Optimal Bidding Strategy without Exploration in Real-time Bidding

Aritra Ghosh [*]

arighosh@cs.umass.edu

Saayan Mitra [†]

smitra@adobe.com

Somdeb Sarkhel [†]

sarkhel@adobe.com

Viswanathan Swaminathan [†]

vishy@adobe.com

Abstract

Maximizing utility with a budget constraint is the primary goal for advertisers in real-time bidding (RTB) systems. The policy maximizing the utility is referred to as the optimal bidding strategy. Earlier works on optimal bidding strategy apply model-based batch reinforcement learning methods which can not generalize to unknown budget and time constraint. Further, the advertiser observes a censored market price which makes direct evaluation infeasible on batch test datasets. Previous works ignore the losing auctions to alleviate the difficulty with censored states; thus significantly modifying the test distribution. We address the challenge of lacking a clear evaluation procedure as well as the error propagated through batch reinforcement learning methods in RTB systems. We exploit two conditional independence structures in the sequential bidding process that allow us to propose a novel practical framework using the maximum entropy principle to imitate the behavior of the true distribution observed in real-time traffic. Moreover, the framework allows us to train a model that can generalize to the unseen budget conditions than limit only to those observed in history. We compare our methods on two real-world RTB datasets with several baselines and demonstrate significantly improved performance under various budget settings.

1 Introduction

Real-time bidding (RTB) has become the dominant mechanism for online advertising in current times. One of the key challenges for the advertiser in the RTB system is to devise a sequential bidding strategy for bid requests to maximize some utility (e.g., impression, clicks, etc.) under some budget constraint. Under repeated auctions with budget constraints, bidding the true value is not the optimal action in second-price auctions. We refer to the policy maximizing expected utility under a budget constraint as the *optimal bidding strategy.*

In recent times, reinforcement learning (RL) achieves almost human-level performance in many games and control problems [18]. Reinforcement learn-

---

[*]UMass, Amherst, MA, USA

[†]Adobe Research, San Jose, CA, USA

ing to bid (RLB), a model-based RL approach for optimal bidding in RTB has been proposed recently [5]. Although RLB improves upon existing methods, RLB suffers from scalability and efficiency issues in RTB systems using a misspecified model learned from inadequate interactions. A better alternative is to use a model-free RL algorithm which requires a large number of interactions with the environment (users, other participants in RTB) to learn the optimal strategy. However, in real-time bidding systems, learning the optimal bidding strategy is often a batch process with limited opportunity to interact with the environment. From historical interactions, the advertiser needs to learn the optimal action (bidding price). In reinforcement learning literature, learning optimal policy from historical interactions without exploration is often termed as *batch reinforcement learning* [16].

Although model-free online RL algorithms are conceptually appealing, real-time bidding systems often have the choice of batch reinforcement learning. Most early approaches of optimal bidding strategy can be classified as some form of batch RL [5, 2, 25, 19]. However, the single drawback of any batch RL algorithm is the limited generalization ability to new state space due to extrapolation error as shown lately in [8]. Without exploration, often it is not feasible to extrapolate under new budget constraints. Besides, a clear evaluation procedure is lacking in the literature of online advertising due to the nature of the batch data. Without interacting with the environment, it is not viable to evaluate reward for any action other than the one taken in the test batch dataset. Further, in the context of RTB, the behavior of the market is censored when the advertiser loses an auction; hence the advertiser often observes censored states which form the test dataset. Early methods of optimal bidding strategy remove the censored/unobserved part of the test dataset for evaluation leading to a different distribution than observed in real-time environments. The dataset shift in the true distribution and the truncated train-test distribution adopted in previous research makes such models unsuitable for directly applying to the real-world traffic.

The batch framework is particularly important as the dataset is usually collected from an (unknown) off-

policy strategy in the RTB system. Further, conducting real-traffic A/B testing is not often feasible and cost-effective in practice. We address the deficiencies in the batch RL methods such as lack of a clear evaluation procedure as well as the error propagation through the training methods in RTB systems. Using two conditional independence structures in the sequential bidding process, we propose a novel framework applying the maximum entropy principle to imitate the behavior of the true distribution observed in real-time traffic. The simulated environment allows accurate evaluation of any model in the RTB system without the dataset shift. Further, the framework enables training any model-free RL algorithm that can generalize to an unseen budget and time constraints, beyond the states observed in historical interactions. We compare our methods on real-world public RTB data-sets with several baselines and show that our framework significantly improves generalization performance under various budget settings.

2 Background and Related Work

Reinforcement Learning. A Markov Decision Process (MDP) $\mathcal{M}$ is represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ which consists of a set of states $s \in \mathcal{S}$, a set of actions $a \in \mathcal{A}$, a transition function $\mathcal{T}(s, a, s') = \mathrm{P}(s_{i+1} = s'|s_i = s, a_i = a)$, a reward function $\mathcal{R}(s, a)$, and, a discount parameter γ [21]. s_i, a_i, r_i are the state, action, and, reward at time step i respectively. A policy is defined as $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ representing the conditional distribution over actions given the state that the agent follows. The goal of the agent is to find the policy π that maximizes the expected discounted reward over the episode [21]. State-Action value function is the expected reward that can be obtained following policy π starting from a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$:

$$Q^\pi(s, a) \quad = \mathbf{E}[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau | s_t = s, a_t = a, \pi]$$

The optimal value function for the state-action pair, $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, satisfies the Bellman Optimality equation [21], $Q^*(s, a) = \mathbf{E}[\mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s, a, s') \max_{a'} Q^*(s', a')]$. Similarly, value function from a state $\mathbf{s} \in \mathcal{S}$ is the expected reward from that state following policy π, $V^\pi(s) = \mathbf{E}_{a \sim \pi} Q^\pi(s, a)$.

Batch reinforcement learning. In batch RL, the agent does not have opportunities to interact with the environment [16]. If the policy used to collect the experiences is known, we can use the deep Q-network (DQN) with importance sampling (IS) to learn from the batch experiences (dataset) [18]. However, DQN with IS suffers from high extrapolation error [8]. Further, in an RTB system, the policy that gathered the historical

data is usually unknown. A general approach to learn from batch data is to learn fitted Q-iteration (possibly with a deep network) [20, 3]. Nevertheless, without exploration, the performance of neural fitted Q iteration is limited to the experiences gathered.

Real-time Bidding. In RTB systems, ad display opportunities are traded using second-price auctions in real-time from the publishers (sellers) to the advertisers (buyers) through the ad-exchange. When a user visits a publisher's page, the supply-side platform, acting on behalf of the publisher, requests the ad-exchange for advertisements to fill up the vacant slots. The ad-exchange, in turn, announces the request to the demand-side platforms (DSP) who represent the advertisers. Subsequent to getting a bid request, DSP chooses one among several ads to bid and participates in the auction. The ad exchange picks the highest bidder and the winning DSP pays the second price (or market price). DSP observes the market price if it wins the auction; otherwise, in case of losing the auction, only the lower bound on the market price is known. This mixture of observed and partially observed data is known as *censored data*. The purpose of DSP is to maximize some utility (impression, click, or, conversion) under some budget and time constraint for each of its advertisers. In the rest of the paper, we use the term advertiser and DSP interchangeably for simplicity.

We represent a bid request as $\mathbf{x}$, market price as $\mathbf{w}$ (can be unknown), budget left as $\mathbf{b}$, and, time left as $\mathbf{t}$. Advertiser bids $\mathbf{a}$ for the bid request $\mathbf{x}$ with cost $\mathbf{c}$ ($\mathbf{w}$ when the advertiser wins the auction and 0 otherwise). The advertiser observes a utility reward $\mathbf{r}$ (impression, click, or, conversion). The objective of the advertiser is to maximize the total reward obtained given the budget and time constraint with a policy $\pi([\mathbf{x}, \mathbf{b}, \mathbf{t}], \cdot)$:

$$\max_{\mathbf{a} \sim \pi} \sum_1^{\mathbf{t}} \mathbf{E}_\mathbf{x}[\mathbf{r}|\mathbf{x}], \quad \text{such that} \quad \sum_1^{\mathbf{t}} \mathbf{E}_\mathbf{x}[\mathbf{c}|\mathbf{a}, \mathbf{x}] \le \mathbf{b}.$$

Optimal Bidding Strategy. To compute the optimal bidding strategy, the advertiser needs to estimate the expected utility and the expected cost. Previous research extensively studied utility estimation problems (such as click-through rate estimation) [17] and the bid landscape forecasting problem for computing expected cost [24, 9]. In real-world applications, due to simplicity and scalability, the bidding systems usually employ a simple linear bidding strategy [19] where the advertiser bids proportional to their expected value (utility). Early attempts for optimal bidding strategy include model-based MDP and partially observable MDP formulations [2, 25] which works in the context of sponsored ads. In [5], the authors proposed the RLB model to solve the Bellman equation on a simplified model. Solving the

Bellman equation has an attractive property of computing the optimal strategy when the model is correct. However, to tackle large continuous state and action space, the RLB model assumes the market price and the winning rate are independent of the bid requests. Moreover, inferring the optimal action has $\mathcal{O}(k)$ time complexity and $\mathcal{O}(T^2 k)$ memory complexity for each bid request (T, k is the maximum number of time steps left and the number of bid price respectively). This is unacceptable in the real world situation. RLB proposed to use a segmented time window with neural network approximation leading to further approximation besides the model simplification. Besides, model-based batch reinforcement learning algorithms do not have the opportunity to explore with the real environment making them prone to extrapolation errors [8].

In this paper, we frame the problem as learning the optimal policy from batch datasets (with censored states) without the choice of explorations and access to the policy used to collect the dataset. The second problem we consider is to learn a correct evaluation procedure from such offline datasets with censored states. Note that, for losing an auction, we only have a lower bound on the market price; consequently, it is not possible to accurately evaluate the reward when evaluating the batch-test dataset. Earlier research excludes all the bid requests where the market price is not known leading to a distributional shift.

3 Problem Definition

We define the MDP $\mathcal{M}$ for the optimal bidding strategy problem $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ with policy π as follows:

i) We represent the **state** $\mathbf{s} \in \mathcal{S}$ at time step i as $\mathbf{s}_i = [\mathbf{x}_i \oplus \mathbf{w}_i \oplus \mathbf{b}_i \oplus \mathbf{t}_i]$ where $\mathbf{x}_i, \mathbf{b}_i, \mathbf{t}_i$ are the bid request, budget left, and, time left respectively and $\oplus$ represents the concatenation operator. $\mathbf{w}_i$ is the market price of auction at time step i and observed (might be censored) at time step $i + 1$. Thus we use another notation $\mathbf{s}_i^O = [\mathbf{x}_i \oplus \mathbf{b}_i \oplus \mathbf{t}_i]$ to represent the partially observed state without the market price $\mathbf{w}_i$ at time i. The market price $\mathbf{w}_i$ represent sufficient statistics about the market (consisting of many DSP) behavior overall. **ii) Action** $\mathbf{a}_i \in \mathcal{A}$ is the bid price for bid request $\mathbf{x}_i$ at time step i. **iii) Reward** $\mathbf{r}_i = imp_i$ or $click_i$ is the user/market response for bid request $\mathbf{x}_i$ and bid price $\mathbf{a}_i$; when impression is the utility, reward is 1 if $\mathbf{a}_i > \mathbf{w}_i$ and 0 otherwise. When the utility of interest is click, besides the impression, the user needs to click the ad for advertiser to get reward 1. **iv) Discount Parameter** γ is usually set as 1 in RTB as the objective is to maximize total rewards with budget constraints. **v) Policy** $\pi(\mathbf{s}^O = [\mathbf{x} \oplus \mathbf{b} \oplus \mathbf{t}], \mathbf{a})$ is the probability of bidding $\mathbf{a}$ when the bid request, time left, budget left

are $\mathbf{x}, \mathbf{t}, \mathbf{b}$ respectively. **vi)** We decompose the state representation into the **market-specific state** ($\mathbf{s}_i^M = [\mathbf{x}_i \oplus \mathbf{w}_i]$) and the **advertiser state** ($\mathbf{s}_i^A = [\mathbf{b}_i \oplus \mathbf{t}_i]$). The **transition function** is:

$$\mathcal{T}(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_{i+1}) = P(\mathbf{s}_{i+1} = [\mathbf{s}_{i+1}^M \oplus \mathbf{s}_{i+1}^A] \mid \mathbf{s}_i, \mathbf{a}_i).$$

We denote the batch dataset as a collection of trajectories $\mathcal{D} = \{\tau_i\}_{i=1}^n$ where trajectory $\tau_i = (\mathbf{s}_1, \mathbf{a}_1, \mathbf{r}_1, \mathbf{s}_2, \mathbf{a}_2, \mathbf{r}_2, \cdots, \mathbf{s}_T, \mathbf{a}_T, \mathbf{r}_T)$ is the sequence of state, action, reward in the ith episode. We need to learn the optimal policy $\pi^*(\mathbf{s}^O = [\mathbf{x}_i \oplus \mathbf{b}_i \oplus \mathbf{t}_i], \cdot)$ from the batch dataset; further we need to devise an evaluation framework that represents the actual test distribution advertiser observes. Moreover, we are interested in policy π that can generalize to any advertiser state space ($\mathbf{s}^A = [\mathbf{b} \oplus \mathbf{t}]$).

Remark 1. Before delving into methodology, we state two important conditional independence properties that we use throughout our formulation. **i)** Current market-specific state and advertiser state are conditionally independent given the last state and the last action. Thus the transition function simplifies to:

$$(3.1) \qquad P(\mathbf{s}_{i+1}|\mathbf{s}_i, \mathbf{a}_i) = P(\mathbf{s}_{i+1}^M|\mathbf{s}_i, \mathbf{a}_i) \cdot P(\mathbf{s}_{i+1}^A|\mathbf{s}_i, \mathbf{a}_i)$$

We can consider the impact of a bidding action taken by the advertiser is negligible on the decision of future bid requests generated by large number of users as well as the bidding behavior of many other DSPs and advertisers. **ii)** Market-specific states at any two time points $i \neq j$ are independent of each other.

$$(3.2) \quad P(\mathbf{s}_i^M = s_i, \mathbf{s}_j^M = s_j) = P(\mathbf{s}_i^M = s_i)P(\mathbf{s}_j^M = s_j)$$

We can assume numerous bid requests generated from many users at different timesteps are independent of each other.

4 Methodology

In Figure 1(a), we show the graphical model for the sequence of states (market and advertiser), actions and rewards. The key idea is that we need to imitate the behavior of the environment from the batch dataset to evaluate the correct metric for any policy and to successfully learn the optimal policy starting with any advertiser state ($\mathbf{s}^A = [\mathbf{b} \oplus \mathbf{t}]$).

The two conditional independence in Remark 1 implies that the next state of the advertiser depends on the past state, and, action while the market-specific states are mutually independent at any two timesteps. Thus, for the agent to explore in a simulated environment, we only need the behavior of market-specific state trajectory ($\mathbf{s}_1^M \rightarrow \mathbf{s}_2^M \cdots \rightarrow \mathbf{s}_n^M$) and the reward distribution
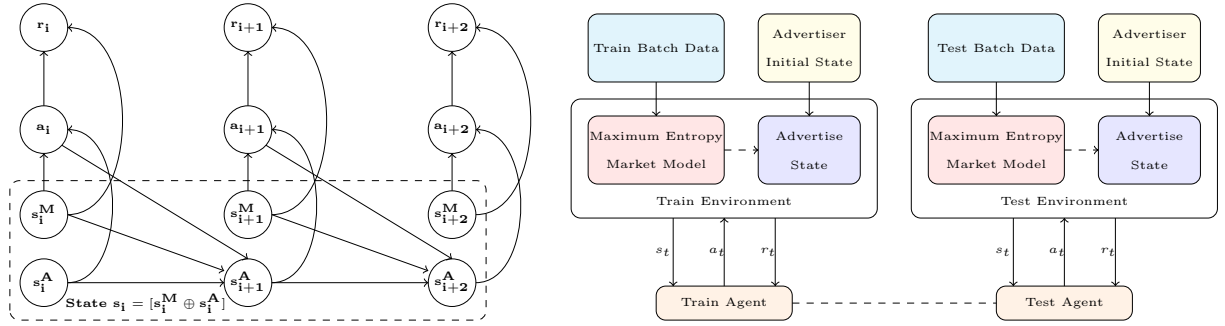
Figure 1: a) Graphical model of the Bidding System (left). b) Training and Evaluation Framework (right). Agent optimized using train environment is tested on test environment.

$\mathbf{p}_r(\mathbf{r}; \mathbf{a}, \mathbf{s}^M)$. Using these models, the agent can instantiate with any starting budget $\mathbf{b}$ and time constraints $\mathbf{t}$ (that constitutes advertiser state) in the environment and explore to learn the optimal bidding strategy that can generalize to unknown advertiser state space. Further, we can evaluate any bidding strategy under any advertiser state space if we have access to the model that can imitate the market behavior and reward distributions. Thus the problem reduces to learning a market model that can sample close to the distribution used to generate the batch dataset. In Figure 1(b), we show a schematic diagram of the training and testing frameworks that exploits the conditional structure on the graphical model to simulate the training and testing environment; details of which follow next. Finally, we note that the market (users, other bidders) behaves rationally and maximize their long-term reward; often the user clicks ads only if it is relevant to them and other bidders optimize their cost rationally [6]. We explicitly state this assumption.

Assumption 1. The market optimizes some unknown cost function and is near-optimal in making decisions.

4.1 Maximum Entropy Market Model In inverse reinforcement learning (IRL), the task is to learn the latent cost function as well as the optimal policy from expert's trajectories [1, 27]. To imitate the behavior of the market, we need to learn the state dynamics and reward distribution of the market. The (near) optimal trajectories from market are $\{\tau_i^M\}_{i=1}^n$ where $\tau_i^M = (\mathbf{s}_1^M, \mathbf{r}_1, \mathbf{s}_2^M, \mathbf{r}_2, \cdots, \mathbf{s}_T^M, \mathbf{r}_T)$, $\mathbf{s}^M$ is the the market-specific state and $\mathbf{r}_i$ is the observed reward by the advertiser. The state consists of bid request $\mathbf{x}_i$ in addition to the (censored) market price $\mathbf{w}_i$ while the reward $\mathbf{r}_i$ is the observed ad impression/click.

For simplicity of derivation, we introduce a separate notation for the maximum entropy market model with a new MDP $\mathcal{M}^M$ along with a modified state and action

space. For the market MDP $\mathcal{M}^M$, we denote $\mathbf{x}_i$ as the new **market state** and $\mathbf{a}_i^M = [\mathbf{w}_i \oplus \mathbf{r}_i]$ as the **market action**. The decomposition follows our intuition that the market action (market price, click behavior of user) is (near) optimal. Finally, note that we do not observe $\mathbf{w}_i, \mathbf{r}_i$ always; in case of losing auction only a lower bound of the market price $\mathbf{w}_i$ is known which we handle later using censored regression-based approaches [24]. Under the new notation, we need to model the market trajectories $\{\tau_i^M\}_1^n$ where $\tau_i^M = (\mathbf{x}_1, \mathbf{a}_1^M, \cdots, \mathbf{x}_T, \mathbf{a}_T^M)$.

We formulate the market trajectory distribution as, $\mathrm{P}_\phi(\tau^M) = \frac{\exp(-c_\phi(\tau^M))}{Z}$ where $c_\phi(\tau^M) = \sum_{i=1}^n c_\phi(\mathbf{x}_i, \mathbf{a}_i^M)$ is the latent cost function the market is optimizing with parameter ϕ, and, Z is the partition function. This form follows our assumption that market is near-optimal as trajectories with lower cost are exponentially more probable than the trajectories with higher cost. Further, the model prefers all the trajectories with the same latent cost functions, equally; thus maximizing the entropy of the distribution [27]. We maximize log-likelihood to optimize ϕ such that historical market trajectories have a low-cost value:

$$\ell(\phi) = \mathbf{E}_{\tau \sim \mathcal{M}^M} \log \mathrm{P}_\phi(\tau) = \mathbf{E}_{\tau \sim \mathcal{M}^M}[-c_\phi(\tau) - \log Z]$$

We further simplify with Eq. 3.2 that $(\mathbf{x}_i, \mathbf{a}_i^M) \perp (\mathbf{x}_j, \mathbf{a}_j^M)$ (note we decompose the market-specific state $\mathbf{s}^M$ into $\mathbf{x}, \mathbf{w}$ in the market MDP formulation). Thus a random state-action pair of the market trajectory will have the same expected values as the whole trajectory. Furthermore, we decompose the cost function as $c_\phi(\mathbf{x}, \mathbf{a}^M) = c_\phi^1(\mathbf{x}) + c_\phi^2(\mathbf{a}^M; \mathbf{x})$.

$$
\begin{aligned}
\ell(\phi) &= \mathbf{E}_{(\mathbf{x}, \mathbf{a}^M) \sim \mathbf{s}^M} \log \mathrm{P}_\phi(\mathbf{x}, \mathbf{a}^M) \\
(4.3) &= \underbrace{\mathbf{E}_{\mathbf{x} \sim \mathbf{s}^M} \log \mathrm{P}_\phi(\mathbf{x})}_{\text{Market State Obj.}} + \underbrace{\mathbf{E}_{\mathbf{a}^M \sim \mathbf{s}^M | \mathbf{x}} \log \mathrm{P}_\phi(\mathbf{a}^M | \mathbf{x})}_{\text{Market Action Obj.}} \\
&= \qquad \ell_1(\phi) \qquad + \qquad \ell_2(\phi)
\end{aligned}
$$

Market State Model. We start with the first term $\ell_1(\phi)$ for the state $\mathbf{x}$ of the MDP $\mathcal{M}^M$.

$$\begin{aligned}\ell_1(\phi) &= \mathbf{E}_{\mathbf{x}\sim P_r}[-c_\phi^1(\mathbf{x})] - \log Z_1 \\ &= \mathbf{E}_{\mathbf{x}\sim P_r}[-c_\phi^1(\mathbf{x})] - \log[\mathbf{E}_{\mathbf{x}\sim\mathbf{q}_\theta}\frac{\exp(-c_\phi^1(\mathbf{x}))}{\mathbf{q}_\theta(\mathbf{x})}]\end{aligned}$$

where we denote P_r as the true distribution of bid requests $\mathbf{x}$. Computing partition function $Z_1 = \int_x \exp(-c_\phi^1(\mathbf{x}))dx$ is intractable; thus we compute Z_1 with samples from a parameterized distribution $\mathbf{q}_\theta$ with parameter θ and multiply with importance sampling weights to get a consistent estimate of Z_1. We use a single Monte-carlo sample to compute the unbiased estimate of the partition function Z_1. The objective becomes:

$$\begin{aligned}\ell_1(\phi) &\simeq \mathbf{E}_{\mathbf{x}\sim P_r}[-c_\phi^1(\mathbf{x})] - \log[\frac{\exp(-c_\phi^1(\mathbf{x}))}{\mathbf{q}_\theta(\mathbf{x})}]_{\mathbf{x}\sim\mathbf{q}_\theta} \\ &= \mathbf{E}_{\mathbf{x}\sim P_r}[-c_\phi^1(\mathbf{x})] - [-c_\phi^1(\mathbf{x}))]_{\mathbf{x}\sim\mathbf{q}_\theta} + \text{const}\end{aligned}$$

where the constant term depends only on sampler parameter θ (not ϕ). However, sampling from any distribution $\mathbf{q}_\theta$ with importance sampling have a high variance in the estimate of Z_1. The optimal distribution (with least variance) to sample from is $\mathbf{q}_\theta^*(\mathbf{x}) \propto \exp(-c_\phi^1(\mathbf{x}))$. Thus similar to guided cost learning [7], we optimize $\mathbf{q}_\theta$ as well to make it more likely distribution under the market cost function ϕ. The objective for the sampler $\mathbf{q}_\theta$ is:

$$L_1(\theta) = \mathbf{E}_{\mathbf{x}\sim\mathbf{q}_\theta}[-c_\phi^1(\mathbf{x})]$$

The market sampler (we call it the generator) samples from $\mathbf{x} \sim \mathbf{q}_\theta$ to minimize the cost on sampled trajectories while the cost function parameter (we call it the critic) uses network parameter c_ϕ to maximize the cost for sampled trajectories and minimize the cost from the real trajectories. The objective for market state model has the Wasserstein-1 distance form [4, 12]:

$$W(P_r, q_\theta) = \sup_{||c_\phi||_L \leq 1} \mathbf{E}_{\mathbf{x}\sim P_r}[c_\phi(\mathbf{x})] - \mathbf{E}_{\mathbf{x}\sim\mathbf{q}_\theta}[c_\phi(\mathbf{x})]$$

where the supremum is over all the 1-Lipschitz functions $c_\phi : X \to R$. The Wasserstein-1 distance with gradient penalty mitigates the vanishing gradient problem observed when minimizing Jensen-Shannon divergence and empirically performs better [10, 4, 12]. Thus, similar to [12], we optimize Wasserstein distance with the gradient penalty term to learn the *market state model*:

$$\min_{c_\phi} \max_{\mathbf{q}_\theta} \mathbf{E}_{\mathbf{x}\sim P_r}[c_\phi(\mathbf{x})] - \mathbf{E}_{\mathbf{x}\sim\mathbf{q}_\theta}[c_\phi(\mathbf{x})] + \lambda L_{gp}$$

where $L_{gp} = \mathbf{E}_{\hat{\mathbf{x}}\sim P_{\hat{\mathbf{x}}}}[||\nabla_{\hat{\mathbf{x}}}c_\phi(\hat{\mathbf{x}})||_2 - 1)^2$ and $P_{\hat{\mathbf{x}}}$ is the sampling distribution[1]. If we parameterize the cost

---

[1] $t\mathbf{x} + (1 - t)\tilde{\mathbf{x}} = \hat{\mathbf{x}} \sim P_{\hat{x}}$ with $t \in \mathcal{U}(0, 1)$, $\mathbf{x} \sim P_r$, $\tilde{\mathbf{x}} \sim \mathbf{q}_\theta$.

function c_ϕ with a binary classifier, we get the standard generative adversarial network resulting in generative adversarial imitation learning [10, 13].

Sampling State. The market state $\mathbf{x}$ or the bid request is multi-categorical (or binary) in nature. The generator (market sampler $\mathbf{q}_\theta$) maps random vectors $z \in R^z$ (often multivariate standard Gaussian) to generated inputs as $\hat{\mathbf{x}} = G(z, \theta)$. We represent the function as $G : (R^z, \theta) \to R_{\{0,1\}}^{d_1} \times \cdots \times R_{\{0,1\}}^{d_F}$ where F is the number of categorical random variables and d_i is the number categories for i^{th} categorical random variable. However, due to discrete nature of the output variables, we can not use re-parameterization trick to sample from $G(z, \theta)$ [15]. Thus, we resort to Gumbel-softmax trick to obtain sample from the distribution while allowing the flow of gradient through the neural network [14]. We represent bid request as $\mathbf{x} = [\mathbf{x}^1 \oplus \cdots \oplus \mathbf{x}^F]$ where $\mathbf{x}^i$ is the one hot encoding of ith categorical random variable with probability distribution π^i. Using Gumbel-Max trick, we can sample from the distribution π^i as $\mathbf{x}^i = \text{onehot}(\arg\max_j[g_j + \log \pi_j^i])$ where g_j is sample from standard Gumbel distribution. As the argmax operator is not differentiable, we use soft-max to sample from the distribution π^i while allowing to compute gradient. Thus, $\mathbf{x}_j^i \simeq \frac{\exp((g_j + \log \pi_j^i)/\tau)}{\sum_k \exp((g_k + \log \pi_k^i)/\tau)}$ where τ is the temperature parameter with $\tau \to 0$ representing argmax. During training we keep $\tau > 0$ to allow flow of gradient similar to [14].

Market Action Model. Maximization of the 2nd term in Eq. 4.3, $\ell_2 = \mathbf{E}_{\mathbf{a}^M\sim\mathbf{s}^M|\mathbf{x}} \log \mathrm{P}(\mathbf{a}^M|\mathbf{x})$, denotes maximizing log-likelihood of market action given a bid request. Note, the market actions are a tuple of market price and the reward value $\mathbf{a}^M = (\mathbf{w}_i, \mathbf{r}_i)$. We solve when the utility $\mathbf{r}_i$ is click_i. The market price and a click from the user (when ad is shown) are independent of each other; thus we can write $\ell_2 = \mathbf{E}\log(\mathrm{P}(\mathbf{w}|\mathbf{x})) + \mathbf{E}\log(\mathrm{P}(\text{click}|\mathbf{x}))$. Note when $\mathbf{r}_i = \text{imp}_i$, we need to only solve $\ell_2 = \mathbf{E}\log(\mathrm{P}(\mathbf{w}|\mathbf{x}))$ as imp_i is a deterministic function of market price $\mathbf{w}_i$ and bid value $\mathbf{a}_i$.

For the first term $\mathbf{E}\log \mathrm{P}(\mathbf{w}|\mathbf{x})$, we know the market price when the advertiser wins the auction $(\mathcal{W})$ but only have lower bound when the advertiser loses $(\mathcal{L})$. Thus we use fully parametric censored regression to estimate the market price distribution parameterized as $\mathcal{N}(f_1(\mathbf{x}), \exp(f_2(\mathbf{x}))^2)$ where f_1, f_2 are deep neural networks [24, 9]. The objective is to maximize:

$$\sum \log \mathrm{P}(\mathbf{w}|\mathbf{x}) = \sum_{\mathcal{L}} \log \mathrm{P}(\mathbf{w} \geq \text{bid}|\mathbf{x}) + \sum_{\mathcal{W}} \log \mathrm{P}(\mathbf{w}|\mathbf{x})$$

The second term, $\mathbf{E}\log(\mathrm{P}(\text{click}|\mathbf{x}))$, is always observed when the user sees the ad impression. Thus, we maximize the binary classification problem,

$\mathbf{E} \log(\mathrm{P(click|\mathbf{x})})$, using logistic regression on observed samples.

4.2 Batch Policy Learning and Evaluation Framework The maximum entropy market model equips us with samples from the market state model $\mathbf{q}_\theta(\mathbf{x})$ and market action model $\mathbf{p}_w(\mathbf{w};\mathbf{x}) \, \mathbf{p}_r(\mathbf{r};\mathbf{w},\mathbf{x},\mathbf{a})$. With any starting advertiser state $\mathbf{s}_0^A = [\mathbf{b}_0 \oplus \mathbf{t}_0]$, the agent can simulate the next advertiser state $\mathbf{s}_1^A$ by sampling the market state $\mathbf{x}_0 \sim \mathbf{q}_\theta$ and the market price $\mathbf{w}_0$ from the action model using 1st conditional independence in Eq. 3.1. The current market state $\mathbf{x}_i$ and price $\mathbf{w}_i$ allow updating the next advertiser state $\mathbf{s}_{i+1}^A$ while market state distribution $\mathbf{q}_\theta$ allows sampling of the next market state $\mathbf{x}_{i+1}$. The simulated training environment enables us to run any model-free RL algorithms to learn the optimal bidding strategy. Further, as the environment allows us to explore, we can start with any budget and time constraints for the advertiser to improve generalization beyond states observed in past experiences. In Algorithm 1, we outline the generic approach for training RL algorithms using the market state and the action model. Moreover using the market simulated (testing) environment, we can evaluate any model under any budget and time constraints. Similar to the training framework, the advertiser can start with any arbitrary state while allowing to evaluate any bidding strategy using the market model. We outline the evaluation framework in Algorithm 2.

Algorithm 1 Generic Approach for Optimal Bidding

1: **Input** Market state model $\mathbf{x} \sim \mathbf{q}_\theta^{\mathrm{train}}(\mathbf{x})$, market action model $\mathbf{w} \sim \mathbf{p}_w^{\mathrm{train}}(\mathbf{w};\mathbf{x})$, $\mathbf{r} \sim \mathbf{p}_r^{\mathrm{train}}(\mathbf{r};\mathbf{w},\mathbf{x},\mathbf{a})$
2: **Input** Advertiser state $\mathbf{s}^D = [\mathbf{b} \oplus \mathbf{t}]$
3: **Input** RL Agent/Strategy with a random Policy
4: **while** agent does not converge **do**
5:    Sample market state $\mathbf{x} \sim \mathbf{q}_\theta^{\mathrm{train}}(\mathbf{x})$, market price $\mathbf{w} \sim \mathbf{p}_w^{\mathrm{train}}(\mathbf{w};\mathbf{x})$
6:    Agent bids $\mathbf{a}$ for observed state $\mathbf{s}^O = [\mathbf{x} \oplus \mathbf{s}^D]$
7:    Agent observes reward $\mathbf{r} \sim \mathbf{p}_r^{\mathrm{train}}(\mathbf{r};\mathbf{w},\mathbf{x},\mathbf{a})$
8:    Update Advertiser state ($[\mathbf{b} \oplus \mathbf{t}]$), Policy
9: **end while**
10: **return** Policy

We use the Dueling Double Deep Q-network (DDQN) to train our agent [23, 22] where we use a current ψ network to train, a target ψ^- network to compute the target value function and a memory replay buffer M to sample from old experience. We optimize:
(4.4)
$$\mathop{\mathbf{E}}_{(s,a,r,s')\sim M}[(r + \gamma Q_{\psi^-}(s', \arg\max_{a'} Q_\psi(s',a')) - Q_\psi(s,a))^2]$$

Q function in DDQN is estimated using two separate

Algorithm 2 Evaluation Framework

1: **Input** Market state model $\mathbf{x} \sim \mathbf{q}_\theta^{\mathrm{test}}(\mathbf{x})$, market action model $\mathbf{w} \sim \mathbf{p}_w^{\mathrm{test}}(\mathbf{w};\mathbf{x})$, $\mathbf{r} \sim \mathbf{p}_r^{\mathrm{test}}(\mathbf{r};\mathbf{w},\mathbf{x},\mathbf{a})$
2: **Input** Advertiser state $\mathbf{s}^D = [\mathbf{b} \oplus \mathbf{t}]$
3: **Input** Agent/Strategy to evaluate
4: Initialize Total Reward to 0
5: **while** episode did not end **do**
6:    Sample market state $\mathbf{x} \sim \mathbf{q}_\theta^{\mathrm{test}}(\mathbf{x})$, market price $\mathbf{w} \sim \mathbf{p}_w^{\mathrm{test}}(\mathbf{w};\mathbf{x})$
7:    Agent bids $\mathbf{a}$ for observed state $\mathbf{s}^O = [\mathbf{x} \oplus \mathbf{s}^D]$
8:    Agent observes reward $\mathbf{r} \sim \mathbf{p}_r^{\mathrm{test}}(\mathbf{r};\mathbf{w},\mathbf{x},\mathbf{a})$
9:    Update Advertiser state ($[\mathbf{b} \oplus \mathbf{t}]$), Total Reward
10: **end while**
11: **return** Total Reward

Adv	$n \times 10^5$	d	imp_{tr}	imp_{te}	cpm_{tr}	cpm_{te}	$\mathrm{KL}(\hat{P}_{tr}||\hat{P}_{te})$
1458	147	2140	0.172	0.302	11.9	20.7	0.057
2259	14	1097	0.65	0.412	56.8	40.6	0.141
2261	12	992	0.602	0.349	52.1	30.0	0.179
2821	29	1919	0.548	0.166	47.5	16.5	0.46
2997	7	428	0.359	0.301	21.4	19.0	0.012
3358	37	1875	0.604	0.405	55.6	39.4	0.171
3386	140	2054	0.168	0.276	12.5	22.0	0.049
3427	140	3970	0.126	0.442	10.1	35.9	0.242
3476	67	1662	0.257	0.419	20.7	31.9	0.078

Table 1: Dataset statistics for iPinYou Advertisers.

branch of the neural network as $Q(s,a) = V(s) + (A(s,a) - \frac{1}{|A|}\sum_a A(s,a))$ where the advantage function is defined as, $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$. We quantize the one-dimensional continuous bidding values (actions) and use a discrete action space RL algorithm, DDQN for improved stability reason on long trajectories [18, 23].

Remark 2. Computational latency is important in real-time bidding system. The final policy π^* takes bid request $\mathbf{x}$, budget left $\mathbf{b}$, and, time left $\mathbf{t}$ as input and computes the optimal bid price $\mathbf{a}$ using neural network ψ in Eq. 4.4. The latency depends on the neural architecture; we use a simple three hidden layer neural network. One can easily augment the policy with a memory state to encode the past using a recurrent neural network. However, in such cases, the bidding platform needs to store memory state for each advertiser in addition to budget and time constraints.

5 Experimental Results

5.1 Experimental Setup

Datasets. We use two real-world datasets to conduct our experiments. Publicly available **iPinYou** dataset contains bidding data from 9 advertisers over multiple days [26]. Bid requests contain multiple categorical features such as city, domain, etc. Table 1 lists total number of bid requests ($n \times 10^5$), categorical features (d), average train and test impression
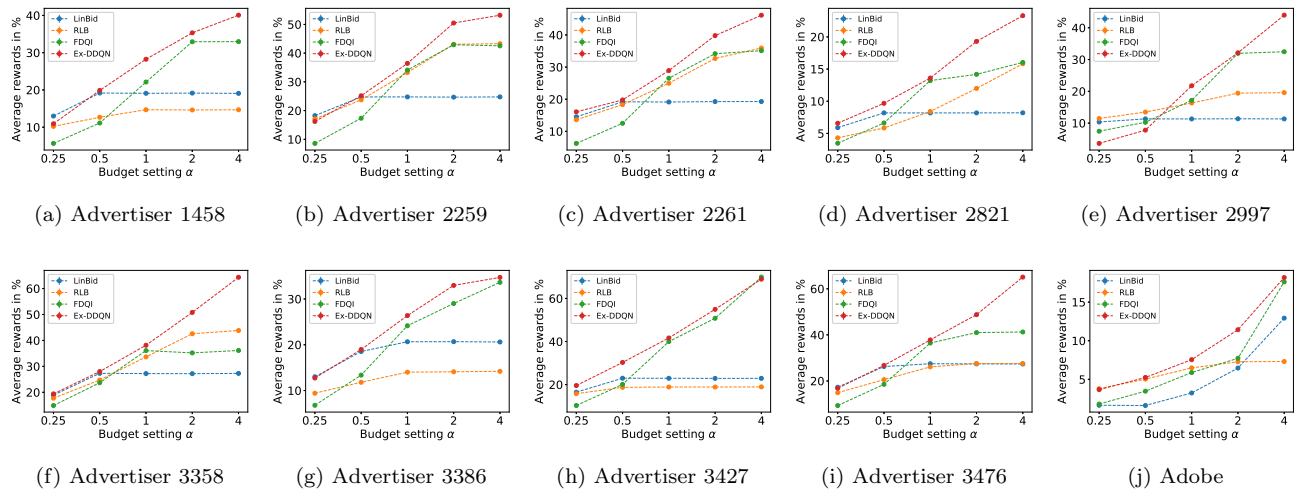
Figure 2: Average Rewards (Impressions) under varied budget settings on iPinYou Advertisers and Adobe Dataset. Standard deviation (std) is on the order of 10^{-1}.

$(\text{imp}_{tr}, \text{imp}_{te})$, training and testing cost per thousands bid requests $(\text{cpm}_{tr}, \text{cpm}_{te})$, and, the KL divergence between train and test empirical market price distribution (where a higher number implies market price distribution in train and test datasets are more different). We learn the optimal bidding strategy from the perspective of each advertiser on the iPinYou dataset. For experimental purposes, we also collect a fraction of bid requests from a week's data from **Adobe**, one of the leading DSP. The bid requests contain categorical features similar to iPinYou. We learn an aggregate optimal bidding strategy for the Adobe DSP without splitting the advertisers. The total number of bid requests and categorical features are $2,323,226$ and 1282 respectively.

Baseline Methods We compare the following methods using the same testing environment with varied budget constraints.

LinBid [19] is the linear bidding strategy with $bid = b_0\theta(\mathbf{x})$ where b_0 is the base parameter tuned on the training dataset to maximize total rewards.

RLB-Segmentation [5] is the state-of-the-art model that solves the Bellman equation using model approximation. Further, RLB uses coarse to fine segmentation to handle large sequences.

FDQI [20] is a generic method for batch RL using neural fitted deep Q-iteration. Although early works in real-time bidding systems do not use FDQI, we add this general batch RL method as a baseline.

Ex-DDQN is our proposed approach based on the market environment for exploration. We learn the training market environment using the maximum entropy principle. We train model-free dueling Q-network with

varied budget constraints using the learned training environment.

Training and Testing Environment. For each advertiser in the iPinYou dataset, we use the 60%, 15%, 25% of the days' samples as the training, validation and testing data respectively. For the Adobe dataset, instead of days, we randomly split the samples as the training, validation, and testing batch dataset with the same fraction. For evaluation, we learn the simulated market environment using only the test datasets. In particular, we learn market state model, impression and click models from test datasets (we use validation datasets to do early stopping). For training, all algorithms use training datasets. For exploration during training, Ex-DDQN uses the simulated market environment learned using training datasets.

We learn the market state model using the Wasserstein Generative Adversarial Network with gradient penalty (WGAN-GP) [12]. To learn the impression model, we maximize fully parametric censored regression [9]. We learn the click model using logistic regression. The impression and click model allows us to sample market price and user click behavior respectively. To generate a bid request, we sample from a standard normal distribution as the input to the generator of the WGAN-GP. We learn these three models to simulate the market environment from the train and test datasets (for training and evaluation purposes respectively). Note, *in no cases*, learning training and the testing environment share any samples. Network architecture, featurization, hyper-parameters used for tuning can be found in the long version of this paper.

Advertiser	Test Sample	Model Sample	Uniform Sample
1458	0.138 ± 0.013	0.253 ± 0.028	2.622 ± 0.123
2259	0.142 ± 0.016	0.182 ± 0.021	2.668 ± 0.118
2261	0.141 ± 0.013	0.166 ± 0.017	2.408 ± 0.079
2821	0.143 ± 0.015	0.184 ± 0.026	2.857 ± 0.098
2997	0.145 ± 0.019	0.15 ± 0.019	4.57 ± 0.187
3358	0.141 ± 0.014	0.339 ± 0.049	2.097 ± 0.065
3386	0.141 ± 0.013	0.228 ± 0.039	2.732 ± 0.128
3427	0.144 ± 0.013	0.2 ± 0.022	2.158 ± 0.108
3476	0.143 ± 0.014	0.172 ± 0.015	1.99 ± 0.108
Adobe	0.143 ± 0.013	0.156 ± 0.015	1.558 ± 0.063

Table 2: $\sqrt{n} \times \hat{\text{MMD}}$ Distance between Test samples and {Test, Model, Uniform} samples. $n = 200$.

Evaluation Metric We consider the number of *total impressions*, over the trajectories, as the utility of interest. Thus, given a budget and time constraint, the goal is to maximize the number of impressions for the advertiser. Note, previous research often considers *click* as the utility removing all non-impressed bid requests [5]. However, for a click to occur, the advertiser needs to win the auction first. Our framework allows both click and impression as the utility measure. However, as clicks are sparse ($\sim 0.1\%$), we choose to tabulate results with impressions as the utility measure to reduce variance in the estimation. Further, we consider a more realistic large scale evaluation where we optimize bidding strategy for a sequence of $T_0 = 100,000$ bid requests, which correspond to real-world 10 minutes auction volume in case of iPinYou [5, 26].

To evaluate generalization properties of any bidding strategy, we set budget as $B_0 = \alpha \times \text{cpm}_{\text{te}} \times \frac{T_0}{10^3}$ where cpm_{te} is the average cost of the advertiser on the test dataset over 1000 bid requests. We set $\alpha = \{\frac{1}{4}, \frac{1}{2}, 1, 2, 4\}$ as the budget constraint. Previous research only considers $\alpha = \{\frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}\}$, and, set test datasets where only the advertisers win the impression. However, this setting is very unrealistic in real-world scenarios; besides the test distribution shift, previous works consider cases where the budget is strictly less than the spent [5]. In our evaluation, we do not directly use the test dataset; instead, we learn a test market sampler and use samples from the sampler to test all model fairly. For all budget settings, we repeat the experiment 10 times to get the average and the standard deviation (std) numbers.

5.2 Results In Figure 2, we plot the percentage of impressions won over the trajectories ($100,000$ timesteps) for all advertisers on the iPinYou dataset as well as on the Adobe dataset with different budget constraints.

On the lowest budget settings (budget $= \alpha \times \text{cpm}_{\text{te}} \times \frac{T_0}{10^3}$) with $\alpha = \frac{1}{4}$, all batch RL methods, RLB,

LinBid, FDQI have experiences from the historical interactions; thus we expect relatively better performance from these algorithms. For all advertisers, except for advertiser 2997, Ex-DDQN consistently performs similar or better than RLB/LinBid methods while significantly improving on the FDQI method. Performances of RLB and Linbid are mixed with cases one works better than the other and vice-versa. For budget setting $\alpha = \frac{1}{2}$, Ex-DDQN performs significantly better than all algorithms on nine of the ten cases. The average gain over the next best performance on the nine advertisers is more than 7%. While FDQI does not perform well, the other two batch algorithms, LinBid, and, RLB perform relatively well with a few cases improving upon Ex-DDQN. The result suggests Ex-DDQN performs similar (and sometimes better) to the state-of-the-art method when budget settings are small ($\alpha \leq 1/2$).

For budget settings $\alpha \geq 1$, we compare the generalization performance of the batch RL algorithms. Interestingly in all cases, Ex-DDQN significantly (sometimes with margin $\geq 10\% - 30\%$) improves LinBid and RLB algorithms while FDQI follows Ex-DDQN performance. The significant performance gain suggests agents trained to explore with varied advertiser states on the simulated training environment can potentially generalize on the new test environment (even where train and test datasets are significantly different, see Table 1).

We further note that RLB performs relatively better when $\alpha \leq 1/2$ and KL divergence (listed in Table 1) between empirical train and test market price is small. RLB being a model-based RL algorithm solves Bellman equation with empirical training market price; thus making them sensitive to model mismatch error. For example, on advertiser 2821 and 3427 (where KL divergences are particularly high), RLB performs significantly worse than even the LinBid algorithm. Both RLB and Linbid have saturating behavior when budget parameter α is more than 1. The saturating behavior suggests that these batch models face difficulty extrapolating optimal behavior with higher budget constraints.

5.3 Market Model Evaluation We evaluate our testing environment to compare whether the generated model bid requests reflect the true samples from the test distribution. We sample $n = 200$ bid requests from test batch dataset and sample another 200 bid requests each from test batch dataset, maximum entropy market model and uniformly randomly from the categorical features to compute empirical Maximum Mean Discrepancy (MMD) distance [11]. In Table 2, we list the average empirical MMD distance and standard deviation from repeating the process 100 times. We use Gaussian

kernel with $\sigma = 1$. Empirical MMD distance is close to zero for samples between test batch datasets and samples from the market model while for a uniformly random sample, distance is an order of magnitude larger. This result validates our testing environment learned using the maximum entropy principle being close to the actual one in real-traffic.

6 Discussion

In this paper, we tackle one of the key challenges in the real-time bidding system, learning the optimal bidding strategy from a batch dataset. We propose a generic framework for evaluating models from batch datasets which were lacking in previous literature. Further, we propose a solution to learn the optimal strategy from the batch dataset that has the property to generalize to unseen state space in addition to competitive (sometimes better) performance on the known regime. Although we analyze only on RTB systems, learning from the batch dataset poses a significant challenge to any recommender system-based solutions. Without interacting with the user (or market) in real-traffic, evaluation/training becomes difficult. Potentially our framework can be applied in recommender systems where state and transition function decomposes similar to RTB systems; such extensions we leave for future work.

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*. ACM, 2004.

[2] K. Amin, M. Kearns, P. Key, and A. Schwaighofer. Budget optimization for sponsored search: Censored learning in mdps. In *UAI*, 2012.

[3] A. Antos, C. Szepesvári, and R. Munos. Fitted q-iteration in continuous action-space mdps. In *NeurIPS*, 2008.

[4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[5] H. Cai, K. Ren, W. Zhang, K. Malialis, J. Wang, Y. Yu, and D. Guo. Real-time bidding by reinforcement learning in display advertising. In *WSDM*. ACM, 2017.

[6] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song. Generative adversarial user model for reinforcement learning based recommendation system. In *ICML*, 2019.

[7] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML*, 2016.

[8] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

[9] A. Ghosh, S. Mitra, S. Sarkhel, J. Xie, G. Wu, and V. Swaminathan. Scalable bid landscape forecasting in real-time bidding. In *ECML-PKDD*. Springer, 2019.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

[11] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar), 2012.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017.

[13] J. Ho and S. Ermon. Generative adversarial imitation learning. In *NeurIPS*, 2016.

[14] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[15] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[16] S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement learning*. Springer, 2012.

[17] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *KDD*. ACM, 2013.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[19] C. Perlich, B. Dalessandro, R. Hook, O. Stitelman, T. Raeder, and F. Provost. Bid optimizing and inventory scoring in targeted online advertising. In *KDD*. ACM, 2012.

[20] M. Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *ECML*. Springer, 2005.

[21] R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[22] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[23] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

[24] W. C.-H. Wu, M.-Y. Yeh, and M.-S. Chen. Predicting winning price in real time bidding with censored data. In *KDD*. ACM, 2015.

[25] S. Yuan and J. Wang. Sequential selection of correlated ads by pomdps. In *CIKM*. ACM, 2012.

[26] W. Zhang, S. Yuan, J. Wang, and X. Shen. Real-time bidding benchmarking with ipinyou dataset. *arXiv preprint arXiv:1407.7073*, 2014.

[27] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, 2008.