YAOGAN: LEARNING WORST-CASE COMPETITIVE ALGORITHMS FROM SELF-GENERATED INPUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

We tackle the challenge of using machine learning to find algorithms with strong worst-case guarantees for online combinatorial optimization problems. Whereas the previous approach along this direction (Kong et al., 2018) relies on significant domain expertise to provide hard distributions over input instances at training, we ask whether this can be accomplished from first principles, i.e., without any human-provided data beyond specifying the objective of the optimization problem. To answer this question, we draw insights from classic results in game theory, analysis of algorithms, and online learning to introduce a novel framework. At the high level, similar to a generative adversarial network (GAN), our framework has two components whose respective goals are to learn the optimal algorithm as well as a set of input instances that captures the essential difficulty of the given optimization problem. The two components are trained against each other and evolved simultaneously. We test our ideas on the ski rental problem and the fractional AdWords problem. For these well-studied problems, our preliminary results demonstrate that the framework is capable of finding algorithms as well as difficult input instances that are consistent with known optimal results. We believe our new framework points to a promising direction which can facilitate the research of algorithm design by leveraging ML to improve the state of the art both in theory and in practice.

1 INTRODUCTION

It has been a long-standing ambition of machine learning to learn from *first principles*, that is, a machine learning program that learns *without any human-provided data* beyond an appropriate definition of the task at hand (Silver et al., 2018). The community has made impressive strides towards this goal in recent years, most notably in the domain of tabletop games such as chess, Go, shogi (Silver et al., 2018) and poker (Brown & Sandholm, 2018), where programs have been trained to play at superhuman levels without consuming any game-specific human expertise except, naturally, the rules of the game.

In this work, we look at the task of using ML to design algorithms for optimization problems. Similar to tabletop games, the task of algorithm design has succinct and well-defined objectives, so learning from first principles is potentially feasible. Moreover, the task is very intriguing as it typically requires significant amount of time and expertise for humans to accomplish.

In our context, the problem of learning algorithms from first principles is also related to an important issue in both ML and algorithms: what input instances is the learned algorithm expected to handle? The typical ML approach is to train the program against a pre-defined training set, with the assumption that the training set is carefully chosen to capture future input distributions well, so the learned algorithm generalizes to future input of interest. On the other hand, the algorithm design community in theoretical computer science (TCS) traditionally considers algorithms with worst-case guarantees due to their robustness against arbitrary input. We focus on the latter domain in our work.

Along the line of learning worst-case algorithms, a particularly relevant result is by Kong et al. (2018) that uses reinforcement learning to learn theoretically optimal algorithms for online optimization problems, and one of the key ideas they introduce is the notion of universal and high-entropy training sets. Roughly speaking, such a training set serves as a very effective benchmark in the sense that the task of finding an algorithm with strong worst-case guarantee reduces to finding

an algorithm that performs well against this training set. Not surprisingly, for a given optimization problem, coming up with such a training set can require significant domain expertise a priori. In the current work the goal is to go beyond the framework of universal training sets, and enable learning of algorithms without even this prior domain knowledge.

To overcome this issue, we draw insights from the fields of game theory, analysis of algorithms, and online learning, as well as generative adversarial networks (GANs, Goodfellow et al. (2014)). In particular, we introduce a training framework that aims to learn a strong worst-case algorithm and an effective benchmark training set simultaneously, all without relying on any human domain expertise beyond specifying how to compute the objective of the optimization problem. If successful, such an approach would have impactful ramifications: researchers could accelerate the speed of discovery by offloading to machines the time-consuming process of finding problem-specific insights, which can be crucial in improving the state of the art for many problems both in theory and in practice.

At the high level, our framework has two components: the algorithm agent and the adversary. Their goals are to learn the optimal algorithm and a set of input instances that captures the essential difficulty of the given optimization problem, respectively. For simple optimization problems, rudimentary strategies such as random search or grid search can provide a fairly effective adversary. For problems with more interesting underlying space, we exploit deep learning to guide our search for the distribution of difficult inputs. Similar to a generative adversarial network (GAN), we use two neural networks, one for the algorithm agent and another for the adversary. We start from scratch, train these two networks against each other, and let them evolve simultaneously. We call our program *YaoGAN* due to this structural similarity to GAN and the connection to the classic Yao's Lemma (see Section 2.1 for details); this is our main technical contribution.

In this initial effort, we follow Kong et al. (2018) in focusing on online optimization problems where each input is a sequence arriving over time, and the algorithm needs to make an immediate and irrevocable decision at each time step without knowing the future input. We further restrict our attention to problems that can be solved or approximated by algorithms requiring a small amount of memory. There are abundant well-studied and interesting problems bearing such properties in online optimization, typically when the structure of the problem exhibits certain Markov property so it is sufficient for the algorithm to maintain a succinct state capturing past information.

We consider two problems, the *ski rental* problem and the *fractional AdWords* problem. The former is a classic problem often used as the introductory example to online algorithms (Buchbinder et al., 2009). The latter is the fractional variant of the AdWords problem (Mehta et al., 2007), which is a form of the classic bipartite matching problem with "budgets," and is motivated by problems in online advertising. For these problems, we have a good understanding of the structures of the optimal algorithm and difficult inputs, so we can verify that our paradigm leads to something meaningful. Our experiments, although fairly limited and preliminary, demonstrate that our framework can be effective at finding algorithms as well as difficult input instances that are consistent with optimal results. We believe further study along this direction can lead to more exciting developments.

We use generic neural network architecture and ML methods (e.g., Adam optimizer (Kingma & Ba, 2014)) in our experiments, and do not optimize the training set-up or hyperparameters, since our focus in this paper is not in extending ML techniques, but in applying our high-level paradigm to learn algorithms from first principles. We believe the co-evolution of new ML techniques tailored to this task will be essential to the ultimate success of this approach. We leave that aspect, as well as applying the framework to more open algorithm design problems, to future work.

Related work. The most relevant paper to ours is an approach by Kong et al. (2018) that uses reinforcement learning (RL) to find competitive algorithms for online optimization problems including the AdWords problem, online knapsack and the secretary problem. They introduce the notions of universal and high-entropy training set that are tailor-made to facilitate the convergence to worst-case competitive algorithm. Deudon et al. (2018) and Kool et al. (2018) use RL to solve the traveling salesman problem (TSP) in the two-dimensional plane. Both uniformly sample the input points from the unit square and teach the network to optimize the expected performance on this input distribution without having access to exact solutions. Nevertheless, the learned algorithms from both approaches seem competitive with state-of-the-art methods, at least for modest input sizes. Vinyals et al. (2015) introduce the Pointer Network, a sequence-to-sequence model that successfully manages to approximately find convex hulls, Delaunay triangulations and near-optimal planar TSPs. They assume

access to an exact solution and use supervised learning to train the model, while their input is uniformly sampled from the unit square. Graves et al. (2014) introduce the Neural Turing machines, a differentiable and computationally universal Turing machine model, and use supervised learning to teach them simple algorithms such as copying or sorting. Kaiser & Sutskever (2015) introduce the parallel and computationally universal Neural GPU and use supervised learning to teach it addition and multiplication that seemingly generalizes to arbitrary lengths.

We point out some other related results without going into details: the work of Bello et al. (2016) studies applying RL on combinatorial problems including TSP and Knapsack; the work of Dai et al. (2017) exploits embedding and RL to design heuristic algorithms for classic graph problems on specific distributions; Boutilier & Lu (2016) uses RL to solve Budget Allocation problem; Duetting et al. (2019) applies deep learning techniques to designing incentive compatible auction mechanisms.

To the best of our knowledge, previous results mostly focus on the algorithm side, and either learn algorithms designed to work on specific input distributions of interest, or utilize supervised learning to mimic a known method, or (in the case of Kong et al. (2018)) rely on domain expertise to provide hard input instances to train algorithms that are competitive in worst-case. We do not wish to detract from the value of optimizing an algorithm on a specific distribution if it is of special interest. Indeed, a practitioner optimizing for a distribution $\mathcal{D}$ could likely design algorithms that outperform classic solutions on $\mathcal{D}$ since classic solutions typically protect against worst-case inputs and might be overly pessimistic on $\mathcal{D}$. Our goal is different: facilitate the discovery of new algorithms with strong worst-case performance on open problems where we don't have a known difficult input distribution at hand. Such learned strategies could then potentially be later fine-tuned to adapt to specific input distributions, while retaining some worst case guarantees.

Finally, from the algorithm design literature, the problems we consider are very well-studied online optimization problems. The ski rental problem (Karlin et al., 1986) has been a staple introductory example in online competitive analysis. The fractional AdWords problem is a fractional relaxation of the AdWords problem (Mehta et al., 2007), which is motivated by online advertising. Moreover, the Adwords problem is closely related to numerous variants of the online bipartite matching problem, which is still an actively studied area in algorithm design, for example in the setting of ride-sharing platforms (Dickerson et al., 2017; Ashlagi et al., 2019). We refer readers to Buchbinder & Naor (2009) and Mehta et al. (2013) for a more thorough coverage.

1.1 Summary of results

As a brief introduction, we can summarize YaoGANs as essentially starting with a "random" algorithm network, then alternating between (1) finding the *best response* (that will force the algorithm to exhibit its worst behavior), and (2) training the algorithm network on this best response. Details are given in Section 3.

We apply the YaoGAN method to two classic problems in the area of online optimization: the *ski rental* problem and a fractional version of the *AdWords* problem. The two problems exhibit a progression in terms of how hard it is to find the "best responses." Finding best response inputs for ski rental is relatively easy—an approximately exhaustive search (over a suitably parameterized space) is sufficient—while for AdWords it seems to be a hard non-convex global optimization problem.

Ski rental. A few minutes of training leads to an algorithm whose *competitive ratio*—the primary figure of quality for an online optimization algorithm—converges approximately to 1.59, which is within 10^{-2} of the theoretically best value achievable. Furthermore, the behavior of trained algorithm exactly matches the one of Karlin et al. (1986) (Appendix B).

Fractional AdWords. For the fractional AdWords problem—a "budgeted bipartite matching problem"—with 3 advertisers with a budget of 3 each and 9 ad slots[1], about 10 hours of training is sufficient to achieve a competitive ratio of 0.73, which is within 10^{-2} of the theoretically best value achievable for this problem. The trained algorithm qualitatively recovers both the greedy strategy and the balance strategy (Kalyanasundaram & Pruhs, 2000), two defining aspects of the optimal solution, as well as seems to find the correct form of tradeoff between the strategies in tested scenarios. (Section 4).

[1]Note that while the problem solved here is defined on "fixed size" instances (27 real numbers), this is already well beyond the regime of executing a simple exhaustive search to discover the optimal algorithm.

2 PRELIMINARIES

We start with a discussion of online algorithms and the classic theoretical results upon which we draw insights for our approach. We will keep the discussion high-level and informal since we use these results mostly for intuition rather than extending them to prove any theoretical guarantee.

We chose online optimization problems as the test bed for our approach because they are rich in structure and application. Online problems comprise a theoretical framework to study tasks in the interactive setting. In particular, an agent is given a sequence of requests that require a response without knowing future requests. A successful algorithm typically needs to hedge between receiving an immediate reward and allowing for potentially lucrative future rewards. The *competitive ratio* of an online algorithm for a particular input instance is defined as the ratio of the (expected) objective value the online algorithm gets for the input over the optimal value an algorithm can receive for the instance if it knows the entire input upfront (i.e., the offline optimal). The worst-case competitive ratio of an online algorithm is its competitive ratio for the worst possible input. We usually drop the "worst-case" when we talk about the competitive ratio of an algorithm.

We now introduce the ski rental problem as a simple example to keep the following discussion concrete. The AdWords problem is introduced in Section 4.

Problem 1 (Ski rental) *Suppose we want to go skiing in the winter, which requires a set of skis. Each day we go skiing, we have two options: either rent the skis for $\$1$, or buy the skis for $\$B$, for some fixed B. Naturally, renting the skis allows us to ski only for that one day, while after buying we can use them for an unlimited number of days. The crux is that we do not know in advance the total number of days k we will be able to ski (e.g., the weather might change, we might break a leg, etc.). The objective is to minimize the total amount of money spent by strategically buying/renting skis.*

Deep learning approaches typically train a tunable network on a particular distribution of interest. When applied to learning algorithm with worst-case guarantees, it is not clear apriori what distribution should one train on. Instead, we take a game-theoretic approach that formalizes the interplay between the algorithm and its inputs.

2.1 GAME-THEORETIC VIEW

Consider a (possibly infinite) two-player zero-sum game matrix V, where we denote the row player as the *algorithm player*, and the column player as the *adversary* who determines the inputs to the algorithm. Each row represents a deterministic algorithm $\mathcal{A}$, and each column represents a specific input instance $\mathcal{I}$, and the corresponding entry $V(\mathcal{A}, \mathcal{I})$ specifies the payoff to the adversary if the algorithm player plays strategy $\mathcal{A}$, and the adversary plays strategy $\mathcal{I}$. We follow the custom that the algorithm player aims to minimize the payoff to the adversary, while the adversary tries to achieve the opposite. In the case of online algorithms, if we are working with a problem with minimization objective, the payoff $V(\mathcal{A}, \mathcal{I})$ will be the competitive ratio of $\mathcal{A}$ for input $\mathcal{I}$. Similarly, for a maximization problem, we take $V(\mathcal{A}, \mathcal{I})$ as the negation of the above competitive ratio.

For concreteness, we exemplify the above on the ski rental problem. A deterministic strategy $\mathcal{A}$ takes the form of renting for up to some fixed number x of days initially, and if it turns out that we go skiing for at least x days, we buy on day x. If the eventual number of days we go skiing is k, the cost of this strategy will be $\min(k, B + x - 1)$, and the offline optimal cost will be $\min(k, B)$. The corresponding entry in the payoff matrix V is $\frac{\min(k,B+x-1)}{\min(k,B)}$, and the competitive ratio of this deterministic algorithm is $\max_{k\geq 1} \frac{\min(k,B+x-1)}{\min(k,B)}$, i.e., for the worst possible input k.

Yao's Lemma. A randomized algorithm can be viewed as a distribution $\boldsymbol{p}$ over rows of V (i.e., deterministic algorithms); a input distribution $\boldsymbol{q}$ can be viewed as a distribution over columns of V (i.e., pure inputs). The optimal randomized algorithm $\boldsymbol{p}^*$ is one whose expected payoff against the worst-case input is minimized, i.e., $\boldsymbol{p}^* = \arg\min_{\boldsymbol{p}} \max_{\mathcal{I}} \boldsymbol{p}^T V(:, \mathcal{I})$. A standard result in game-theory stipulates that in this case maximizing over pure inputs is equivalent to maximizing over input distributions. The namesake of our paper, Yao's Lemma (Yao, 1977) (see also Borodin & El-Yaniv (1998)) says that there is an adversarial distribution $\boldsymbol{q}^*$ over inputs such that the expected payoff of the best deterministic algorithm over $\boldsymbol{q}^*$ is the same as the value of the best randomized algorithm $\boldsymbol{p}^*$ over its worst possible input. In other words, $\max_{\mathcal{I}} (\boldsymbol{p}^*)^T V(:, \mathcal{I}) = \min_{\mathcal{A}} V(\mathcal{A}, :)\boldsymbol{q}^*$. Using

classic game theory notations, we also refer to the best randomized algorithm as the row player's minmax strategy, and $\boldsymbol{q}^*$ the column player's maxmin strategy.

Solving an online optimization problem corresponds to solving a zero-sum game, i.e., finding both $\boldsymbol{p}^*$ and $\boldsymbol{q}^*$. Prior work (Kong et al., 2018) explores learning online algorithms by training only the $\boldsymbol{p}^*$ side. They address the $\boldsymbol{q}^*$ by considering only problems for which the optimal input distribution is known. However, there are two acknowledged weaknesses in this approach:

First, such knowledge of the adversarial maxmin distribution is typically only available for well-studied problems where we already have a theoretically established limit on the performance of any algorithm. Thus, to fulfill the ultimate goal of using machine learning to discover (unknown) algorithms with competitive "worst-case" performances, we need an approach without relying on domain knowledge of a worst-case input distribution.

Second, a somewhat fundamental issue with the approach is that an algorithm that performs well against the adversarial distribution alone is in fact not necessarily the optimal algorithm. For a simple example, consider the game of rock-paper-scissors. The maxmin column strategy is to play the three moves uniformly at random, and any row strategy (e.g., always play rock) is a best response to the column player's minmax strategy. Indeed, this was acknowledged in Kong et al. (2018), e.g., for the AdWords problem they combined the adversarial distribution with another hand-picked input distribution to avoid converging to a suboptimal greedy algorithm.

3 YAOGAN

In this work, the goal is to explicitly find the minmax strategy for the row player, by co-training the row and column player networks. YaoGAN facilitates this by training both an *algorithm network* and an *adversarial network*. On a conceptual level, YaoGAN alternates between (a) training the adversarial network to produce the best response to the algorithm network (in terms of the payoff matrix, i.e., competitive ratios), and (b) training the algorithm network on the generated input. We give a brief theoretical motivation behind YaoGANs in Appendix A.

3.1 YAOGAN PARADIGM

We instantiate a trainable neural network representing the algorithm that starts by outputting meaningless (random) values, and our goal is to train it into a competitive algorithm. Similarly, we use some method to generate (approximately) best response inputs: This can either by an *exhaustive search* through all deterministic inputs, or can be an trainable *adversarial neural network* that starts off with outputting meaningless values and can be tuned. We focus on the more interesting, latter, case. Finally, we are given a way to evaluate the performance of the algorithm on a input. In the realm of online algorithms this corresponds to running the algorithm network, running a solver to compute the offline optimum and then reporting the competitive ratio.

Experience array. We maintain an *experience array* of input instances that we use to train the algorithm neural network. At each training step of the algorithm, we take some number of samples from the experience array, evaluate the algorithm's performances (i.e., the competitive ratio in our case), find the worst input among them for the current algorithm, and train the algorithm against that input.

All instances in the experience array are generated by the adversarial network at some point during the training. At every step the algorithm network is trained against the best response from some number of samples from the experience array. We periodically append new input instances to the experience array, which are chosen among inputs generated by the adversarial network and the samples from the current experience array. We append among these candidates the one that is the best response against the algorithm network.

This stabilizes the training and resolves a multitude of issues. The most apparent being that training the adversarial network to convergence takes significantly more time than one step of the algorithm training. Experience arrays amortizes the time spent during adversarial training with an equal amount of algorithmic network training. Furthermore, an empirical issue that has been hindering the training is that the adversarial network can often fail to find the best response possible. This is not at all unexpected since the question can be in general intractable. Experience arrays allow the

adversarial networks to use multiple tries in finding the best counterexample. This counterexample might then be used multiple times since one often needs to train on it a few times in succession or revisit it during later training.

One explanation on why such training is effective is that hard distributions often have small support (a small number of different inputs). This is the case in Kong et al. (2018) where the handcrafted adversarial distributions are often comprised of a small number of different inputs (although their structure might be complex). Furthermore, even if all sets are of large size, one might still find a satisfactory small-set approximation that converges to the right set.

Pseudocode. The pseudocode of YaoGAN is given in Algorithm 1. We make a final note that our adversarial network takes n_{noise}-dimensional Gaussian noise as input and generates a problem input. The random noise was typically attenuated during training, but we empirically found that it helped with input space exploration. We use hyperparameters $n_{\text{batch}} = 100$, $n_{\text{noise}} = 100$, $T_{\text{alg}} = T_{\text{adv}} = 4$, $T_{\text{add}} = 100$, and $T_{\text{restart}} = 100$. The Adam (Kingma & Ba, 2014) optimizer is used with paper suggested parameters.

Algorithm 1 Generic YaoGAN Training Paradigm.

Require: Differentiable evaluation function $V(\cdot, \cdot)$
Require: Parameters $T, T_{\text{alg}}, T_{\text{adv}}, T_{\text{add}}, T_{\text{restart}}, n_{\text{batch}}$.
  Instantiate algorithm network ALG, adversarial network ADV
  Generate n_{batch} inputs using ADV, and add them to the experience array E.
  For $t = 1, \ldots, T$
    For $u = 1, \ldots, T_{\textbf{alg}}$
      $I \leftarrow n_{\text{batch}}$ uniformly randomly sampled inputs from E.
      $i^* \leftarrow \arg\max_{i \in I} V(\text{ALG}, i)$
      Update ALG using Adam($\nabla_{\text{ALG}} V(\text{ALG}, i^*)$).
    For $u = 1, \ldots, T_{\textbf{adv}}$
      Sample n_{batch} random Gaussian tensors z
      Update ADV using Adam($\nabla_{\text{ADV}} V(\text{ALG}, \text{ADV}(z))$).
    If $t \bmod T_{\text{add}} = 0$
      Generate n_{batch} inputs using ADV, sample n_{batch} inputs from E.
      Among the above inputs, pick the worst one for ALG, append it to E.
    If $t \bmod T_{\text{restart}} = 0$
      Reinitialize ADV to produce random (untuned) problem inputs.

4 ADWORDS

In this section, we show how to apply the YaoGAN paradigm to the fractional version of the classic AdWords problem. The algorithm we derive here will apply to fixed-length inputs. In **Appendix B**, we present an application of the YaoGAN paradigm to derive a uniform algorithm (one that works for all input lengths) for the **Ski Rental** problem.

We start with defining the **AdWords** problem Mehta et al. (2007): There are m advertisers with budget B each, and n ad slots. Each ad slot j arrives sequentially along with a vector $v^j = (v_0^j, v_2^j, \ldots, v_{m-1}^j)$ where v_i^j is the bid of advertiser i for slot j. Once a slot arrives the algorithm must irrevocably allocated to an advertiser or no one at all. In the event that the ad j is allocated to advertiser i, the algorithm collects a revenue which is the minimum of v_i^j and advertiser i's remaining budget. The remaining budget of advertiser i is reduced accordingly. The objective of the algorithm is to maximize the total revenue over the entire sequence.

Optimal Algorithm. Under a "small bid" assumption, which says that $\forall\, i, j : v_i^j \ll B$, the optimal online algorithm for AdWords is the following by Mehta et al. (2007). Define $s_i^j := \frac{r_i^j}{B}$ as the fraction of advertiser i's budget remaining when ad j arrives. Let the "scaled bid" of the slot j for i be $q_i^j := v_i^j(1 - e^{-s_i^j})$. The ad slot j is assigned to the advertiser i with the largest scaled bid q_i^j (with arbitrary tie breaking). The algorithm always guarantees at least a $(1 - 1/e)$ fraction of the offline optimal revenue, which is asymptotically optimal.

Here, we will study the fractional version of the problem.

Problem 2 (Fractional AdWords) *The fractional problem is the same as in AdWords, except that the algorithm can make fractional allocations. That is, for each ad slot j, the algorithm chooses an assignment $(p_0^j, p_1^j, \ldots, p_{m-1}^j) \in \mathbb{R}_{\geq 0}$ where $\sum_{i=0}^{m-1} p_i^j \leq 1$. The budget of each advertiser i with remaining budget r_i^j is then depleted by $\min(r_i^j, p_i^j \cdot v_i^j)$ and the algorithm's revenue is increased by the same amount (summed over all i).*

One way to interpret the fractional AdWords as a discrete AdWords instance is to view each ad j as $Q \to \infty$ infinitesimal sub-slots, and each advertiser i has bid v_i^j/Q for each such sub-slot. Under this view, we can derive what the optimal algorithm must look like. Suppose the algorithm chooses the allocation $(p_0^j, p_1^j, \ldots, p_{m-1}^j)$ for ad j, then by considering the MSVV allocation of the "last" infinitesimal sub-slot of j, the following optimality condition should hold: all i's where $p_i^j > 0$ must achieve the maximum $v_i^j \cdot (1 - e^{-(s_i^j - (v_i^j p_i^j / B))})$ value (among all advertisers).

Fractional relaxations may sometimes lose some combinatorial flavor of the original problems, but it is meaningful (and common) to study them in algorithm design: often they can lead to randomized algorithms or can be rounded to integral solutions. For Adwords, the fractional version retains the essential characteristics, and the optimal algorithms and hard distributions in fractional and discrete cases have a strong connection. The advantage is that the fractional relaxation leads to easy evaluation of the competitive ratio without sampling the policy, which makes the program fully differentiable, and this is very useful for our training framework.

4.1 Demonstration of Results

In this section we demonstrate some qualitative behaviors of our training result. We fix $m = 9, n = 3$ and a common budget $B = 3$. We defer the representation details to Appendix C.1 and extra empirical results to Appendix C.2. About 10 hours of training is sufficient to achieve a competitive ratio of ≈ 0.7327, which is within 10^{-2} of the theoretically best value achievable for this problem.

We start with a brief discussion of the optimal algorithm and input distribution so we can better interpret our results. At a high level, there are two intuitive aspects of the AdWords problem (in the discrete case): the *greedy strategy* and the *balance strategy*.

Greedy strategy allocates an ad slot to the advertiser with the highest bid for the ad when their remaining budgets are equal. This is a natural strategy to maximize revenue.

Balance strategy allocates an ad slot to the advertiser with the largest remaining budget when all advertisers have the same bid for the ad. This is also natural to avoid depleting the budget of any advertiser, since we may run into a future ad where only the advertiser with depleted budget has high bid for the ad. The "scaled bid" used in the MSVV algorithm (even for the fractional version) can be viewed as a careful interpolation between two strategies when the advertisers' bids and remaining budgets differ.

We first demonstrate that our trained algorithm captures the "greedy" aspect in Figure 1. For example, in the leftmost plot, we fix the remaining budgets of all advertisers at 2, the last two advertisers' bid for an ad at 0.25 (the vertical dotted line), and vary the bid x of advertiser 0 to plot the fractional allocations returned by the algorithm for different x values. We can see that the algorithm correctly allocates more to advertiser 0 as the bid x increases, and also correctly allocates roughly the most to the advertiser with the highest bid. We observe consistent behavior for other threshold values in the next 3 plots (and more comprehensive plots in the appendix).

We demonstrate the "balance" aspect of our trained algorithm in Figure 2. In the leftmost plot, we vary the remaining budget of advertiser 0 while fixing the other advertisers' budget at 0.9, and all advertisers' bids are 1. The algorithm correctly allocates more the advertiser 0 when he/she has more remaining budget, and also correctly gives roughly the most fractional share to the advertiser with largest remaining budget. Again, the observation is consistent in the other plots (and more in the appendix). We note that different from the discrete case where one should expect a sharp transition of allocation around the threshold, the smooth curves observed in our plots actually agree with the optimal algorithm in the fractional case. For a concrete example (see Figure 3), we consider the case where the remaining budgets of the advertisers are all fixed at 1, and advertisers have

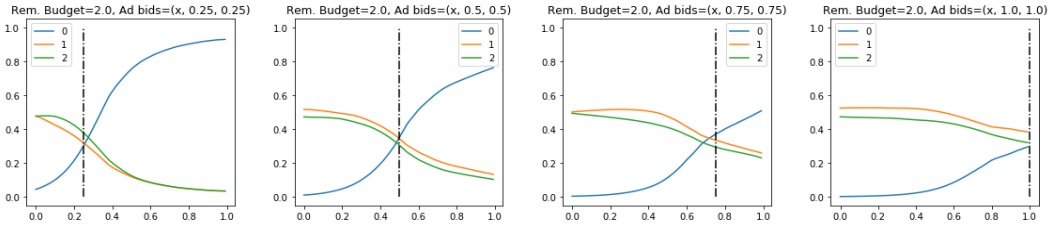

Figure 1: Each plot shows the fractional assignment of a new ad to the three different advertisers indexed $\{0, 1, 2\}$ while keeping the remaining budget equal across all advertisers. The bids of the current ad are (x, c, c), where x changes with the x-axis and c is fixed in each plot.

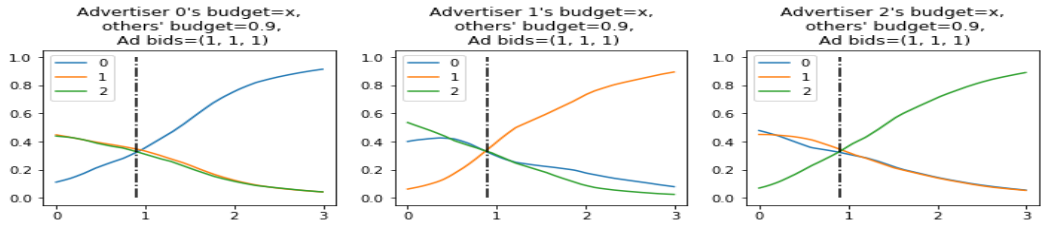

Figure 2: Each plot shows the fractional assignment of a new ad to the three different advertisers indexed $\{0, 1, 2\}$. The ad has a common bid of 1 to all advertisers. The remaining budgets change for one advertiser along the x axis while staying the same for all other advertisers.

bids $(x, 0.5, 0)$ for varying x values. These parameters are chosen for simpler calculation. Let's consider the optimal algorithm's allocation in this case. From our discussion of the optimal fractional algorithm (immediately following the problem definition of fractional AdWords), we know that the optimal algorithm will split the ad between advertiser 0 and 1 (as advertiser 2 has bid 0), and if the fraction allocated to advertiser 0 is $t \in [0, 1]$, and $1 - t$ to advertiser 1, we must have

$$x \cdot (1 - e^{-\frac{1-t}{3}}) = 0.5 \cdot (1 - e^{-\frac{1-(1-t)}{3}})$$

If we use the approximation of $e^{-x} \approx 1 - x$ for small x, the rough calculation gives t as a function of x where $t = \frac{2x}{2x+1}$. We plot this as the dotted (red) curve in Figure 3, and we observe that the fractional allocation to advertiser 0 (as a function of x) by our trained algorithm is fairly consistent with the optimal.

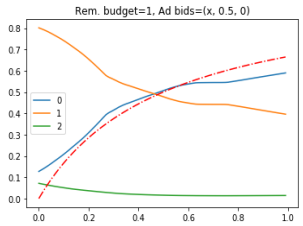

Figure 3: The allocation of the trained algorithm where every advertiser has budget 1, the bids are $(x, 0.5, 0)$ where x varies. The dotted (red) curve depicts the approximate behavior of the optimal algorithm for bidder 0.

On the adversary side, in the specific case of 3 advertisers and 9 ad slots, an adversarial input distribution (i.e. a minmax column strategy in the game theoretical discussion of Section 2.1) is the uniform distribution over the 6 matrices which are the column permutations of the transpose of the following matrix, where ads arrive from top to bottom (left-to-right in the transposed matrix below).

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In the previous work by Kong et al. (2018), they showed that a mixture of an adversarial input distribution and a high-entropy (i.e. noisy) distribution gives a sufficiently good (pre-determined) training set. We plot a set of random samples from the experience array at the end of our training in Figure 8 in Appendix C.2. It is clear that without any expert input, the self-generated input instances during training capture enough structure of such an expert crafted mixture. We find this highly non-trivial.

5 Conclusion and Future Directions

In this work we explore the idea of using adversarial training for learning optimal algorithms from first principles. We propose the YaoGAN technique and use it to recover several near-optimal online algorithms without using any problem-specific domain knowledge about the problem, courtesy of adversarial training. Our results suggest YaoGAN is effective in learning algorithms that are robust to worst-case inputs. We also show an important limitation of our approach: The training is as effective as the search for best response inputs. When this search can be effectively performed the trained algorithm seem to be near optimal; if the search is approximate, the algorithms are also approximate.

The ultimate goal of our investigation is to gain crucial insight into interesting open problems in previously unexplored ways. While this goal is currently out of reach, we hope our research opens up such a path. On a less ambitious note, our future directions include (1) to use adversarial training to develop algorithm independent of problem size for hard problem such as AdWords, (2) train offline algorithms, and (3) solve general online packing/covering problems as an immediate extension of our techniques for AdWords.

REFERENCES

Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. *Theory of Computing [electronic only]*, 8, 01 2012. doi: 10.4086/toc.2012.v008a006.

Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Edge weighted online windowed matching. In *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019.*, pp. 729–742, 2019. doi: 10.1145/3328526.3329573. URL https://doi.org/10.1145/3328526.3329573.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998. ISBN 978-0-521-56392-5.

Craig Boutilier and Tyler Lu. Budget allocation using weakly coupled, constrained markov decision processes. In *UAI*, 2016.

Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009. doi: 10.1561/0400000024. URL https://doi.org/10.1561/0400000024.

Niv Buchbinder, Joseph Seffi Naor, et al. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6348–6358, 2017.

Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 170–181. Springer, 2018.

John Dickerson, Karthik Sankararaman, Aravind Srinivasan, and Pan Xu. Allocation problems in ride-sharing platforms: Online matching with offline reusable resources. 11 2017.

Paul Duetting, Zhe Feng, Harikrishna Narasimhan, David C. Parkes, and Sai Srivatsa Ravindranath. Optimal auctions through deep learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 1706–1715, 2019. URL http://proceedings.mlr.press/v97/duetting19a.html.

Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, COLT '96, pp. 325–332, New York, NY, USA, 1996. ACM. ISBN 0-89791-811-8. doi: 10.1145/238061.238163. URL http://doi.acm.org/10.1145/238061.238163.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Bala Kalyanasundaram and Kirk R. Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1):319 – 325, 2000. ISSN 0304-3975. doi: https://doi.org/10.1016/S0304-3975(99)00140-1. URL http://www.sciencedirect.com/science/article/pii/S0304397599001401.

A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pp. 244–254, Oct 1986. doi: 10.1109/SFCS.1986.14.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Weiwei Kong, Christopher Liaw, Aranyak Mehta, and D Sivakumar. A new dog learns old tricks: Rl finds classic optimization algorithms. 2018.

Wouter Kool, Herke van Hoof, and Max Welling. Attention solves your tsp, approximately. *stat*, 1050:22, 2018.

Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5), October 2007. ISSN 0004-5411. doi: 10.1145/1284320.1284321. URL http://doi.acm.org/10.1145/1284320.1284321.

Aranyak Mehta et al. Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science*, 8(4):265–368, 2013.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pp. 222–227, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.24. URL https://doi.org/10.1109/SFCS.1977.24.

A No regret dynamics and convergence to the MinMax

There is a rich line of work on no regret dynamics (Freund & Schapire, 1996; Arora et al., 2012) which lies at the center of various fields including game theory, online learning and optimization. In the context of two-player zero-sum games, this gives a method to approximately solve the value of the game without knowing the optimal column player strategy (i.e., the adversarial input distribution) in advance. We extend the notation to use $V(\boldsymbol{p}, \boldsymbol{q})$ to denote $\boldsymbol{p}^T V \boldsymbol{q}$.

More specifically, we can let the algorithm player maintain a randomized algorithm $\boldsymbol{p}^t$ over iterations $t = 1, \ldots, T$. In each step the adversary plays the best response $\mathcal{I}^t$ (i.e. the worst-case input against the algorithm at that iteration). The algorithm $\boldsymbol{p}^{t+1}$ is then computed from $\boldsymbol{p}^t$ and $\mathcal{I}^t$ using a specific no-regret dynamic such as Hedge (Freund & Schapire, 1996). Define

$$\tilde{\boldsymbol{p}} = \arg\min_{\boldsymbol{p}^t} V(\boldsymbol{p}^t, \mathcal{I}^t), \qquad \tilde{\boldsymbol{q}} = \frac{1}{T} \sum_t \mathcal{I}^t.$$

Classic results give us that the payoff of the pair $\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{q}}$ converges to the optimal minmax value for sufficiently large T.

YaoGAN loosely follows this approach. It maintains a randomized algorithm $\boldsymbol{p}^t$ and searches for the best response $\mathcal{I}^t$. However, it does not follow the no-regret dynamics of prior algorithms, but rather uses simple gradient descent to optimize the payoff of the algorithm on $\mathcal{I}^t$. We avoid going into formal details since we use the theoretical guarantees mostly to draw insights.

B SKI RENTAL

In this Section, we derive a uniform algorithm for the classic ski rental problem (defined in Section 2).

A randomized algorithm for the ski rental problem is represented by a (marginal) probability distribution $p_1, p_2, \ldots$, with the semantics that for each $x = 1, 2, \ldots$, p_x denotes the probability that the we rent skis for the first (up to) $x - 1$ days and buy skis on day x, if the season extends that far. For reference, we note that the optimal choices are given by $p_i = (\frac{B-1}{B})^{B-i}\frac{c}{B}$ for $i = 1, \ldots, B$, where $c = \frac{1}{1-(1-1/B)^B}$ so these probabilities sum to 1. It has a competitive ratio of $\frac{1}{1-(1-1/B)^B}$, which tends to $\frac{e}{e-1} \approx 1.582$ as $B \to \infty$ (Karlin et al., 1986).

Our goal here is to learn an algorithm that works for all values of B and N values. (If we only care about fixed B and N, Section A gives a direct and provable way. Let $\boldsymbol{p}^t = \boldsymbol{p}_1^t, \ldots, \boldsymbol{p}_B^t$ describe the algorithm (updated via no-regret dynamics), and let the $\mathcal{I}^t$'s be computed by exhaustive search. This works remarkably well in practice, e.g., for $B = 10$ and $N = 40$, this procedure converges in minutes, and achieves a competitive ratio that is only 10^{-3} off from the best possible value.)

For the ski rental problem, we apply a simpler variant of the YaoGAN paradigm. Specifically, we train a deep neural network to represent the algorithm, but for the adversary, we don't resort to deep neural networks yet. This is because the input space to our algorithm network is still fairly straightforward, and simple methods are quite effective to come up with good "best responses" for training.

As B and N can be (asymptotically) large, we need to work with a continuous space, where every parameter is essentially normalized by N. In this setting, time goes over the range $[0, 1]$ indicating what fraction of the time horizon has passed. At time $\alpha \in [0, 1]$, if we are still skiing, the cost of renting from the beginning of time up to this point is captured by α. The cost of buying is captured by a real number $\beta \in [0, 1]$, which is the fraction of the time one needs to go skiing in order to make buying the skis worthwhile. If we stop skiing at point α of the continuous timeline, the optimal offline cost is $\min(\alpha, \beta)$.

Applying the YaoGAN paradigm: On the algorithm side, we train a deep neural network that takes $\alpha, \beta \in [0, 1]$ as inputs, computes the CDF of a probability distribution p_β, and outputs the value of the CDF at α. As for the architecture of the algorithm neural network, we use 50 Gaussian kernels $N(x_i, \sigma)$ for $i = 1, \ldots, 50$ with fixed means (i.e., x_i's equally spaced over $[0, 1]$) and standard deviation $\sigma = 2/50$, and take a weighted average of these Gaussian kernels as the distributions learned by our neural network. In particular, given any (α, β) as input, the neural network uses β to derive weights w_i's over the Gaussian kernels (where $\sum_i w_i = 1$), and returns the weighted sum of the CDF of these fixed Gaussian kernels at point α as the output of the algorithm. We use a fairly generic neural network structure to learn the kernel weights, e.g. densely connected layers with ReLU activation, and Adam optimizer to update the neural network.

On the adversary side, in each training iteration, we randomly sample a small number of values for β, and for each sampled β we use an ϵ-net over $[0, 1]$ as values of α with ϵ chosen to be 0.01. We approximately evaluate the competitive ratios (up to certain precision) of the current algorithm for these (α, β) instances, and train the algorithm neural network with the instance where it gets the worst competitive ratio. The approximation is due to not really taking a integral over the learned cdf but rather a step function approximation of it.

We demonstrate the trained algorithm in Figure 4 by instantiating it at various values of B and N. The result is very close to the optimal strategy in all cases, and indicates that our theoretical insights from Section A largely extends to more general settings. We also note here that the competitive ratio achieved by this algorithm converges approximately to 1.59 within a few minutes of training, about 0.01 worse than the optimal value (1.582).

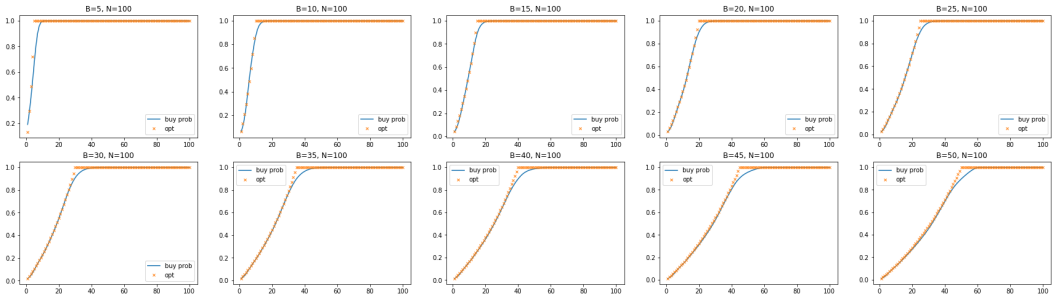

Figure 4: The learnt algorithm instantiated at various (B, N) values as CDF of buying over time (blue curve). The orange x's correspond to the optimal strategy in each case.

C ADWORDS

C.1 ALGORITHM AND ADVERSARY REPRESENTATIONS

In this work we train an algorithm with a fixed number of advertisers $m = 3$. Although this is a rather limited setting, it captures the essential structure of the problem, and we can observe very non-trivial behavior from both the algorithm side and the input side. There does exist a representation for a uniform algorithm, i.e., one that works for arbitrary number of m (e.g. see Kong et al. (2018)), and we leave it for future work to extend our framework in that set-up.

Upon the arrival of an ad, the neural network for the algorithm takes as input a vector containing the bids of the 3 advertisers for the current ad and their remaining budgets. The neural network outputs a probability distribution $(p_0, \ldots, p_3)$ indicating the fractional allocation of the ad (with p_3 being the fraction not given to anyone). The network itself is composed of 4 hidden and ReLU-activated layers with 500 neurons succeeded by a softmax output layer. The training environment feeds each input instance to the algorithm sequentially, and updates the total revenue and advertisers' remaining budgets along the way with the algorithm outputs.

On the adversary side, we consider inputs with a fixed number of ad slots $n = 9$, and initial budget $B = 3$ (both can be changed to arbitrary values if needed). The network itself takes n_{noise} Gaussian variables, acts upon them using 4 hidden ReLU-activated layers with batch normalization and outputs a $m \times n$ sigmoid-activated matrix.

It is important to note that in this problem (as opposed to the ski-rental problem) there is no easy exhaustive strategy for the adversary side, even with discretization of the weights on the edges (as there are 27 numbers in an input graph).

C.2 ADDITIONAL PLOTS FOR ADWORDS

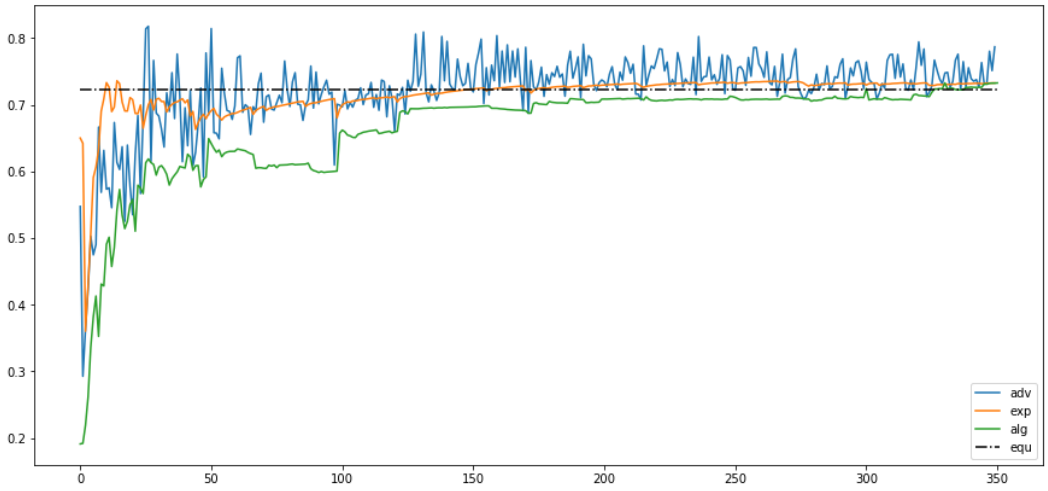

Figure 5: Competitive ratio at snapshots taken just before the adversarial network is restarted. The blue (adv) line represents the CR of the fully trained adversarial network before its restart. The orange (exp) line at snapshot i represents the CR of the best response from the experience array at snapshot i versus the trained algorithm at snapshot i. The green (alg) line at snapshot i represents the CR of the best response from the entire experience array (from the entire training) versus the trained algorithm at snapshot i.

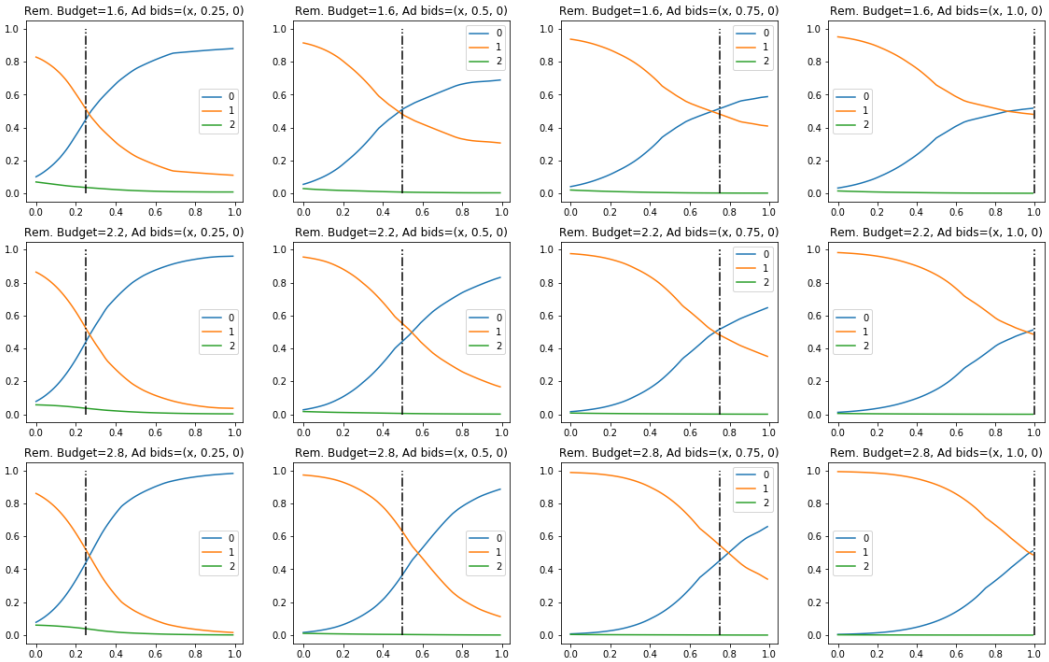

Figure 6: Each plot shows the fractional assignment of a new ad to the three different advertisers indexed $\{0, 1, 2\}$ while keeping the remaining budget equal across all advertisers. The bids of the incoming ad slot are $(x, 0.25, c)$, where x changes with the x-axis and c is common for every plot in the same column. Each row features advertisers with a common remaining budget.

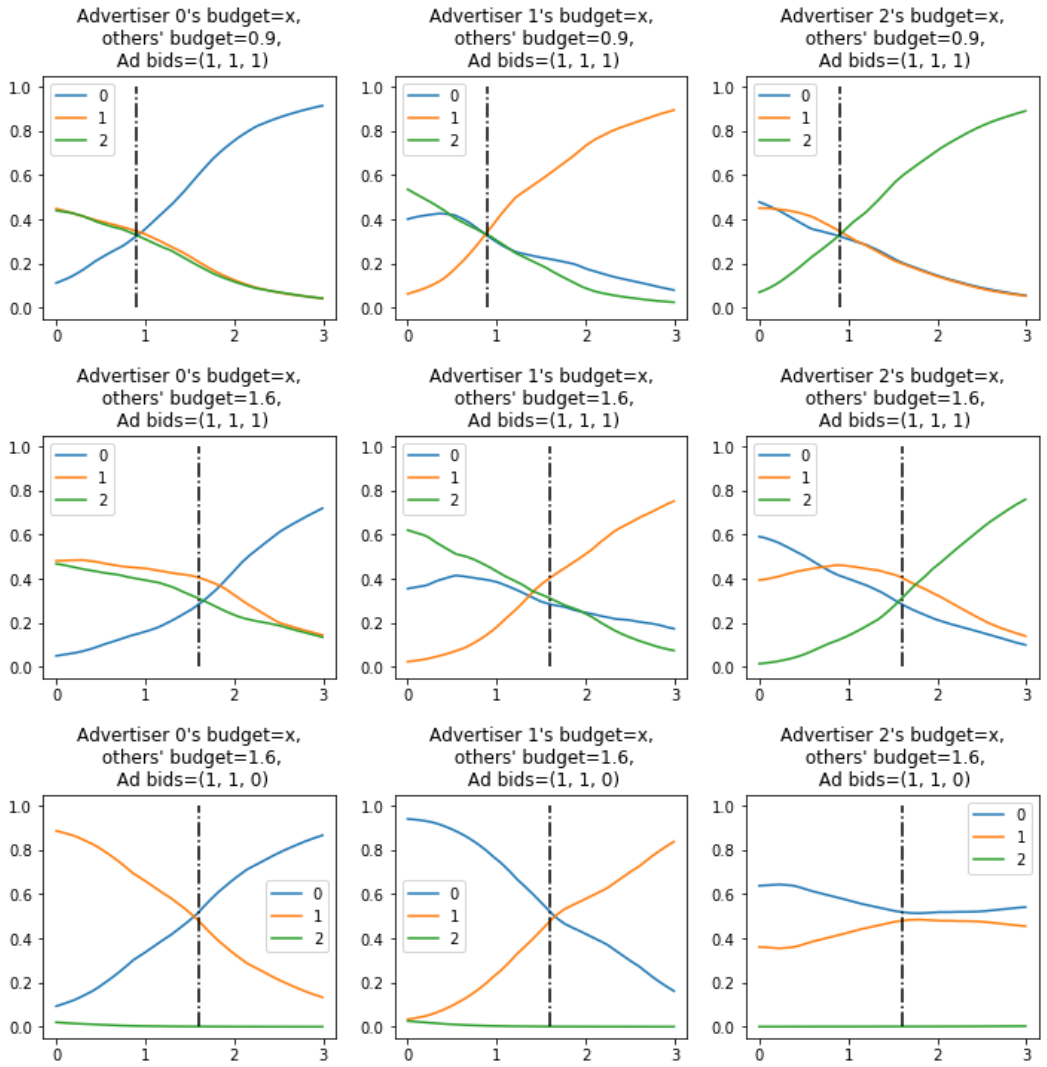

Figure 7: Each plot shows the fractional assignment of a new adword to the three different advertisers indexed $\{0, 1, 2\}$. The ad slot has a common bid of 1 to all advertisers. The remaining budgets change for one advertiser along the x axis while staying the same for all other advertisers.

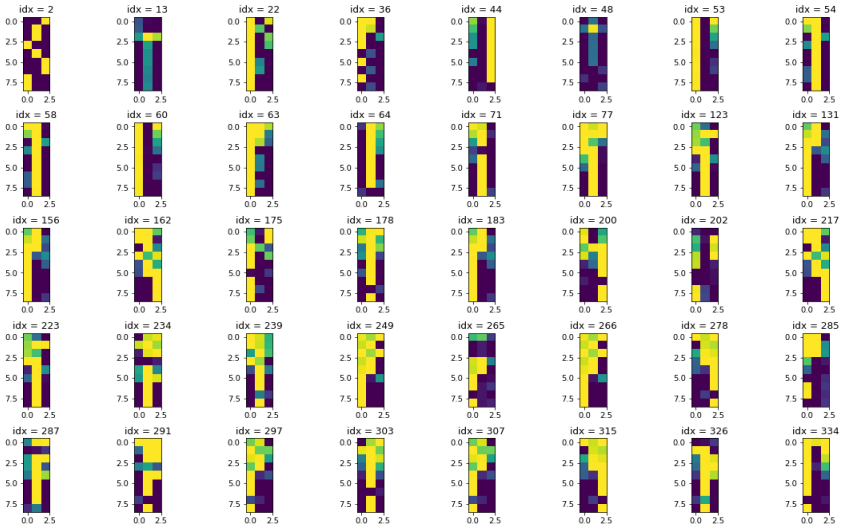

Figure 8: A set of random samples from the experience array after training. The jth row and i^{th} column represent the bid of advertiser i for ad slot j. Darker color represents a bid close to 0, while brighter color represents a bid close to 1.