Nishant Kumar

ADX RL Research Summary

1 Project Idea and Motivation

The project I worked on was applying reinforcement learning to (small subgames of) the AdX game [1]. The AdX game crudely models the digital advertising domain: advertisers buy *impression opportunities* [2] from websites, where the objective of each advertiser is to minimize spend and the objective of each website is to maximize revenue. This buying and selling is usually done through an *ad exchange* (e.g. Google's AdX), which canonically holds digital auctions; the bidders are the advertisers and the goods being sold are the impression opportunities. In the AdX game, each player plays the role of an *advertiser liaison* [3]; advertisers procure *ad campaigns* [4] to liaisons, who are responsible for fulfilling the campaign within a certain time frame [5]. The goal of each player is to learn what bids to place [6] in order to maximize their payout by the end of the game.

The AdX game is interesting to tackle from an RL perspective because it poses several interesting properties and challenges:

- Stochasticity in the game can be a consequence of both the randomness of an impression opportunity's demographic(s) and the randomness of each player's strategy (i.e. players playing mixed strategies).
- Determining an optimal policy via *planning* is difficult because determining a model *a priori* is difficult or infeasible [7].
- The domain offers continuous control, as bids are real-valued. Furthermore, the domain can be highly dimensional, as there can be many types of demographics. Consequently exhaustive search of the space is often infeasible or intractable, thus smart exploration and/or generalization is required.
- The domain can be explored from both a single-agent perspective and a multi-agent perspective.

[1] This project was a collaboration with Prof. Amy Greenwald and her PhD student Enrique Areyan.

[2] An *impression opportunity* is an opportunity to show an ad to a user of a website.

[3] An *advertiser liaison* is an intermediary which handles bidding on behalf of an advertiser. Alternatively, this is sometimes referred to as an *ad network*, though that term is easy to confuse with *ad exchange*.

[4] An *ad campaign* is a contract between an advertiser and a liaison which specifies how many impression opportunities the advertiser wants to capture (i.e. *reach*), the *budget* of the advertiser (equivalently the payout to the liaison), and the *target* demographic(s) of interest. A *demographic* represents salient characteristics of a user (e.g. a "Male" user, a "young" user, a "rich" user, etc.)

[5] The when and how of campaign procurement, campaign fulfillment, and campaign payout can vary, thus there are many variants of the "base" game. The variant explored in this project is outlined in the problem definition section.

[6] In other words, which demographic(s) to target and how much to offer for each impression opportunity of the given demographic.

[7] Players are not given the distribution from which impression opportunities are drawn. Furthermore, in game variants in which players are procured campaigns at random, the players are also not given the distribution from which campaigns are drawn.

2 Existing Background and State-of-the-Art

Technology for matching advertisers to ad slots is often done by companies that also run ad exchanges. Alternatively, there are companies that also offer ad network technology. Furthermore, some relevant published work includes:

- Kong et al. (2019) achieved moderate success applying standard REINFORCE to the AdWords problem. The AdWords problem is not quite the same as the AdX problem (e.g. lack of explicit auction), but it shares some core similarities (an allocation of advertisers to ad slots is to be found, based on maximizing some objective).
- Zhang et al. (2014) studies bid optimization for real-time bidding. They formulate programmatic bidding as a function optimization problem and derive simple bidding functions for real-time use.
- Linden (1993) highlights the challenges of learning discontinuous value functions and outlines some approaches (e.g. use discontinuous VFAs, design relative value functions). In particular relevance to our work, it claims that "backpropagation nets have severe difficulties to simply represent discontinuous functions" [8].
- Asadi et al. (2020) propose a deep NN with a radial basis function output layer for learning value functions in continuous control domains. The deep RBF is interesting to us for its ability to both support universal function approximation and scale to large continuous action spaces.

3 Problem Definition

The AdX game variant studied in this project is the **1-campaign, N-day** variant. This variant works as follows:

- The game lasts for N days.
- At the beginning of the game, each agent (i.e. player) is assigned a campaign at random from an underlying distribution [9]. The goal of each agent is to maximize its profit by fulfilling its one campaign over the N days of the game.

[8] Something not discussed in my paper is the difficulty we had with learning discontinuous value functions; For AC Q with baseline, we plotted the learned Q function against the actual reward function. Even for a 1 day game of experimental setup 1, various NNs we tried (1 dense and 4 dense) were poor approximators. Our results imply that either the net needs to be deeper or it needs much more data (or both). Regardless, the policy network learned far faster than the Q value network did. This meant bias was a huge problem for AC methods.

[9] The campaign distribution is known to the game engine, but unknown to the players.

- Each day, k newly-drawn impression opportunities are for sale, where k is drawn uniformly at random from [num_items_min, num_items_max). The demographic of each impression opportunity is assigned at random from an underlying distribution [10]. At the end of the day, unsold opportunities are discarded (i.e. they are **not** carried to the next day).
- Each day, each agent places a *bid bundle*, which specifies how much they offer per impression opportunity per each type of demographic in the game. All agents place their bid bundles simultaneously and only once per day. Once submitted, bids cannot be canceled or replaced.
- Each day, an auction is run to sequentially sell the k impression opportunities to the bidders (i.e. agents). Each winner must pay for their impression according to the auction rules [11].
- At the end of the N days, each agent is payed. The payment amount is a function of the budget and the total (relevant) impressions won [12].
- Note that an agent only has incentive to win impression opportunities which match its campaign's target demographic (this is a *relevant* impression) [13]; winning impressions that don't match is not useful, as such impressions do not count toward fulfilling the campaign.

A formal RL definition is provided in the Appendix.

4 Methodology

In this project, we focused on applying, analyzing, and optimizing policy gradient methods to small subgames of the AdX game. Specifically, we study single-agent scenarios by having one learning agent and fixing the strategy of the other agents. We developed and used the Derby codebase for this purpose. See the Appendix for more detail about Derby.

We focused on the following algorithms:

- REINFORCE
- REINFORCE with baseline

[10] The demographic distribution is known to the game engine, but unknown to the players.

[11] For this project, we used a first-price auction.

[12] For this project, we used a linear payout: $\frac{\text{impressions won}}{\text{reach}} \times budget$

[13] The definition of *match* also varies from game variant to game variant. For this project, match means logical subtype, for example: an imp. opp. with demo {Male} and an imp. opp. with demo {Male, rich} both match a campaign with target demo {Male} because both {Male} and {Male, rich} are logical subtypes of {Male}.

- Actor-Critic Q with baseline
- Actor-Critic TD (contains baseline)
- Actor-Critic SARSA with baseline

Where the policy network learns a Gaussian distribution.
We coded and tested different versions of each algorithm:

- Untuned/Semi-tuned version
- Tuned version v2

See the Appendix for more detail about the algorithms, network architectures, and algorithm tuning.
We tested these algorithms in the following experimental setup:

- Setup 1
  - The campaign set C contains only one campaign: C ={{reach: 10, budget: \$100, target: {Female}}}. It has 100% chance of being drawn at the beginning of the game.
  - The demographic set D contains only one demographic: $D = \{\{\text{Female}\}\}$. Each impression opportunity created has 100% chance of being said demographic. Exactly 1 impression opportunity is created each day of the game.
  - There are two agents in the system: one RL agent and one fixed strategy agent. The fixed strategy agent uses a fixed bid policy: every day, the agent bids (on its campaign's target) \$5 per impression, up to \$5 total that day.

  This setup is interesting to study because it represents the simplest, non-trivial subgame of the problem domain; there is no stochasticity, there is only 1 impression per day, the RL agent is in direct competition with the other agent, and the other agent only bids \$5 each day for said impression. Effectively, the RL agent must learn to bid just above \$5 every day of the game.

5 Preliminary Results

The results presented here are for the untuned/semi-tuned versions of the PG algorithms.
For more context:

- **game_length** represents how many days the game lasts.

- **num_trajs** represents the number of trajectories (i.e. batch size) used for each epoch of training.
- **num_epo** represents the number of iterations of training.
- For visual clarity, std. dev. intervals are not plotted on the graphs.
- The maximal reward an agent can achieve in the 5-day game is 25.

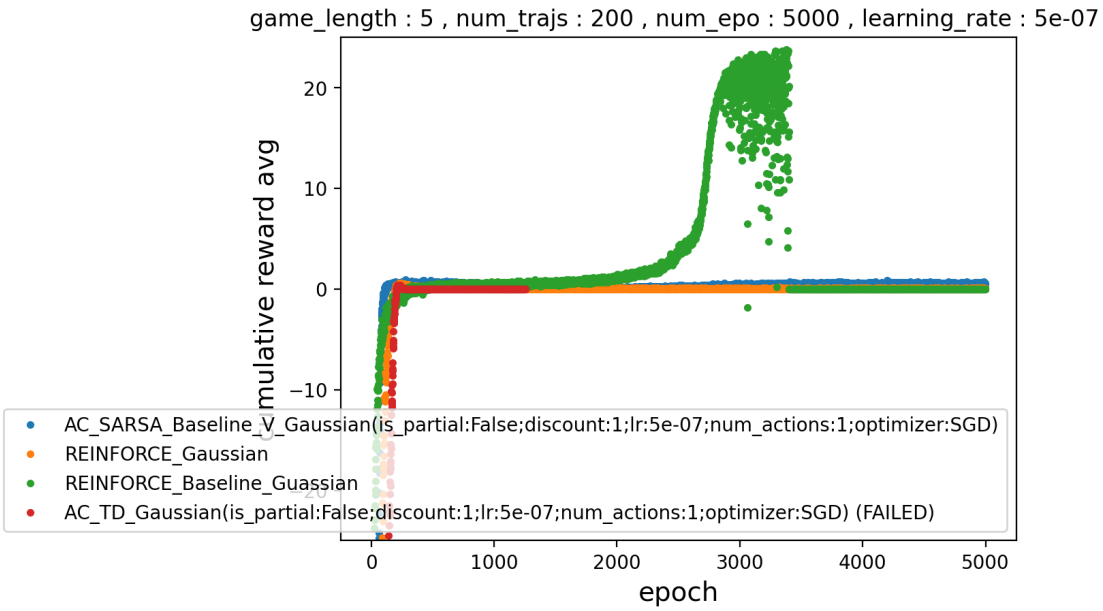

Figure 1: Untuned/Semi-tuned PG algos in 5 day game

While at first glance, the results in 1 look abysmal, there is more nuance than meets the eye:

- The rewards of the initial epochs for all algorithms is actually worse than it appears; for visual clarity, the y-axis min is roughly -30. The actual initial avg. cuml. rewards for the algorithms is roughly in the range $[-200, -100]$. In other words, the algorithms spend a lot of money initially.
- While it looks like the AC methods and REINFORCE never get off the ground, they actually have different reasons. At this learning rate, REINFORCE is very slowly climbing toward higher reward. In fact, in our other results (not in this paper) we see it climbs toward 20 avg. cuml. reward (though in the span of 50k epochs, not 5k). The AC methods, however, show an initial climb toward positive reward, but then **zero collapse** [14] [15].

[14] see Appendix for more about zero collapse
[15] See the Tuned Results section for a little more info as to why AC methods zero collapse

- This particular plot of REINFORCE with baseline is somewhat of an outlier. In this plot, there is an inflection point about which it climbs very quickly toward higher reward, then quickly destabilizes and zero collapses. I suspect something odd must have happened this run, as in our other results we usually see REINFORCE with baseline slowly and stably climb toward 20+ avg. cuml. reward (though in the span of 50k epochs, not 5k), for learning rates between $[7e-7, 2e-6]$. Only for learning rates higher than $2e-6$ do we see the zero collapse behavior exhibited here. One possibility is that I incorrectly increased the learning rate of the value network of this run.

In general, the untuned/semi-tuned versions of the PG algorithms exhibit large negative initial avg. cuml. reward and take many epoch to get to 15+ avg. cuml. reward (from our other results, this occurs around 15k epochs).

6 Tuned Results

The results presented here are for the v2 versions of the PG algorithms. See section `Algorithm Tuning` in the Appendix for details about how the algorithms were tuned for better results.

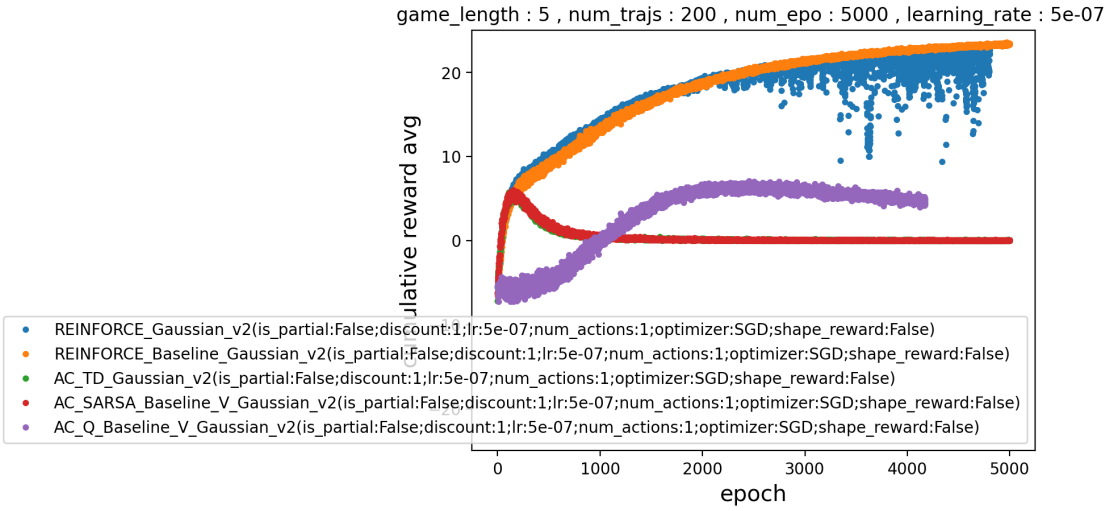

Figure 2: Tuned PG algos in 5 day game

With the maximal reward of 25 in mind, we can see that REINFORCE and REINFORCE w/ baseline perform quite well; they rise fast and they climb toward the optimal reward. REINFORCE, however, starts to exhibit some destability about halfway through.

The AC algorithms show interesting results; TD and SARSA both rise fast, but quickly exhibit **zero collapse** [16]. Similarly, AC Q initially rises toward positive reward, but eventually

[16] See Appendix for more about zero collapse

trends down (likely going to zero collapse). I won't go into detail in this paper as to why the AC methods exhibit zero collapse, but at a high level: as alluded to in the Appendix, the bias of AC methods causes the policy to quickly climb up the reward cliff and subsequently jump off. By the time the bias is corrected, the policy is already deep in the flat-and-zero reward region.

7 References

Weiwei Kong, Christopher Liaw, Aranyak Mehta, & D. Sivakumar (2019). A new dog learns old tricks: RL finds classic optimization algorithms. In *ICLR*, 2019.

Zhang, W., Yuan, S., & Wang, J. (2014). Optimal Real-Time Bidding for Display Advertising. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1077–1086). Association for Computing Machinery.

Linden A. (1993) On discontinuous Q-Functions in reinforcement learning. In: Jürgen Ohlbach H. (eds) GWAI-92: Advances in Artificial Intelligence. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), vol 671. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0019005

Kavosh Asadi, Ronald E. Parr, George D. Konidaris, & Michael L. Littman. (2020). Deep RBF Value Functions for Continuous Control.

8 Appendix

8.1 Formal RL Definition

8.1.1 Variable Definitions

- Let m be the number of agents in the system.
- Let $a_t^{(i)}$ represent the action agent i takes at time t, for $i \in [1, \ldots, m]$.
- Let $s_t \in S$ represent a full, joint state for the entire system (i.e. all agents' states), at each time step t.
- Let $r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)})$ be the reward function for agent i.
- Let π_{θ_i} be the policy for agent i.
- Let $p_\theta(\tau)$ be the probability of seeing joint trajectory τ:

$$p_\theta(\tau) := p_\theta(s_0, a_0^{(1)}, \ldots, a_0^{(m)}, s_1, a_1^{(1)}, \ldots, a_1^{(m)}, \ldots, s_{T-1})$$

  and more generally:

$$p_\theta(\tau_t) := p_\theta(s_t, a_t^{(1)}, \ldots, a_t^{(m)}, s_{t+1}, a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)}, \ldots, s_{T-1})$$

- Let $J(\theta_i)$ be the (undiscounted) objective function for agent i:

$$J(\theta_i) := \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right] := \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r^{(i)}(\tau) \right]$$

- Let value functions $Q^{\pi_{\theta_i}}$ and $V^{\pi_{\theta_i}}$ for agent i with policy π_{θ_i} be defined as follows:

$$V^{\pi_{\theta_i}}(s_t) := \mathbb{E}_{\tau \sim p_\theta(\tau_t)} \left[\sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \ldots, a_{t'}^{(m)}) \middle| s_t \right]$$

$$Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) := \mathbb{E}_{\tau \sim p_\theta(\tau_t)} \left[\sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \ldots, a_{t'}^{(m)}) \middle| s_t, a_t^{(1)}, \ldots, a_t^{(m)} \right]$$

8.1.2 Assumptions

- Assume users (i.e. impression opportunities) are exogenous.
- Assume campaign distributions are independent.

- Assume all instances of the game use a pre-determined fixed set of campaigns C.
- Full observability equivalently means "every agent sees other agents' states". Without this assumption, state aliasing can occur.
- Assume agents must take actions independently (cond. indep. based on state) and simultaneously at each time step t (i.e. no pre-determined collusion/coordination). Then:

$$
\begin{aligned}
p_\theta(\tau) &= p_\theta(s_0, a_0^{(1)}, \dots, a_0^{(n)}, s_1, a_1^{(1)}, \dots, a_1^{(m)}, \dots, s_{T-1}) \\
&= p(s_0) \prod_{t=0}^{T-2} \pi_\theta(a_t^{(1)}, \dots, a_t^{(m)} | s_t) p(s_{t+1} | s_t, a_t^{(1)}, \dots, a_t^{(m)}) && \text{(Markov prop.)} \\
&= p(s_0) \prod_{t=0}^{T-2} \pi_\theta(a_t^{(1)}, \dots, a_t^{(m-1)} | a_t^{(m)}, s_t) \pi_{\theta_m}(a_t^{(m)} | s_t) p(s_{t+1} | s_t, a_t^{(1)}, \dots, a_t^{(m)}) \\
& && \text{(product rule)} \\
&= p(s_0) \prod_{t=0}^{T-2} \pi_\theta(a_t^{(1)}, \dots, a_t^{(m-1)} | s_t) \pi_{\theta_m}(a_t^{(m)} | s_t) p(s_{t+1} | s_t, a_t^{(1)}, \dots, a_t^{(m)}) \\
& && \text{(assume cond. indep.)} \\
&\vdots \\
&= p(s_0) \prod_{t=0}^{T-2} \left(\prod_{j=1}^{m} \pi_{\theta_j}(a_t^{(j)} | s_t) \right) p(s_{t+1} | s_t, a_t^{(1)}, \dots, a_t^{(m)}) && \text{(repeat } m \text{ times)}
\end{aligned}
$$

8.1.3 Game Definitions

Markov Game
A fully-observable Markov Game $\langle S, A, R, T, \gamma \rangle$ for this game variant would be:

- Let $S = \prod S^{(i)}$, where state $s^{(i)} \in S^{(i)}$ for agent i would be:
  - **campaign**: a campaign initially assigned at the beginning of the game. The agent should attempt to fulfill the campaign by the end of the game (i.e. after N days). A campaign consists of:
    - **target** (a.k.a market segment): a demographic(s) to be targeted.
      There are 26 market segments in this game, corresponding to combinations of {Male, Female} x {HighIncome, LowIncome} x {Old, Young}. A market segment might target only one of these attributes (for example, only Female) or two (Female_Young) or three (Female_Old_HighIncome).
    - **reach**: the number of impression opportunities to capture in the target market segment.

    - ∗ **budget**: the amount of money the advertiser will pay if the campaign is fulfilled.
  - **spend**: the amount spent so far in fulfilling this campaign [17].
  - **impressions**: the number of (relevant) impressions achieved so far. Note that impressions ≤ reach.
  - **day**: the day of the game (i.e. time t).

- Let $A = \prod A^{(i)}$, where action $a^{(i)} \in A^{(i)}$ for agent i would be:
  - $\{(b_1, u_1), \ldots, (b_{26}, u_{26})\}$
    Where $b_j \in \mathbb{R}$, $u_j \in \mathbb{R}$.
    Where each subaction (b_j, u_j) is the **bid per impression** and **total spend limit** for market segment j.
    Note that this represents a bid bundle.

- Let $r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)})$ for each agent i be:

$$r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) := \text{how much earned on day } t \; - \; \text{how much spent on day } t$$

  A couple of subtleties here [18] [19].

- Let T be a probability function that represents the probability of transitioning from state s_t to state s_{t+1} after bids have been placed and the auction has been resolved. Namely:

$$T(s_t, a_t^{(1)}, \ldots, a_t^{(m)}, s_{t+1}) := p_\theta(s_t, a_t^{(1)}, \ldots, a_t^{(m)}, s_{t+1})$$

  Where stochasticity is due to exogenous users (i.e. it is random which impression opportunities are present on a given day).

Markov Decision Process
Let agent x be a learning agent and let all other agents have fixed policies. A fully-observable MDP $\langle S, A, R, T, \gamma \rangle$ w.r.t. agent x would be:

[17] Spend is needed in the state in order to avoid state aliasing

[18] On most days of the game, an agent receives non-positive reward because it spends but doesn't earn (recall payout only happens on the last day). On the last day, the agent participates in its last auction, after which the game ends and payout occurs. Thus only on the last day can an agent receive positive reward.

[19] Technically, $r_t^{(i)} \sim P(r|s_t, a_t^{(1)}, \ldots, a_t^{(m)})$, since the reward each day is based on the impression opportunities sold each day, which in turn are drawn randomly and independently of the state and agent actions. However, it can be shown that $r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}, s_{t+1}) = \mathbb{E}_{r_t^{(i)} \sim P(r|s_t, a_t^{(1)}, \ldots, a_t^{(m)})}\left[r_t^{(i)}\right]$. The proof is excluded from this paper.

- Let $S = S^{(x)} \times S^{(-x)}$, where (x) denotes agent x and $(-x)$ denotes all the other agents. Let state $s^{(i)} \in S^{(i)}$ for agent i be the same as in the MG.
- Let action $a \in A$ for agent x be the same as in the MG.
- Let $r(s_t, a_t) := r^{(x)}(s_t, \pi_{\theta_1}(s_t), \ldots, a_t, \ldots, \pi_{\theta_m}(s_t))$
  where $r^{(x)}$ is defined the same as in the MG.
- Let T be defined using the T in the MG:

$$T(s_t, a_t, s_{t+1}) := T(s_t, \pi_{\theta_1}(s_t), \ldots, a_t, \ldots, \pi_{\theta_m}(s_t), s_{t+1})$$

8.2 Derby Framework

[Source: https://github.com/nkumar15-brown-university/derby]

Derby is a simple bidding, auction, and *market* framework for creating and running auction or market games. Environments in derby can be interfaced in a similar fashion as environments in OpenAI's gym:

```
env = ...
agents = ...
env.init(agents, num_of_days)
for i in range(num_of_trajs):
        all_agents_states = env.reset()
        for j in range(horizon_cutoff):
            actions = []
            for agent in env.agents:
                agent_states = env.get_folded_states(
                                    agent, all_agents_states
                                )
                actions.append(agent.compute_action(agent_states))
            all_agents_states, rewards, done = env.step(actions)
```

A *market* can be thought of as a stateful, repeated auction:

- A market is initialized with m bidders, each of which has a state.
- A market lasts for N days.
- Each day, auction items are put on sale. Each day, the bidders participate in an auction for the available items.
- Each bidder's state is updated at the end of every day. The state can track information such as auction items bought and amount spent.

8.3 Algorithms

The algorithms in this section are derived from a multi-agent perspective. Since this paper only addresses the single-agent scenario, assume $m = 1$ for the algorithms below. The algorithms should fold naturally into their respective canonical single-agent PG algorithms.

8.3.1 Multi-Agent REINFORCE

Definitions

(See Formal RL Definition section)

Assumptions

(See Formal RL Definition section)

ADX RL Research Summary

Algorithm Derivation

$$
\begin{aligned}
\nabla_{\theta_i} J(\theta_i) &= \nabla_{\theta_i} \int p_\theta(\tau) r^{(i)}(\tau) d\tau \\
&= \int \nabla_{\theta_i} p_\theta(\tau) r^{(i)}(\tau) d\tau \\
&= \int p_\theta(\tau) \nabla_{\theta_i} \log p_\theta(\tau) r^{(i)}(\tau) d\tau \\
&= \int p_\theta(\tau) \nabla_{\theta_i} \log \left[p(s_0) \prod_{t=0}^{T-2} \left(\prod_{j=1}^{m} \pi_{\theta_j}(a_t^{(j)}|s_t) \right) p(s_{t+1}|s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right] r^{(i)}(\tau) d\tau \\
&= \ldots \left[\nabla_{\theta_i} \log p(s_0) + \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\prod_{j=1}^{m} \pi_{\theta_j}(a_t^{(j)}|s_t) \right) + \nabla_{\theta_i} \log p(s_{t+1}|s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right] \ldots \\
&\qquad \text{(let } \ldots \text{ represent other parts of equation)} \\
&= \ldots \left[0 + \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\prod_{j=1}^{m} \pi_{\theta_j}(a_t^{(j)}|s_t) \right) + 0 \right] \ldots \\
&= \ldots \left[\sum_{t=0}^{T-2} \sum_{j=1}^{m} \nabla_{\theta_i} \log \left(\pi_{\theta_j}(a_t^{(j)}|s_t) \right) \right] \ldots \\
&= \ldots \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)}|s_t) \right) \right] \ldots \\
&= \int p_\theta(\tau) \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)}|s_t) \right) \right] r^{(i)}(\tau) d\tau \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)}|s_t) \right) \right) \left(\sum_{t=0}^{T-2} r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right) \right] \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)}|s_t) \right) \left(\sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \ldots, a_{t'}^{(m)}) \right) \right] \\
&\qquad \text{(causality trick)} \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)}|s_t) \right) \hat{Q}_t^{(i)} \right] \\
&\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)}|s_{j,t}) \right) \hat{Q}_{j,t}^{(i)}
\end{aligned}
$$

using N trajectories sampled from $p_\theta(\tau)$

Algorithm

1. Sample N trajectories from $p_\theta(\tau)$
2. Use trajectories to calculate $\nabla_{\theta_i} J(\theta_i)$
3. $\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} J(\theta_i)$

8.3.2 Multi-Agent REINFORCE with baseline

Definitions
(Same as REINFORCE)
Assumptions
(Same as REINFORCE)
Algorithm Derivation

$$\nabla_{\theta_i} J(\theta_i) = \nabla_{\theta_i} \int p_\theta(\tau) r^{(i)}(\tau) d\tau$$

$$\vdots \quad \text{(see multi-agent REINFORCE derivation for more detail)}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \hat{Q}_t^{(i)} \right]$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) (\hat{Q}_t^{(i)} - V^{\pi_{\theta_i}}(s_t)) \right]$$

(use baseline to lower variance)

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) (\hat{Q}_{j,t}^{(i)} - V^{\pi_{\theta_i}}(s_{j,t}))$$

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) (\hat{Q}_{j,t}^{(i)} - \hat{V}_{\mathbf{w}_i}(s_{j,t})) \qquad \text{(use VFA)}$$

using N trajectories sampled from $p_\theta(\tau)$
where $\hat{V}_{\mathbf{w}_i}$ is a VFA of parameters $\mathbf{w}_i$ to fit $V^{\pi_{\theta_i}}$

Algorithm

1. Sample N trajectories from $p_\theta(\tau)$
2. Use trajectories to calculate $\nabla_{\theta_i} J(\theta_i)$.
   Use trajectories to calculate $\text{loss}(\hat{V}_{\mathbf{w}_i}) = \sum_{j,t} \left(y_{j,t} - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right)^2$,
   where $y_{j,t} = \sum_{t'=t}^{T-2} r^{(i)}(s_{j,t'}, a_{j,t'}^{(1)}, \ldots, a_{j,t'}^{(m)})$.
3. $\theta \leftarrow \theta + \alpha_1 \nabla_{\theta_i} J(\theta_i)$.
   $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_2 \nabla_{\mathbf{w}_i} \text{loss}(\hat{V}_{\mathbf{w}_i})$.

8.3.3 Multi-Agent Actor-Critic Q with baseline

Definitions
(Same as REINFORCE)

Assumptions
(Same as REINFORCE)

Algorithm Derivation

$$\nabla_{\theta_i} J(\theta_i) = \nabla_{\theta_i} \int p_\theta(\tau) r^{(i)}(\tau) d\tau$$

$\vdots$ (see multi-agent REINFORCE derivation for more detail)

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \hat{Q}_t^{(i)} \right]$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \left(Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) - V^{\pi_{\theta_i}}(s_t) \right) \right]$$

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) \left(Q^{\pi_{\theta_i}}(s_{j,t}, a_{j,t}^{(1)}, \dots, a_{j,t}^{(m)}) - V^{\pi_{\theta_i}}(s_{j,t}) \right)$$

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) \left(\hat{Q}_{\mathbf{u}_i}(s_{j,t}, a_{j,t}^{(1)}, \dots, a_{j,t}^{(m)}) - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right)$$

($\hat{Q}_{\mathbf{u}_i}$ introduces bias, lowers variance)

using N trajectories sampled from $p_\theta(\tau)$

where $\hat{Q}_{\mathbf{u}_i}$ is a VFA of parameters $\mathbf{u}_i$ to fit $Q^{\pi_{\theta_i}}$

where $\hat{V}_{\mathbf{w}_i}$ is a VFA of parameters $\mathbf{w}_i$ to fit $V^{\pi_{\theta_i}}$

Algorithm

1. Sample N trajectories from $p_\theta(\tau)$
2. Use trajectories to calculate $\nabla_{\theta_i} J(\theta_i)$.

   Use trajectories to calculate $\text{loss}(\hat{V}_{\mathbf{w}_i}) = \sum_{j,t} \left(y_{j,t} - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right)^2$,
   where $y_{j,t} = \sum_{t'=t}^{T-2} r^{(i)}(s_{j,t'}, a_{j,t'}^{(1)}, \dots, a_{j,t'}^{(m)})$.

   Use trajectories to calculate $\text{loss}(\hat{Q}_{\mathbf{u}_i}) = \sum_{j,t} \left(y_{j,t} - \hat{Q}_{\mathbf{u}_i}(s_{j,t}, a_{j,t}^{(1)}, \dots, a_{j,t}^{(m)}) \right)^2$,
   where $y_{j,t} = \sum_{t'=t}^{T-2} r^{(i)}(s_{j,t'}, a_{j,t'}^{(1)}, \dots, a_{j,t'}^{(m)})$.
3. $\theta \leftarrow \theta + \alpha_1 \nabla_{\theta_i} J(\theta_i)$.
   $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_2 \nabla_{\mathbf{w}_i} \text{loss}(\hat{V}_{\mathbf{w}_i})$.
   $\mathbf{u}_i \leftarrow \mathbf{u}_i + \alpha_3 \nabla_{\mathbf{u}_i} \text{loss}(\hat{Q}_{\mathbf{u}_i})$.

8.3.4 Multi-Agent Actor-Critic TD (has baseline)

Definitions
(Same as REINFORCE)

Assumptions
(Same as REINFORCE)

Multi-Agent Bellman Rollout Derivation

$$
\begin{aligned}
V^{\pi_{\theta_i}}(s_t) :&= \mathbb{E}_{\tau\sim p_\theta(\tau_t)}\left[\sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \dots, a_{t'}^{(m)}) \middle| s_t\right] \\
&= \mathbb{E}_{a_{t+1}^{(1)},\dots,a_{t+1}^{(m)}\sim\pi_\theta(a_{t+1}^{(1)},\dots,a_{t+1}^{(m)}|s_{t+1})}\left[\mathbb{E}_{\tau\sim p_\theta(\tau_t)}\left[\sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \dots, a_{t'}^{(m)}) \middle| s_t, a_t^{(1)}, \dots, a_t^{(m)}\right] \middle| s_t\right] \\
&= \mathbb{E}_{a_{t+1}^{(1)},\dots,a_{t+1}^{(m)}\sim\pi_\theta(a_{t+1}^{(1)},\dots,a_{t+1}^{(m)}|s_{t+1})}\left[Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) \middle| s_t\right]
\end{aligned}
$$

And:

$$
\begin{aligned}
Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) :&= \mathbb{E}_{\tau\sim p_\theta(\tau_t)}\left[\sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \dots, a_{t'}^{(m)}) \middle| s_t, a_t^{(1)}, \dots, a_t^{(m)}\right] \\
&= \mathbb{E}_{\tau\sim p_\theta(\tau_t)}\left[r^{(i)}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) + \sum_{t'=t+1}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \dots, a_{t'}^{(m)}) \middle| s_t, a_t^{(1)}, \dots, a_t^{(m)}\right] \\
&= r^{(i)}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) + \mathbb{E}_{\tau\sim p_\theta(\tau_t)}\left[\sum_{t'=t+1}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \dots, a_{t'}^{(m)}) \middle| s_t, a_t^{(1)}, \dots, a_t^{(m)}\right] \\
&= r^{(i)}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) + \mathbb{E}_{\tau\sim p_\theta(\tau_{t+1})}\left[\sum_{t'=t+1}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \dots, a_{t'}^{(m)})\right] \\
&= \cdots \mathbb{E}_{\tau\sim p_\theta(\tau_{t+1})}\left[\sum_{t'=t+1}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \dots, a_{t'}^{(m)})\right] \cdots \\
&\qquad\qquad \text{(Let } \dots \text{ represent other parts of equation)}
\end{aligned}
$$

$$= \cdots \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t, a_t^{(1)}, \ldots, a_t^{(m)})} \Bigg[\mathbb{E}_{a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)} \sim \pi_\theta(a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)}|s_{t+1})} \left[\mathbb{E}_{\tau \sim p_\theta(\tau_{t+1})} \left[\sum_{t'=t+1}^{T-2} r^{(i)}\big(s_{t'}, a_{t'}^{(1)}, \ldots, a_{t'}^{(m)}\big) \middle| s_{t+1}, a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)} \right] \middle| s_{t+1} \right] \Bigg] \cdots$$

$$= \cdots \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t, a_t^{(1)}, \ldots, a_t^{(m)})} \left[\mathbb{E}_{a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)} \sim \pi_\theta(a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)}|s_{t+1})} \left[Q^{\pi_{\theta_i}}\big(s_{t+1}, a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)}\big) \middle| s_{t+1} \right] \right] \cdots$$

$$= \cdots \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t, a_t^{(1)}, \ldots, a_t^{(m)})} \left[V^{\pi_{\theta_i}}\big(s_{t+1}\big) \right] \cdots$$

$$= r^{(i)}\big(s_t, a_t^{(1)}, \ldots, a_t^{(m)}\big) + \mathbb{E}_{s_{t+1} \sim p(s_{t+1}|s_t, a_t^{(1)}, \ldots, a_t^{(m)})} \left[V^{\pi_{\theta_i}}\big(s_{t+1}\big) \right]$$

Algorithm Derivation

$$\nabla_{\theta_i} J(\theta_i) = \nabla_{\theta_i} \int p_\theta(\tau) r^{(i)}(\tau) d\tau$$

$\vdots$ (see multi-agent REINFORCE derivation for more detail)

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \hat{Q}_t^{(i)} \right]$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \left(Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) - V^{\pi_{\theta_i}}(s_t) \right) \right]$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \left(r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) + \mathbb{E}\left[V^{\pi_{\theta_i}}(s_{t+1}) \right] - V^{\pi_{\theta_i}}(s_t) \right) \right]$$

(bootstrapping)

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) \left[r^{(i)}(s_{j,t}, a_{j,t}^{(1)}, \ldots, a_{j,t}^{(m)}) + V^{\pi_{\theta_i}}(s_{j,t+1}) - V^{\pi_{\theta_i}}(s_{j,t}) \right]$$

(single sample bootstrapping)

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) \left[r^{(i)}(s_{j,t}, a_{j,t}^{(1)}, \ldots, a_{j,t}^{(m)}) + \hat{V}_{\mathbf{w}_i}(s_{j,t+1}) - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right]$$

(use VFA; VFA in one-step rollout introduces bias, lowers variance)

using N trajectories sampled from $p_\theta(\tau)$

where $\hat{V}_{\mathbf{w}_i}$ is a VFA of parameters $\mathbf{w}_i$ to fit $V^{\pi_{\theta_i}}$

Algorithm

1. Sample N trajectories from $p_\theta(\tau)$
2. Use trajectories to calculate $\nabla_{\theta_i} J(\theta_i)$.

   Use trajectories to calculate $\text{loss}(\hat{V}_{\mathbf{w}_i}) = \sum_{j,t} \left(y_{j,t} - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right)^2$,
   where $y_{j,t} = r^{(i)}(s_{j,t}, a_{j,t}^{(1)}, \ldots, a_{j,t}^{(m)}) + \hat{V}_{\mathbf{w}_i}(s_{j,t+1})$.
3. $\theta \leftarrow \theta + \alpha_1 \nabla_{\theta_i} J(\theta_i)$.
   $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_2 \nabla_{\mathbf{w}_i} \text{loss}(\hat{V}_{\mathbf{w}_i})$.

8.3.5 Multi-Agent Actor-Critic SARSA with baseline

Definitions
(Same as REINFORCE)

ADX RL Research Summary

Assumptions
(Same as REINFORCE)

Multi-Agent Bellman Rollout Derivation
(SAME as AC TD)

Algorithm Derivation

$$\nabla_{\theta_i} J(\theta_i) = \nabla_{\theta_i} \int p_\theta(\tau) r^{(i)}(\tau) d\tau$$

$$\vdots \quad \text{(see multi-agent REINFORCE derivation for more detail)}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \hat{Q}_t^{(i)} \right]$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \left(Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) - V^{\pi_{\theta_i}}(s_t) \right) \right]$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\dots \left(r^{(i)}(s_t, a_t^{(1)}, \dots, a_t^{(m)}) + \mathbb{E} \left[\mathbb{E} \left[Q^{\pi_{\theta_i}}(s_{t+1}, a_{t+1}^{(1)}, \dots, a_{t+1}^{(m)}) \right] \right] - V^{\pi_{\theta_i}}(s_t) \right) \right]$$

(bootstrapping; using $\dots$ to represent preceding parts of equation)

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) \left[r^{(i)}(s_{j,t}, a_{j,t}^{(1)}, \dots, a_{j,t}^{(m)}) + Q^{\pi_{\theta_i}}(s_{j,t+1}, a_{j,t+1}^{(1)}, \dots, a_{j,t+1}^{(m)}) - V^{\pi_{\theta_i}}(s_{j,t}) \right]$$

(single sample bootstrapping)

$$\approx \frac{1}{N} \sum_{j=0}^{N-1} \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_{j,t}^{(i)} | s_{j,t}) \right) \left[r^{(i)}(s_{j,t}, a_{j,t}^{(1)}, \dots, a_{j,t}^{(m)}) + \hat{Q}_{\mathbf{u}_i}(s_{j,t+1}, a_{j,t+1}^{(1)}, \dots, a_{j,t+1}^{(m)}) - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right]$$

(use VFA; $\hat{Q}_{\mathbf{u}_i}$ introduces bias, lowers variance)

using N trajectories sampled from $p_\theta(\tau)$

where $\hat{Q}_{\mathbf{u}_i}$ is a VFA of parameters $\mathbf{u}_i$ to fit $Q^{\pi_{\theta_i}}$

where $\hat{V}_{\mathbf{w}_i}$ is a VFA of parameters $\mathbf{w}_i$ to fit $V^{\pi_{\theta_i}}$

Algorithm

1. Sample N trajectories from $p_\theta(\tau)$
2. Use trajectories to calculate $\nabla_{\theta_i} J(\theta_i)$.

   Use trajectories to calculate $\text{loss}(\hat{V}_{\mathbf{w}_i}) = \sum_{j,t} \left(y_{j,t} - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right)^2$,
   where $y_{j,t} = \sum_{t'=t}^{T-2} r^{(i)}(s_{j,t'}, a_{j,t'}^{(1)}, \dots, a_{j,t'}^{(m)})$.

   Use trajectories to calculate $\text{loss}(\hat{Q}_{\mathbf{u}_i}) = \sum_{j,t} \left(y_{j,t} - \hat{Q}_{\mathbf{u}_i}(s_{j,t}, a_{j,t}^{(1)}, \dots, a_{j,t}^{(m)}) \right)^2$,
   where $y_{j,t} = r^{(i)}(s_{j,t}, a_{j,t}^{(1)}, \dots, a_{j,t}^{(m)}) + \hat{Q}_{\mathbf{u}_i}(s_{j,t+1}, a_{j,t+1}^{(1)}, \dots, a_{j,t+1}^{(m)})$.

3. $\theta \leftarrow \theta + \alpha_1 \nabla_{\theta_i} J(\theta_i)$.
   $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_2 \nabla_{\mathbf{w}_i} \text{loss}(\hat{V}_{\mathbf{w}_i})$.
   $\mathbf{u}_i \leftarrow \mathbf{u}_i + \alpha_3 \nabla_{\mathbf{u}_i} \text{loss}(\hat{Q}_{\mathbf{u}_i})$.

8.4 Theoretical Analysis and Additional Challenges

For simplicity, consider a 1 day game of experimental setup 1. The reward curve for such a game would look like `Figure 3`.

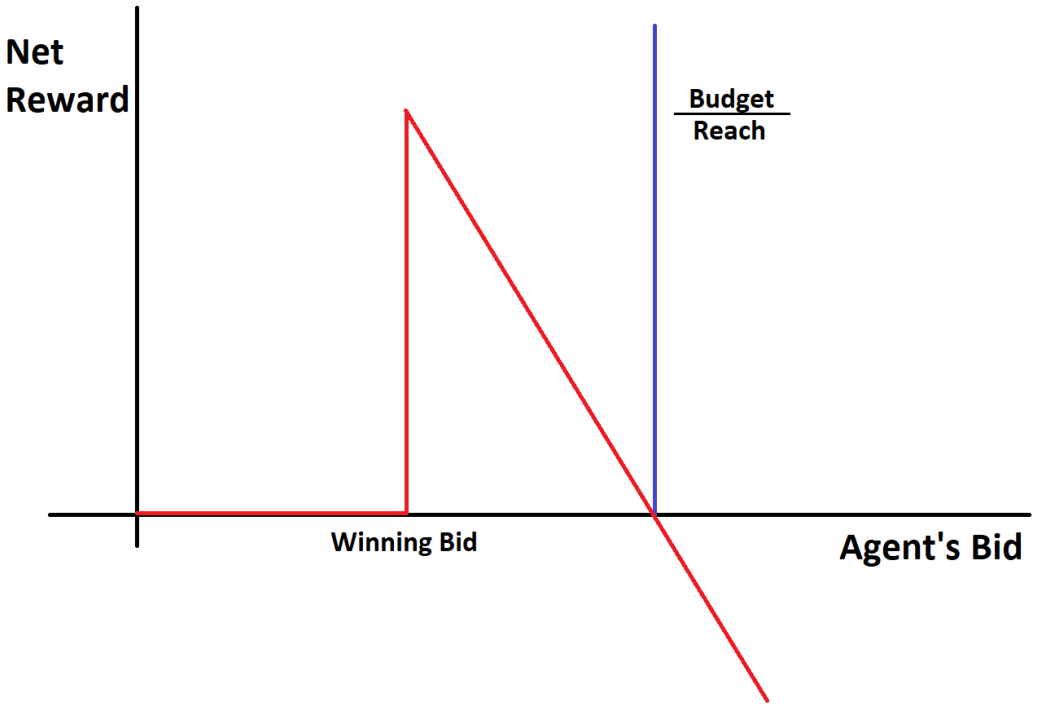

Figure 3: Example Reward Curve (for a 1 day game)

[NOTE: in `Figure 3`, reward is called "net reward" because its concept of "daily reward curve" can be extended to multi-day games, though that won't be shown in this paper.]

There are some notable aspects about such a reward curve:

- If the agent spends less than the winning bid, then it gets no reward [20]. So the reward curve is both **flat** and **zero** in the region $[0, \text{winning bid})$.
- Once the agent passes the winning bid threshold, it receives its biggest reward [21]. Any

[20] Although the agent placed a bid, its spend is 0 because it won nothing.

[21] The agent receives appropriate payout for the impression won, while spending the least it could have to win said impression.

higher bid only results in decreasing its reward [22]. The reward curve is **not flat** in the region [winning bid, $\frac{\text{budget}}{\text{reach}}$).

- Continuing the previous logic, the agent's reward keeps decreasing until its bid crosses the $\frac{\text{budget}}{\text{reach}}$ threshold, after which point it receives negative net reward. The reward curve is **not flat** in the region $[\frac{\text{budget}}{\text{reach}}, \infty)$.

From the perspective of a policy gradient algorithm, there are two regions of the reward curve: 1) the **flat** and **zero** region $[0, \text{winning bid})$ and 2) the **not flat** region $[\text{winning bid}, \infty)$. The two regions are "connected" via a discontinuity.

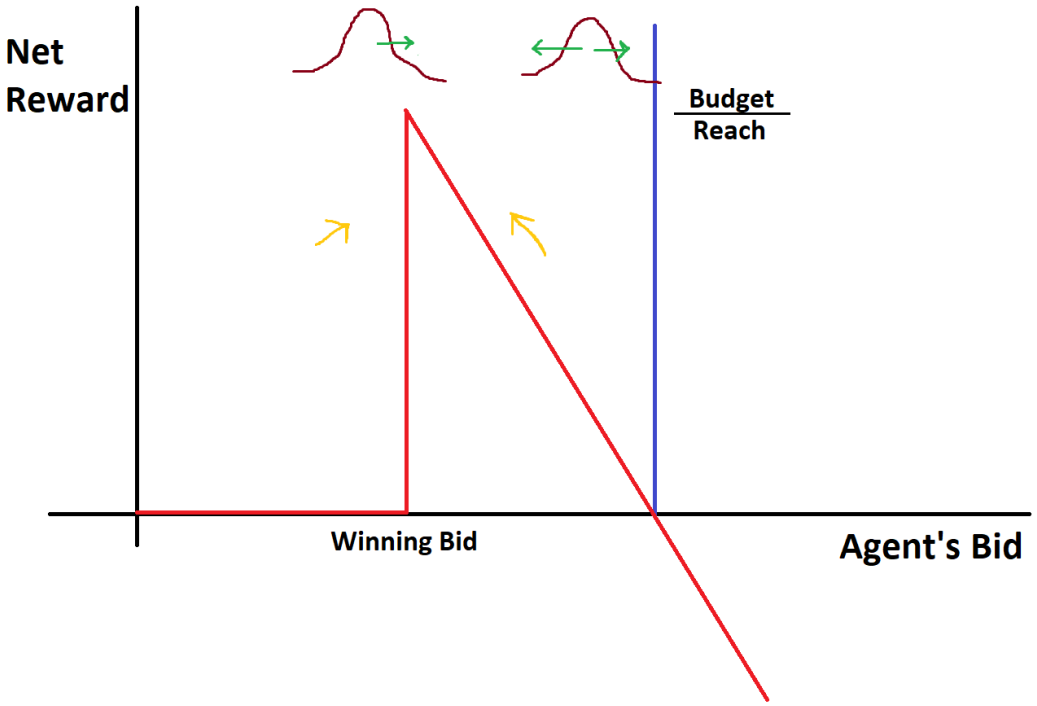

Figure 4: A (Gaussian) policy learning via the Reward Curve

`Figure 4` portrays how a policy centered in either region would move toward the optimal policy. In the non-flat region, learning is straightforward: climb up the hill. In the flat and zero region, however, learning is more complicated; Because the region has zero reward, many sample trajectories will contribute zero gradient, thus making them essentially useless. Furthermore, even if the region had non-zero reward, the flat nature would lead to many sample trajectories "canceling" each other out. These issues imply that although a policy in the zero reward region can technically recover toward the optimal policy, it becomes increasingly harder (due to sample inefficiency) the deeper (i.e. more left) the policy gets.

[22] The impression is already won, so the payout does not change. However, a larger bid means greater spend by the agent.

It seems then that the challenge of learning under such a reward curve is to start in the non-flat region and climb up the hill toward the optimal bid. However, there is further subtlety:

- The "effective step size" (i.e. $\Delta\theta_i$) each PG iteration is determined both by the learning rate and by $\hat{Q}_t^{(i)}$.
- Increasing the learning rate risks the policy "jumping off the cliff" once it nears the edge.
- The effective step increases higher up the hill because $\hat{Q}_t^{(i)}$ increases. Thus another reason the policy risks "jumping off the cliff" once it nears the edge.

Is "jumping off the cliff" into the flat and zero region really so bad? Empirically, yes, as it is easy for a policy in this region to be positioned such that it takes a massive number of sample trajectories to ever recover. In fact, the problem is so bad that we coined it **zero collapse**. When an algorithm hits zero collapse, it is almost always better to just restart the learning process. All of this implies that PG algorithms in this problem are sensitive to the learning rate. It further implies that it is worth trying techniques that decrease the effective step size as the policy climbs up the cliff.

Although we don't know what the reward curve would look like in higher dimensions (and longer horizon games), we believe the core aspects of the simple 1-day analysis will hold. Namely: the existence of flat-and-zero regions, the existence of non-flat regions, a discontinuity "connecting" the two types of regions, and consequently the sensitivity to the effective step size.

8.5 Network Architectures

8.5.1 Policy Network

For all algorithms, the policy network learns a Gaussian distribution using the architecture portrayed in `Figure 5`.

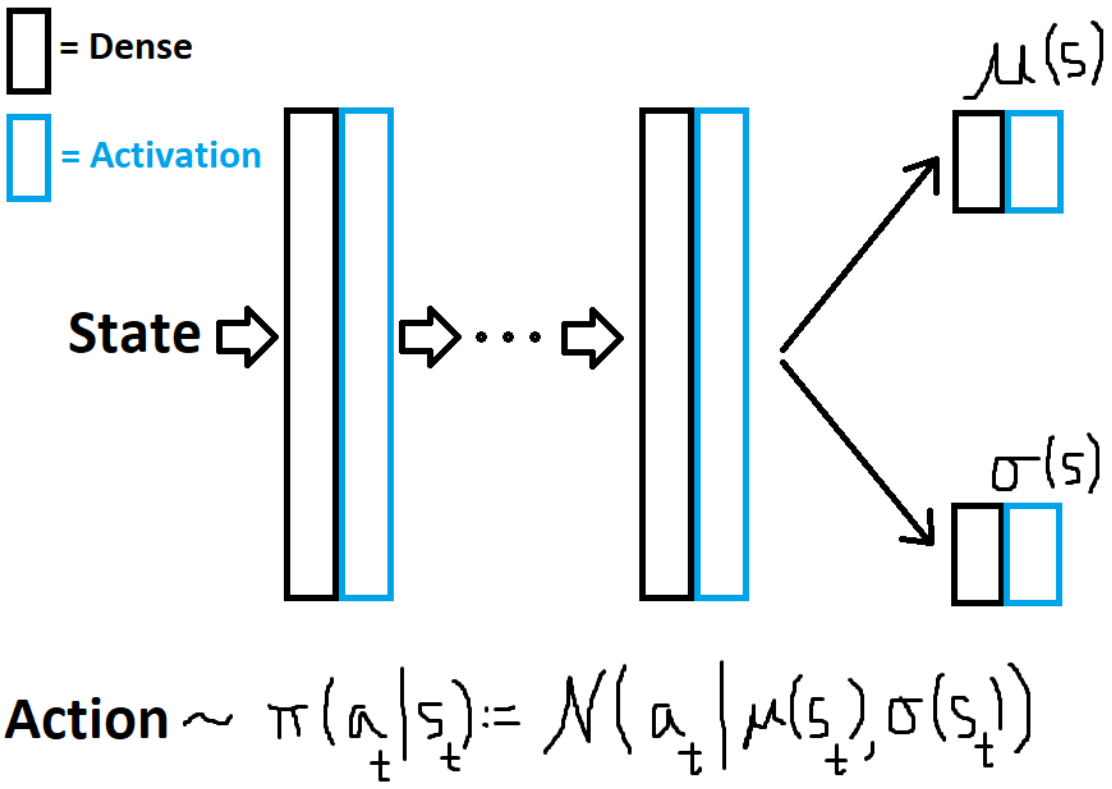

Figure 5: Policy Network Architecture.

For the simple experimental setups in this paper, all the algorithms have only 1 dense layer with size 1 before the μ, σ layers.

See the `Algorithm Tuning` section for details about the activation functions.

8.5.2 Value Networks

For algorithms that learn Q or V, the networks were pretty standard: one or more dense layers taking state (and action for Q) as input and returning a Q or V value as output. The activation functions in the value networks were leaky ReLU. The untuned/semi-tuned versions have 2 dense layers for Q, with a layer sizes ($6 \times$ num_of_subactions) and 1 respectively. They also have 2 dense layers for V, with layer sizes 6 and 1 respectively. Tuned v2 versions differ in that they have 4 dense layers for Q, with the first three layers having size $6 + (2 \times$ num_of_subactions) and the last layer having size 1.

8.6 Algorithm Tuning

See sections `Theoretical Analysis and Additional Challenges` on the motivation for the tuning presented.

The first optimization we made was to initialize the policy distribution to be roughly centered around the $\frac{\text{budget}}{\text{reach}}$ point. The second optimization we made was to setup the policy network to learn an offset from its initial position [23].

The next optimization we made was to change the activation functions of the policy network. We shied away from activation functions with large linear regions (e.g. ReLU, sigmoid, tanh) and instead chose those whose slope becomes more shallow as its input decreases (e.g. softplus, ELU) [24] [25]. Activation functions of the latter type are beneficial because they mimic decreasing the effective step size. That said, we found that leaky ReLU on all activation functions preceding the μ, σ layers still worked well when combined with softplus on the μ, σ layers.

In total we have three versions of each of the PG algorithms:

- Untuned/semi-tuned

  The only tuning here is leaky ReLU on all policy network activation functions except the μ, σ layers, which have softplus.

- Tuned v2

  All the optimizations described in this section. Leaky ReLU on all policy network activation functions except the μ, σ layers, which have softplus.

- Tuned v3 (not discussed in this paper)

  All the optimizations described in this section. ELU on all policy network activation functions except the μ, σ layers, which have softplus.

Finally, for all versions of all algorithms we do state and action scaling. This is important so as to keep the network weights from blowing up.

[23] Note that this usually amounts to the policy learning to decrease its weights (which moves it leftward of the $\frac{\text{budget}}{\text{reach}}$ point).

[24] Technically, we did experiment with activation functions like ReLU, sigmoid, and tanh, which both confirmed our reservations and posed other issues.

[25] Note that the only rational activation functions for the μ, σ layers are those that always return non-negative output (e.g. ReLU, softplus), since bids must always be non-negative.