1)$ means the number of the winning impressions. Note that if the agent doesn’t win any impression at time slot $t-1$, set $\text{ctr}(t-1) = 0$;
\[
\text{ctr}(t-1) = \frac{\text{clks}(t-1)}{\text{imps}(t-1)}
\]

(13)

$\text{win\_rate}(t-1)$: the win rate of auctions of time slot $t-1$, defined by the formula (14), where $\text{bid\_nums}(t-1)$ denotes all impressions received by the agent at time slot $t-1$.
\[
\text{win\_rate}(t-1) = \frac{\text{clks}(t-1)}{\text{bid\_nums}(t-1)}
\]

(14)

Note that the state $s_1$ is initialized with $(1, 0, 0, 0)$ in each episode.

- **Action**: we define the action $a_t$ at time slot $t$ as the bidding factor (a continuous value within a given range, such as $[-0.99, 0.99]$) to adjust the expected cost per click (ecpc($B$)) of time slot $t$. In FAB, the action is generated by the policy, $a_t = \pi(s_t)$, where $\pi$ is parameter vector of the policy. In addition, as shown in the formula (15), to ensure the balance of exploration and exploitation, we add the Gaussian noise $N(0, \delta)$ to the deterministic policy.
\[
a_t = \pi(s_t) + N(0, \delta)
\]

(15)

- **Reward**: the reward function is defined as (16). Here $\text{clks}(t)$ and $\text{cost}(t)$ are respectively the click number obtained by FAB at time slot $t$ and corresponding actual cost; $h\cdot \text{clks}(t)$ and $h\cdot \text{cost}(t)$ are the click number obtained by HB and corresponding actual cost, which is defined in (5).
\[
\text{reward}_f(t) = \begin{cases} 
\varepsilon_a, & \text{if clks}(t) > h\cdot \text{clks}(t) \text{ and cost}(t) < h\cdot \text{cost}(t) \\
\varepsilon_b, & \text{if clks}(t) > h\cdot \text{clks}(t) \text{ and cost}(t) > h\cdot \text{cost}(t) \\
\varepsilon_c, & \text{if clks}(t) < h\cdot \text{clks}(t) \text{ and cost}(t) < h\cdot \text{cost}(t) \\
\varepsilon_d, & \text{if clks}(t) < h\cdot \text{clks}(t) \text{ and cost}(t) > h\cdot \text{cost}(t) 
\end{cases}
\]

(16)

In FAB, we design a new way to make reward rules by comparing the costs and clicks in FAB and HB at each time slot. That is, if the results are better than those in HB, the environment gives a positive reward to the bidding agent, else a negative reward. The reason for this design is that HB has achieved good performance in practice. The specific rules are designed as:

- if FAB gets more clicks with more cost, the environment feedbacks a **positive** reward (in experiments, $\varepsilon_a = 0.005$);
- if FAB gets fewer clicks with more cost, the environment feedbacks a **negative** reward (in experiments, $\varepsilon_c = -0.0025$);
- if FAB gets more clicks with less cost, the environment feedbacks a **positive** reward (in experiments, $\varepsilon_b = 0.001$);
- if FAB gets fewer clicks with less cost, the environment feedbacks a **negative** reward (in experiments, $\varepsilon_d = -0.005$).

For comparison, we also use the number of clicks as the immediate reward at each time slot, shown as the formula (17), where $\varphi$ is the scaling factor to reduce the immediate reward’s value (in experiments, $\varphi = 1000$). We set the immediate reward to a small value in two kinds of reward functions because we found that the RL algorithm’s performance and convergence speed decreased when the reward value was set large, easy to make the policy fall into local optimum.
\[
\text{reward}_h(t) = \frac{\text{clks}(t)}{\varphi}
\]

(17)

**V. SOLUTION BASED ON TD3**

This section introduces how to learn the optimal action generation policy based on the TD3 algorithm. The architecture and the pseudocode of our learning algorithm are given in Fig. 5 and Algorithm 1. Table 2 clarifies the meaning of various variables in the learning algorithm.

As shown in Fig. 5, there are observation and training processes in our algorithm. Before training, we first need to initialize a replay buffer, recorded as $M$. The replay buffer is a fixed-sized cache. Transitions were sampled from the interaction process between agent and environment according to an exploration policy and each transition $<s_t, a_t, r_t, s_{t+1}>$ was stored in the replay buffer. In FAB, each time slot is taken as a time step, so a transition represents bidding results at time slot $t$. Specifically, we first use a randomly initialized Actor network with exploration noise to generate each time slot’s action (bidding factor), $a_t = \pi(s_t) + \varepsilon (\varepsilon \sim N(0, \sigma))$. Here, $s_t$ is the agent’s observation of the auction environment at the beginning of time slot $t$, described by the statistics of the last finished time slot $t-1$. The agent uses $a_t$ to determine each auctioned impression’s bidding price during time slot $t$ according to formula (7). After time slot $t$ finished, the agent uses the obtained total clicks and the actual cost of this time slot to compute the immediate reward based on (16), recorded as $r_t$. At the same time, the agent computes the statistics of time slot $t$ as the next state $s_{t+1}$. The agent repeats the above steps to participate in the bidding at the subsequent time slots and obtains 24 transitions. We store the 24 transitions in the replay buffer. We can repeat the above observation process many times to get more transitions by introducing the exploration noise in action.

Then, enter the training process. As shown in Fig. 5, the architecture consists of two parts: EvalNet and TargetNet. EvalNet includes one Eval Actor network and two Eval Critic networks. The Eval Actor network is the policy model for generating each time slot’s bidding factor, and two Eval Critic networks are used to compute the Q values given a state and an action. TargetNet includes one Target Actor network.
The structure of our learning algorithm based on TD3.

Algorithm 1 Learning Algorithm in FAB

Randomly initialize two critic networks $Q_{\theta_1}$ and actor network $\pi_{\phi}$ with random weights $\theta_1 (i = 1, 2)$ and $\phi$ Initialize target networks $\theta_i' (i = 1, 2)$ and $\phi' \leftarrow \phi$ Initialize replay buffer $M$

for episode $= 1, E$ do

Receive initial state $s_1$

for $t = 1, R$ do

Generate an action $a_t$ according to (15)

Obtain the reward $r_t$ from (16) and observe next state $s_{t+1}$

Sample mini-batch of $N$ transitions $<s_i, a_i, r_i, s_{i+1}>$ from $M$

Compute $\tilde{a} = \pi_{\phi'} (s_{t+1}) + clip (N (0, \delta'), -c, c)$

$y_t = r_t + \gamma \min (Q_{\theta_1} (s_{t+1}, \tilde{a}), Q_{\theta_2} (s_{t+1}, \tilde{a}))$

Update $\theta_i (i = 1, 2)$ by minimizing the loss:

$Loss(\theta_i) = \frac{1}{N} \sum_{t=1}^{N} (y_t - Q_{\theta_i} (s_t, a_t))^2, (i = 1, 2)$

if $t \mod k$ then

Update $\theta_1$ by using the sampled gradient:

$\nabla_{\theta_1} J (\phi) \approx \frac{1}{N} \sum_{j=1}^{N} \nabla_\theta \nabla_{\phi} Q_{\theta_1} (s_t, a_t) |_{a = \pi_{\phi} (s_t)} \nabla_{\pi_{\phi}} (s_t)$

Update the target networks:

$\theta_i' \leftarrow \theta_i' + (1 - \sigma) \theta_i'$ and $\phi' \leftarrow \phi' + (1 - \sigma) \phi$

end if

end for

end for

and two Target Critic networks, all of which help update two Eval critic networks. In our learning algorithm, only the parameters of the networks in EvalNet are updated based on training; the three networks’ parameters in TargetNet are copied from the corresponding networks in EvalNet by using “soft” target updates.

Now, we introduce the details of the training. As shown in Algorithm 1, for each round, we first randomly sample $N$ transitions from the replay buffer to form a mini-batch and input them into the networks for updating their parameters once. For each $<s_t, a_t, r_t, s_{t+1}>$, $s_t$ is input into the Eval Actor network to generate the action, $\pi_{\phi} (s_t)$, where $\phi$ is the parameters of the Eval Actor network. Furthermore, $\pi_{\phi} (s_t)$ is fed into two Eval Critic networks respectively to calculate two Q values, recorded as $Q_{\theta_1} (s_t, \pi_{\phi} (s_t))$ and $Q_{\theta_2} (s_t, \pi_{\phi} (s_t))$. Here, $\theta_1$ and $\theta_2$ are two Eval Critic networks’ parameters.

So we can update the Eval Actor network’s parameters by using the deterministic policy gradient, as shown in formula (18).

$$\nabla_{\theta_1} J (\phi) = E_s \left[ \nabla_{\theta_1} \nabla_{\phi} Q_{\theta_1} (s_t, a_t) |_{a = \pi_{\phi} (s_t)} \nabla_{\pi_{\phi}} (s_t) \right]$$

$$\approx \frac{1}{N} \sum_{j=1}^{N} \nabla_{\theta_1} Q_{\theta_1} (s_t, a_t) |_{a = \pi_{\phi} (s_t)} \nabla_{\pi_{\phi}} (s_t) \approx \frac{1}{N} \sum_{j=1}^{N} \nabla_{\theta_1} Q_{\theta_1} (s_t, a_t) |_{a = \pi_{\phi} (s_t)} \nabla_{\pi_{\phi}} (s_t) \approx \frac{1}{N} \sum_{j=1}^{N} \nabla_{\theta_1} Q_{\theta_1} (s_t, a_t) |_{a = \pi_{\phi} (s_t)} \nabla_{\pi_{\phi}} (s_t) \approx \frac{1}{N} \sum_{j=1}^{N} \nabla_{\theta_1} Q_{\theta_1} (s_t, a_t) |_{a = \pi_{\phi} (s_t)} \nabla_{\pi_{\phi}} (s_t) \approx \frac{1}{N} \sum_{j=1}^{N} \nabla_{\theta_1} Q_{\theta_1} (s_t, a_t) |_{a = \pi_{\phi} (s_t)} \nabla_{\pi_{\phi}} (s_t)$$

Meanwhile, we input $(s_t, a_t) (<s_t, a_t, r_t, s_{t+1}>)$ into two Eval Critic networks respectively to calculate two Q values, recorded as $Q_{\theta_1} (s_t, a_t)$ and $Q_{\theta_2} (s_t, a_t)$. We update two Eval Critic networks’ parameters by minimizing the two networks’ loss functions based on TD error, as shown in formula (19). Here, $y_t$ is defined in (20), where $r_t$ comes from the input $<s_t, a_t, r_t, s_{t+1}>$. $\gamma$ is the discount factor, $Q_{\theta_1} (s_{t+1}, \tilde{a})$ and $Q_{\theta_2} (s_{t+1}, \tilde{a})$ are Q values computed by two Target Critic networks upon the next state $s_{t+1}$ and the action $\tilde{a}$. Here, $\tilde{a}$ is the action generated by the Target Actor network, as shown in formula (21). We add a clipped random noise to the action for achieving the smoother state-action value estimation.

$$Loss(\theta_1) = \frac{1}{N} \sum_{t=1}^{N} (y_t - Q_{\theta_1} (s_t, a_t))^2$$

$$Loss(\theta_2) = \frac{1}{N} \sum_{t=1}^{N} (y_t - Q_{\theta_2} (s_t, a_t))^2$$

(19)
TABLE 2. Variable description.

| Variable | Description |
|----------|-------------|
| \(<s_t, a_t, r_t, s_{t+1}>\) | A transition obtained by the agent during the observation process, where \(s_t\) means the current state, \(a_t\) means the action of time slot \(t\), \(r_t\) means the immediate reward of time slot \(t\), \(s_{t+1}\) means the next state. |
| \(\pi_g(s_t)\) | An Eval Actor network used to generate the action upon the state \(s_t\), where \(\phi\) means the parameters of the Eval Actor network. |
| \(\pi_e(s_{t+1})\) | A Target Actor network used to generate the action upon the state \(s_{t+1}\), where \(\phi'\) means the parameters of the Target Actor network. |
| \(Q_g(s_t, a_t)\) | An Eval Critic network used to calculate the Q value upon the state \(s_t\) and the action \(a_t\), where \(\theta_i\) means the parameters of the Eval Critic network \(i\). |
| \(Q_e(s_t, \pi_e(s_t))\) | An Eval Critic network used to calculate the Q value upon the state \(s_t\) and the action \(\pi_e(s_t)\), where \(\theta_i\) means the parameters of the Eval Critic network \(i\). |
| \(Q_e(s_{t+1}, \tilde{a})\) | A Target Critic network used to calculate the Q value upon the state \(s_t\) and the action \(\tilde{a}\), where \(\theta'_i\) means the parameters of the Target Critic network \(i\). |

\[
y_t = r_t + \gamma \min(Q_{\theta_1}'(s_{t+1}, \tilde{a}), Q_{\theta_2}'(s_{t+1}, \tilde{a}))
\]
\[
\tilde{a} = \pi_{\phi'}(s_{t+1}) + \text{clip}(N(0, \delta'), -c, c)
\]

As mentioned above, the three networks’ parameters in TargetNet are copied from the corresponding networks in EvalNet by using soft updates, the formula are shown in (22), where \(\sigma\) is the weight.

\[
\begin{align*}
\theta_1' & \leftarrow \sigma \theta_1 + (1 - \sigma) \theta_1' & \theta_2' & \leftarrow \sigma \theta_2 + (1 - \sigma) \theta_2' \\
\phi' & \leftarrow \sigma \phi + (1 - \sigma) \phi'
\end{align*}
\]

VI. EXPERIMENTAL SETUP

A. DATASET

We use a public dataset from a well-known DSP company named iPinYou, which comprises logs of impressions, bids, clicks, and final conversions. For each impression, the iPinYou DSP bids with a fixed price of 300, competing with bidding prices from other DSPs. Then, the ADX uses the generalized second pricing (GSP) mechanism to determine the winning advertiser and charges the winner according to the second-highest bidding price that it received. In the dataset, all numbers related to price use the RMB currency, and the unit is \(10^{-3}\) Chinese FEN. The impression and click logs provide the feature information of winning impressions (i.e., ad slots, IP addresses, browser types, etc.), paying prices, and user feedback (e.g., clicks). More details of the dataset can be found in [28]. Our experiments only use seven days’ data (from 2013/06/06 to 2013/06/12) of two advertisers to construct two datasets, recorded as 1458 and 3427.

First, we build the CTR prediction model for each advertiser using a widely used factorization machine (FM) model [7]. The aggregated data for the first five days is employed as the training set, and the aggregated data of the last two days is used to the testing set. We evaluate the two CTR estimators in terms of AUC values, shown in Fig. 6. The results show that the performance of the two CTR estimators is not very satisfactory. The reason is that the click number is tiny, which leads to the imbalance of positive and negative samples in the training set. Although inaccurate CTR estimators may lead to improper bidding decisions, we still use the two FM-based estimators to evaluate each auctioned impression’s value since this paper’s focus is on the bidding strategy, not the CTR estimation.

In RTB, we usually regard a single day as an ad delivery period. In our experiments, we use the data of 2013/06/11 as the training set and the data of 2013/06/12 as the testing set to evaluate different bidding strategies’ performance. Furthermore, we regard a day as an episode and divide it into 24 time slots. Here, the training set means the historical delivery period, and the testing set means the new delivery period. Note that we use the winning impressions as the received impressions in our experiments, ignoring the losing impressions, because the losing impressions have no paying prices and user feedback. Therefore, the number of impressions in our experiments is far less than the actual number in the iPinYou dataset. Table 3 gives a description of the six statistics. The descriptive statistics of the two datasets are shown in Table 4, and Fig. 7 shows the statistics of each time slot for the 6th and 7th day. We can observe that there are significant gaps between the two days at many time slots. As described in [12], the RTB environment is continuously dynamic and challenging to predict.
B. BASELINE BIDDING STRATEGIES

In this subsection, we introduce some representative bidding strategies as baselines.

- **LIN**: The linear bidding strategy is defined as (4).
- **ORTB**: The non-linear bidding strategy is defined as (5).
- **HB**: The heuristic bidding strategy is defined as (6).
- **RLB**: The bidding strategy is learned based on a model-based RL framework proposed in [12], which can directly select an optimal bidding price for each impression.
- **DRLB**: The bidding strategy is learned based on a model-free RL framework proposed in [15]; primarily, it uses the DQN algorithm to train the optimal action policy that can help choose the regulating values for adjusting each time slot’s bidding factor. Here, \( \lambda(0) \) is determined by HB, as shown the formula (23).

\[
\lambda(0) = \frac{\text{base\_bid}^*}{\text{avg\_pctr}} \quad (23)
\]

- **FAB**: The bidding strategy is described in Algorithm 1, with the reward function \( reward_I(t) \) defined in (16). Note that FAB is recorded as FAB-I in section VII-B and VII-C.

C. HYPER-PARAMETERS SETTING

In the FAB framework, we adopt the fully-connected neural networks to construct both Actor and Critic networks. We use Adam [29] to learn the parameters of neural networks with the learning rate of 3e-4 for them, respectively. The Actor and Critic networks’ architecture is illustrated in Fig. 8. All hidden layer nodes use ReLU [30] as their activation functions. The Actor network’s final output node uses Tanh as its activation function to bound the output bidding factor. The Actor and Critic networks both have two hidden layers with 128 and 64 nodes. In our experiments, we apply a useful technology – batch normalization [31], which is applied to the input layer. In our scheme, the Actor network’s input layer is the state vector, and the Critic network’s input layer is the state vector and action value. Some hyper-parameters settings in our experiments are given in Table 5.
TABLE 5. Hyper-parameter implementation.

| Para  | Setting | Description                      |
|-------|---------|----------------------------------|
| $\gamma$ | 1       | The discount factor for MDP in (10) |
| $\delta$ | 0.1     | The noise in (15)                 |
| $\varphi$ | 1000    | The scaling factor for reward in (17) |
| $\delta'$ | 0.2     | The noise in (21)                 |
| $c$   | 0.5     | The parameter in (21)             |
| $k$   | 3       | The update frequency for soft updates |
| $\sigma$ | 0.0005 | The parameter in (22)             |
| $N$   | 32      | Mini-batch size in Algorithm 1    |
| $M$   | $10^6/000$ | The replay memory size in Algorithm 1 |

As shown in Table 5, $\gamma$ is the discount factor for MDP. In the RTB environment, the MDP is an episodic process, so the cumulated reward is the sum of the following time slots’ immediate rewards. Thus, we set $\gamma = 1$ in our experiments. Here, $\delta, \delta'$ and $c$ are parameters in TD3, so we set them according to the typical TD3 model’s setting [16]. And other parameters, i.e., $\varphi, k, \sigma, N,$ and $M$, are set based on the experience by using grid search.

VII. EXPERIMENTAL RESULTS
In this section, we first compare the performance of several representative bidding strategies with ours, in terms of the number of clicks and CPC (cost per click). Then, we discuss the effects of different reward functions on the performance of the bidding strategy. Finally, we analyze the convergence of the policy-based RL algorithms with different reward functions.

A. PERFORMANCE COMPARISON
In the first set of experiments, we evaluate the performance of six typical bidding strategies. Among them, LIN, ORTB, and HB are static bidding strategies, while RLB, DRLB, and FAB are dynamic bidding strategies based on RL. We set the daily budget of $16,000,000(1/1), 8,000,000(1/2), 4,000,000(1/4),$ and $2,000,000(1/8)$ to evaluate the adaptability of all bidding strategies to budget changes, where $16,000,000$ is about half of the total cost of buying all impressions on the testing set. There are eight scenarios in the first set of experiments considering four budgets and two datasets. The click numbers and the CPC values of these bidding strategies are demonstrated in Table 6 and Table 7. Based on the two tables, we first analyze the performance of these six bidding strategies in general. Then, we discuss each strategy’s performance in detail according to each strategy’s winning impression distribution.

The performance of FAB is the best compared with the baseline strategies. Specifically, it gets the most clicks under six scenarios and the lowest CPC values under four scenarios. We also observe that all bidding strategies’ click numbers decrease with the budgets shrinking on two datasets. Because of the lower the budget, the fewer impressions are purchased, resulting in fewer clicks. On the other hand, CPC values show a decreasing trend, which illustrates that the budget’s cost efficiency is improved when the budgets decrease. We analyze that most of the purchased impressions are invalid since only a tiny number of impressions will be clicked by users. Therefore, the budget reduction can reduce the number of purchase invalid impressions, thus effectively avoiding budget waste.

Next, we will discuss each static bidding strategy’s performance in detail. First, we analyze the performance of three static bidding strategies. Among them, the overall performance of ORTB is the worst, followed by LIN, and HB is the best. Because the bidding functions’ parameters in LIN and ORTB are obtained based on the historical training set, each impression’s bidding price is only related to its predicted CTR. Therefore, their performance is not good enough when there are significant gaps between the distributions of impressions or the market competition models on the testing set and training set. The same problem also exists in HB, although its performance is the best. HB cannot adjust each impression’s

TABLE 6. Performance comparison on two datasets.

|        | 1/1 | 1/2 | 1/4 | 1/8 | 1/1 | 1/2 | 1/4 | 1/8 |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|
| LIN    | 218 | 166 | 88  | 46  | 186 | 147 | 84  | 48  |
| ORTB   | 233 | 166 | 89  | 51  | 196 | 121 | 72  | 47  |
| HB     | 270 | 172 | 102 | 45  | 233 | 156 | 91  | 75  |
| RLB    | 261 | 163 | 90  | 51  | 215 | 130 | 80  | 45  |
| DRLB   | 270 | 150 | 81  | 45  | 202 | 120 | 93  | 46  |
| FAB    | 276 | 172 | 104 | 49  | 235 | 159 | 94  | 71  |

TABLE 7. The CPC values on two datasets.

|        | 1/1 | 1/2 | 1/4 | 1/8 | 1/1 | 1/2 | 1/4 | 1/8 |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|
| LIN    | 49151 | 48192 | 43454 | 34478 | 56041 | 54421 | 47618 | 41665 |
| ORTB   | 50444 | 48192 | 44943 | 39215 | 64575 | 66115 | 55554 | 42553 |
| HB     | 59259 | 46511 | 39215 | 44444 | 67249 | 51282 | 43956 | 24722 |
| RLB    | 61303 | 49080 | 44444 | 39216 | 74419 | 61539 | 59000 | 44445 |
| DRLB   | 59259 | 53333 | 49383 | 44446 | 79208 | 66667 | 43011 | 43479 |
| FAB    | 57971 | 46511 | 38461 | 39529 | 68085 | 50315 | 42553 | 25423 |
bidding price according to the dynamic RTB environment, but it has good adaptability to budget changes. In a new ad delivery period, the base bid price is the optimal one that maximizes the total clicks on the historical training set under the given budget. Therefore, the optimal base bid prices decrease with the budgets shrinking, as shown in Fig. 9. When the budget is very inadequate (for example, budget = 2,000,000), each impression’s bidding price is reduced to a low value. So, HB can capture more cost-effective impressions than LIN and ORTB.

Fig.10 gives a detailed description of the distributions of winning impressions throughout a day. We find that when the budget is sufficient (budget = 16,000,000), both ORTB and LIN can continuously display the ads throughout a whole day. Moreover, both ORTB and LIN have significant budget surpluses at the end, representing the budgets that have not been fully used, and many impressions that may lead to clicks have been lost potentially. When the budget is equal or less than 8,000,000, the budgets in LIN and ORTB are wiped out on two datasets. For instance, on the 1458 dataset, LIN stopped bidding at time slot 20, and ORTB stopped bidding at time slot 19. We call this phenomenon as early stop. The budgets spent out prematurely can make the bidding strategy lose all impressions that could obtain clicks in the subsequent time slots, so that reducing the total clicks of the whole period.

Unlike LIN and ORTB, HB can adjust its optimal base bid price according to the given budget. In Fig.10, HB is the last to stop bidding among three bidding strategies, leading to capture the most clicks.

To sum up, LIN and ORTB have neither the adaptability to the budget changes nor the ability to adjust the bidding function according to the RTB environment dynamically. HB’s performance is better than LIN and ORTB because it can adapt to the budget changes, but it can’t adapt to the dynamic RTB environment.

Then, we analyze the performance of three RL-based bidding strategies. Fig.11 gives the distributions of winning impressions on the testing set. First, we observe that RLB can strategically adjust each impression’s bidding price throughout a whole delivery period according to the dynamic RTB environment. So, the advertisers in RLB can display their ads to users at all time slots under different budgets. Unfortunately, although RLB can make the bidding decision for each impression according to the real-time market environment, it gets fewer total clicks than HB in seven scenarios, as shown in Table 6. The main reason is that RLB assumes that the market price distribution is stable and generates the bidding price based on the market price distribution of historical data, making it insensitive to the dynamic environment. Thus, it is not easy to make accurate bidding decisions for consecutive impressions during the new ad delivery period.

DRLB gives up the idea of making a bidding decision on a single impression, but divides a day into several time slots, and adapts to the dynamic RTB environment by adjusting each time slot’s bidding factor. Unfortunately, DRLB’s experimental results reveal that its performance is inferior to HB in five scenarios. DRLB even stopped bidding earlier than HB in six settings. The reason for this result is that DRLB learns the optimal action policy based on DQN, which only supports selecting an action from a preset action space at each time slot. Unfortunately, manually setting the action space is very challenging. In fact, the elements of the action space in
DRLB are the discrete regulating values, for example, \{-8\%, -3\%, -1\%, 0\%, 1\%, 3\%, 8\%\}, which are used to adjust each time slot’s bidding factor iteratively based on the formula (3). These limited regulating values can only regulate the bidding factor in coarse-grained so that there is a large gap between the calculated bidding factor and the optimal one. Compared with RLB and DRLB, FAB achieves the most clicks under five scenarios. Fig.12 shows each time slot’s click number in FAB on two datasets, where the black dot represents the number of real clicks of each time slot. The results show that FAB is indeed able to allocate the budget more reasonably and efficiently across all available impressions during the entire period.

In summary, compared with the three static bidding strategies, RLB can adapt to the dynamic changes of the environment. Still, its performance is not ideal because of its conservative estimate of the market price distribution. Both DRLB and FAB are optimized based on the optimal base bid price of HB. However, the experimental results reveal that the optimization effect of our FAB is significantly higher than that of DRLB. It indicates that using the policy-based RL to generate the optimal bidding factor directly is superior to using the value-based RL to adjust the optimal bidding factor sequentially.

### B. Effectiveness of the Reward Functions

In the second set of experiments, we evaluate the effects of different reward functions on our bidding strategy’s performance. Our bidding strategies with three reward functions are detailed in Section VI-B. Table 8 summarizes the number of clicks obtained by three bidding strategies under different settings. The results show that FAB-I gets the most number of clicks, followed by FAB-II, and the click number of FAB-III is far less than the former two. In Fig.13, we display the actions’ distributions generated by three different bidding strategies on the testing sets. First of all, we analyze why the number of clicks of each time slot cannot be used as the immediate reward directly, like in FAB-III. Let us review the formulas (20), the first part means the immediate reward, and the second part represents the future target cumulative discounted reward (Q value). In FAB-III, the immediate reward is defined by the number of clicks of each time slot, which is much higher than the estimated Q value. So, during the training process, the bidding agent will be more affected by the immediate reward and ignore the future reward.

Then, let us discuss the problems that may occur during training in FAB-III. At the beginning of an episode, the available budget is sufficient. If the agent’s exploration actions tend to -0.99 in the first few time slots, that is, the bidding price for each impression is almost all 300 (the highest price in iPinYou). The agent will obtain all real clicks at each time slot (i.e., get the highest immediate reward), which will guide the gradient update toward the wrong direction, thus increasing the Critic network’s losses corresponding to these actions significantly. Influenced by that, the agent is challenging to reach a stable convergence. The plots in Fig.14 demonstrate this point. Then, the agent may fall into the local optimum and still generates the action value closer to -0.99 in the future time slots to maximize the immediate reward. We can find this phenomenon in Fig.15, especially on the dataset (3427).
Although we add a fixed noise $\mathcal{N}(0, \delta)$ to the action value, as shown in (15), it is not large enough for the agent to jump out of this trap. To help the bidding agent jump out of the trap of local optimum, we introduce a scalar factor ($\psi = 1000$) to numerically reduce the immediate reward close to estimated $Q$ value. Thus, the experimental results show that our design is useful, especially for $\text{reward}_I$, which improves the bidding performance significantly.
To sum up, the designed reward functions can effectively guide the agent to learn the global optimal action policy by numerically scaling the immediate reward. Compared with $\text{reward}_I$, $\text{reward}_II$ can help the agent learn the optimal action policy and generate a smoother bidding factor much better, as shown in Fig.13.

C. CONVERGENCE COMPARISON

In this subsection, we analyze the convergence of FAB-I, FAB-II, and FAB-III. Under every scenario, we execute the three bidding strategies for 5000 episodes. We record the Critic network’s losses on the training set every ten episodes and calculate the average reward by evaluating the action policy’s performance on the testing set. Fig.14 and Fig. 15 depict the changing trends of the losses and the average rewards. It can be observed that FAB-I’s convergence is the best (refers to the red curves). Its loss is decreased rapidly to a low and stable value. The obtained rewards also reach the global optimum quickly.

In contrast, the losses of FAB-II and FAB-III fluctuate during the whole training process on the two datasets and do not converge to a stable value. The reason for convergence failure is that the training set contains too few positive samples. There are only 395 and 283 clicks on the 1458 and 3427 datasets. However, under the extreme imbalance of the positive and negative samples, our proposed new reward function $\text{reward}_III$ guides the learning algorithm converges quickly. In particular, we observe that the average rewards of both FAB-I and FAB-II converge to the globally optimal values rapidly in Fig.15.

Unfortunately, on the 1458 dataset, FAB-III falls into the local optimum for a long time in the early training stage. Only in the later stage will it gradually jump out of local optimum and update to the direction of global optimization. More seriously, on the 3427 dataset, FAB-III cannot jump out of local optimum during the whole training process.

The reason is that the immediate reward generated by $\text{reward}_{III}$ is much larger than the Q values generated by the Target Critical Networks, which makes it difficult to get a higher cumulative reward through the exploration action, thus helping to jump out of the trap. In conclusion, our reward function’s convergence is much better than that of the click number or the reduced click number as the reward function.

VIII. CONCLUSION

In this paper, we propose a new bidding strategy for RTB in display advertising, which maximizes the number of clicks of an ad delivery period by using an optimal bidding factor generation policy. In our bidding strategy, the bidding agent divides an ad delivery period into several time slots and adjusts each impression’s bidding price based on the optimal bidding factor of each time slot, to adapt to the highly dynamic RTB environment. In this paper, we utilize the policy-based RL (TD3) framework to learn the optimal bidding factor generation policy, to produce an optimal continuous value as the bidding factor for each time slot.

We are the first one using policy-based RL to optimize the bidding strategy. We also define a new reward function by comparing our method with the heuristic bidding (HB) in terms of budget cost and revenue of each time slot, to guide the bidding factor generation policy to converge to the global optimal efficiently. Finally, extensive experimental results on a real dataset show that our bidding strategy is the best compared with other state-of-the-art baselines. In our future work, we will first review using real clicks as the revenue in the RL framework. It is defective to use the real clicks as the basis of the reward function, because the factors of click behavior may be very complex and uncertain. How to design a more appropriate reward function will be the focus of our future work. Besides, when analyzing the experimental dataset, we note that most of the impressions are invalid and cannot bring user clicks. Namely, there are many impressions with low prices and low estimated values in RTB. Buying these low-valued impressions is a waste of budget, especially when the budget is not sufficient. Therefore, avoiding buying this kind of invalid impressions is the problem we need to solve in the future.

REFERENCES

[1] Xu, X. Shao, J. Ma, K. C. Lee, H. Qi and Q. Lu, “Lift-based bidding in ad selection,” in Proc. AAAI, Phoenix, AZ, USA, 2016, pp. 651–657.
[2] S. Yuan, J. Wang, and X. Zhao, “Real-time bidding for online advertising: measurement and analysis,” in Proc. ACM ADKDD, Chicago, IL, USA, 2013, pp. 19–27.
[3] J. Wang, W. Zhang, and S. Yuan, “Display advertising with real-time bidding (RTB) and behavioural targeting,” Found. Trends, vol. 11, pp. 297–435, 2017.
[4] J. Wang, and S. Yuan, “Real-Time bidding: a new frontier of computational advertising research,” in Proc. WSDM, Shanghai, China, 2015, pp. 415–416.
[5] P. Grigas, A. Lobos, Z. Wen, and K.-C. Lee, “Profit maximization for online advertising demand-side platforms,” in Proc. ADKDD ZZZ (ADKDD), Halifax, NS, Canada, 2017, pp. 1–7.
[6] M. Richardson, E. Dominowska, and R. Ragno, “Predicting clicks: Estimating the click-through rate for new ads,” in Proc. 16th Int. Conf. World Wide Web, Banff, AB, Canada, 2007, pp. 521–530.
[7] S. Rendle, “Factorization machines with libFM,” ACM Trans. Intell. Syst. Technol., vol. 3, no. 3, pp. 1–22, May 2012.
[8] W. Zhang, T. Du, and J. Wang, “Deep learning over multi-field categorical data: a case study on user response prediction,” in Proc. ECIR, Padua, Italy, 2016, pp. 45–57.
[9] Y. Yuan, F. Wang, J. Li, and R. Qin, “A survey on real-time bidding advertising,” in Proc. IEEE Int. Conf. Service Oper. Logistics, Inform., Qingdao, China, Oct. 2014, pp. 418–423.
[10] W. Zhang and J. Wang, “Statistical arbitrage mining for display advertising,” in Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), Sydney, NSW, Australia, 2015, pp. 1465–1474.
[11] W. Zhang, S. Yuan, and J. Wang, “Optimal real-time bidding for display advertising,” in Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), New York, NY, USA, 2014, pp. 1077–1086.
[12] H. Cai, K. Ren, W. Zhang, K. Malialis, I. Wang, Y. Yu, and D. Guo, “Real-time bidding by reinforcement learning in display advertising,” in Proc. 10th ACM Int. Conf. Web Search Data Mining (WSDM), Cambridge, U.K., 2017, pp. 661–670.
[13] R. S. Sutton, “Introduction: The challenge of reinforcement learning,” Mach. Learn., vol. 8, nos. 3–4, pp. 225–227, May 1992.
[14] P. L. David and M. K. Alan, “Artificial intelligence: Foundations of computational agents,” in Artificial Intelligence. Cambridge, U.K.: Cambridge Univ. Press, 2011.
MENGJUAN LIU received the Ph.D. degree in communication and information system from the University of Science and Technology of China, in 2007. She is currently an Associate Professor with the Department of Information and Software Engineering, University of Electronic Science and Technology of China. Her main research interests include computational advertising, data mining, and artificial intelligence.

LI JIAJING was born in Sichuan, China, in 1996. He received the B.S. degree in software engineering from the University of Electronic Science and Technology (UESTC), Chengdu, China, in 2018. He is currently pursuing the M.S. degree in software engineering with the School of Information and Software Engineering, UESTC. His research interests include machine learning and computational advertising.

ZHENGNING HU was born in 1998. He received the B.S. degree in software engineering from the University of Electronic Science and Technology (UESTC), Chengdu, China, where he is currently pursuing the Ph.D. degree in software engineering with the School of Information and Software Engineering. His research interests include computational advertising and machine learning.

JINYU LIU was born in Sichuan, China, in 1997. He received the B.S. degree in software engineering from the Chongqing University of Posts and Telecommunications (CQUT), in 2019. He is currently pursuing the M.S. degree in software engineering with the School of Information and Software Engineering, University of Electronic Science and Technology (UESTC), Chengdu, China. His research interests include machine learning and computational advertising.

XUYUN NIE received the Ph.D. degree in information security from the Graduate University of Chinese Academy of Sciences, Beijing, China, in 2007. He is currently an Associate Professor with the University of Electronic Science and Technology of China. His research interests include cryptography and information security.

M. Liu et al.: Dynamic Bidding Strategy Based on Model-Free RL in Display Advertising

[15] D. Wu, X. Chen, X. Yang, H. Wang, Q. Tan, X. Zhang, J. Xu, and K. Gai, “Budget constrained bidding by model-free reinforcement learning in display advertising,” in Proc. 27th ACM Int. Conf. Inf. Knowl. Manage., New York, NY, USA, Oct. 2018, pp. 1443–1451.

[16] S. Fujimoto, H. Van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” in Proc. ICML, Stockholm, Sweden, 2018, pp. 2587–2601.

[17] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” IEEE Trans. Neural Netw., vol. 9, no. 5, p. 1054, Sep. 1998.

[18] C. Perlich, B. Dalessandro, R. Hook, O. Stitelman, T. Raeder, and F. Provost, “Bid optimizing and inventory scoring in targeted online advertising,” in Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), Beijing, China, 2012, pp. 804–812.

[19] M. Liu, W. Yue, L. Qiu, and J. Li, “An effective budget management framework for real-time bidding in online advertising,” IEEE Access, vol. 8, pp. 131107–131118, 2020, doi: 10.1109/ACCESS.2020.2970463.

[20] M. Du, S. Redouane, V. Georgios, and S. Radu, Improving Real-Time Bidding Using a Constrained Markov Decision Process. Springer, 2017, pp. 711–726.

[21] G. Perter, “Reinforcement learning for MDPs with constraints,” in Proc. ECML, Berlin, Germany, 2006, pp. 645–653.

[22] J. Zhao, G. Qiu, and Z. Guan, “Deep reinforcement learning for sponsored search real-time bidding,” in Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, London, U.K., 2018, pp. 1021–1030.

[23] G. J. Gordon, Stable Function Approximation in Dynamic Programming. Pittsburgh, PA, USA: Carnegie Mellon Univ., 1995.

[24] R. S. Sutton, D. McAllester, and S. Singh, “Policy gradient methods for reinforcement learning with function approximation,” in Proc. Adv. Neural Inf. Process. Syst., vol. 12, 2000, pp. 1057–1063.

[25] T. P. Lillicpap, J. H. Jonathan, A. Pritzel, and N. Heess, “Continuous control with deep reinforcement learning,” in Proc. ICLR, San Juan, Puerto Rico, 2016. [Online]. Available: https://arxiv.org/abs/1509.02971

[26] J. Peters and S. Schaal, “Natural actor-critic,” Neurocomputing, vol. 71, nos. 7–9, pp. 1180–1190, Mar. 2008.

[27] G. Ivo, B. Lucian, L. A. D. Gabriel, and B. Robert, “A survey of actor-critic reinforcement learning: Standard and natural policy gradients,” IEEE Trans. Syst., Man, Cybern., C (Appl. Rev.), vol. 42, no. 6, pp. 1291–1307, Nov. 2012.

[28] H. Liao, L. Peng, Z. Liu, and X. Shen, “iPinYou global RTB bidding algorithm competition dataset,” in Proc. 8th Int. Workshop Data Mining Online Advertising, New York, NY, USA, 2014, pp. 1–6.

[29] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. ICLR, San Diego, CA, United states, 2015.

[30] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” J. Mach. Learn. Res., vol. 14, pp. 315–323, Apr. 2011.

[31] S. Loffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in Proc. ICML, Lile, France, 2015, pp. 448–456.