Nishant Kumar
B01535964

AdX RL Research

1 Project Overview and Motivation

The focus of our project was applying reinforcement learning to the AdX game. The project was a collaboration with Prof. Amy Greenwald and PhD student Enrique Areyan. The AdX game crudely models the digital advertising domain: advertisers buy *impression opportunities* 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 advertisers and the goods being sold are impression opportunities. In the AdX game, each player plays the role of an *advertiser liaison* [1]; advertisers procure *ad campaigns* [2] to liaisons, who are responsible for fulfilling the campaign within a certain time frame [3]. The goal of each player is to learn what bids to place in order to maximize their payout by the end of the game.

The AdX game is interesting to tackle from a reinforcement learning 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 [4].
- 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 examined from both a single-agent perspective and a multi-agent perspective.

[1] 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*.

[2] 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.)

[3] 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.

[4] 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 Background

2.1 Reinforcement Learning

- An MDP is defined by $\langle S, A, R, T, \gamma \rangle$, where:
  - S is a set of states.
  - A is a set of actions.
  - R is a reward function and $R(s, a)$ (or alternatively $R(s, a, s')$) is the immediate reward received from being in state s and taking action a.
  - T is a transition function and $T(s, a, s')$ is the probability of transitioning to state s' after taking action a from state s.
  - γ is a discount factor between 0 and 1 (inclusive).
- An *agent* behaves according to a *policy* π, where π can be either:
  - a deterministic function dictating which action to take from which state, i.e. $\pi : S \to A$.
  - a stochastic function dictating with what probability to take an action from a state, i.e. $\pi : S \times A \to Pr[0, 1]$.
- A *trajectory* or *episode* τ is a sequence of (s_t, a_t, r_t, s_{t+1}) an agent experiences in one "run":

  $$\tau = s_0 \to a_0 \to r_0 \to s_1 \to \cdots \to s_{T-1}$$

  Where T denotes the *horizon* or length of the "run". Note that T could be ∞.

  Where $p(\tau)$ represents the probability distribution over trajectories τ. Note that if rewards are deterministic, then the r_i's are not needed in $p(\tau)$. Note that $p(\tau)$ depends on π.

  Where r_t is the immediate reward the world gives to the agent at time t. Note that the world determines r_t according to $r_t = R(s_t, a_t)$.
- Canonically, the goal of an agent is to maximize its *expected sum of discounted rewards*:

  $$V^{\pi}(s_t) = \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} | s_t \right]$$

  $$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s_0)$$

2.2 AdX Game

- An agent to play a greatly simplified version of the most recent game, Ad Exchange (AdX), in which agents play the role of ad networks, competing to procure contracts (i.e., advertising campaigns) from advertisers (e.g., retailers), and then bidding in an exchange on opportunities to exhibit those ads to Internet users as they browse the web.
- The primary simplifications are: your agent will not compete to procure campaigns (instead your agent will be assigned one, randomly, from a known distribution), user demographics are fully visible.
- A *campaign* is defined as:
  - 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).
  - A *reach*: the number of ads to be shown to users in the relevant market segment.
  - A *budget*: the amount of money the advertiser will pay if this contract is fulfilled.
- Here is an example of a campaign:

  [Segment = Female_Old, Reach = 500, Budget = $40.0]

  To fulfill this campaign, your agent must show at least 500 advertisements to older women. If successful, it will earn $40. Showing an advertisement to a user is equivalent to winning that user's auction. But note that winning an auction for a user who does not match a campaign's market segment does not count toward fulfilling that campaign.
- The agent with the highest profit wins that simulation. Because of the randomness in each simulation, the game is simulated repeatedly and scores are averaged over multiple simulations to determine an overall winner.

2.2.1 1 campaign over N days game variant

- Each agent will be given an initial campaign to fulfill over the N days of the game.
- Each agent will be paid out at the end of the last day of the game. Pay out will be based on campaign budget.

2.2.2 N campaigns over N days game variant

- agents will be given an initial campaign that lasts for just one day (just like in the One-Day variant), and then, contingent upon the agent's performance in fulfilling its initial campaign, which will be measured by its *quality score*, it will be given a second campaign, whose budget will be discounted by this quality score. This repeats for N days.

- The *quality score* is a number between 0 and 1.38442, where 0 denotes very low quality (in case the agent acquired 0 impression for the first campaign), 1 denotes very good quality (in case the agent acquired the number of impressions needed), and 1.38442 denotes perfect quality (in case the agent acquired many many impressions, above and beyond the required number of impressions). Your agent is thus faced with the usual task of fulfilling its first campaign profitably, but at the same time in such a way as to earn a valuable second campaign!

3 Applying RL to the AdX game

3.1 System Overview

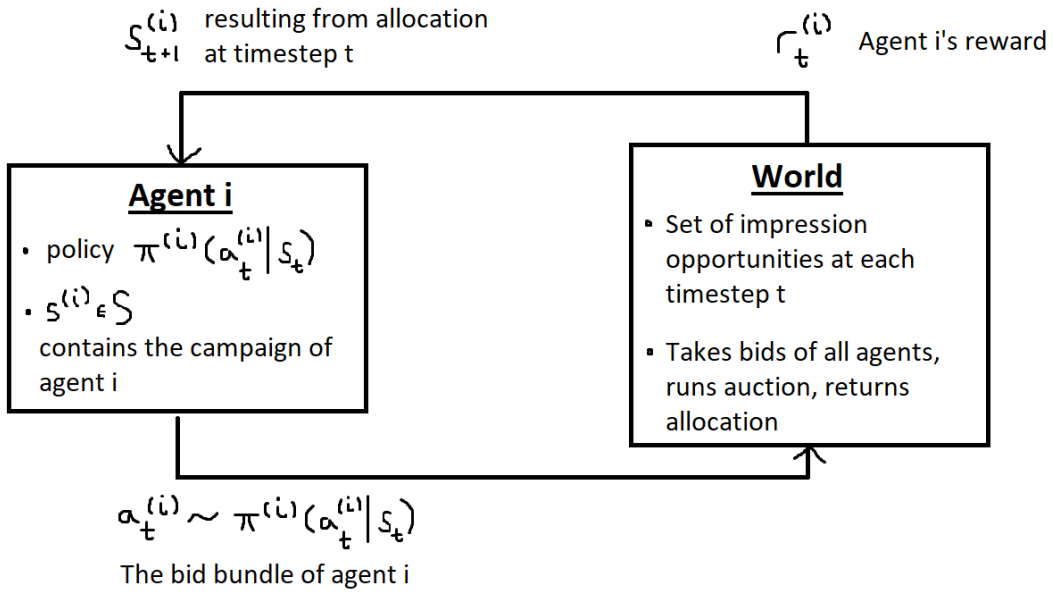

Figure 1: Agent-World System

3.2 1 campaign over N days game

[Last Updated: September 2020]

3.2.1 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:

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

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

- 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).
    - **spend**: the amount spent so far in fulfilling this campaign. This is needed to avoid state aliasing.
    - **impressions**: the number of impressions achieved so far. Note that impressions $\leq$ reach.
    - **day**: the day of the game (i.e. timestep 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, u_j) is the **bid** and **budget** for market segment j.
  - Let $r^{(i)}(s, a^{(1)}, \ldots, a^{(m)})$ for each agent i be:

$$r^{(i)}(s, a^{(1)}, \ldots, a^{(m)}) := \text{how much earned on day } t \text{ (e.g. budget on day } N) - \text{how much spent on day } t$$

    A couple of subtleties here:

  - 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.
  - 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 $r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}, s_{t+1}) = \mathbb{E}_{r_t^{(i)} \sim P(r|s_t, a_t^{(1)}, \ldots, a_t^{(m)})}\left[r_t^{(i)}\right]$.
- 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).

- 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:
  - 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:

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

- Let agent x be a learning agent and let all other agents have fixed policies. A partially-observable MDP $\langle S, A, R, T, \gamma \rangle$ w.r.t. agent x would be:
  - Let $S = S^{(x)}$, where (x) denotes agent x and $S^{(x)}$ is the same as in the MG. Let state $s \in S^{(x)}$ for agent x be the same as in the MG, but without the **spend** field.
  - Let A be the same as in the MDP.
  - Let $r(s_t, a_t) := r^{(x)}(s_t, \pi_{\theta_1}(s_t^{(1)}), \ldots, a_t, \ldots, \pi_{\theta_m}(s_t^{(m)}))$
    where $r^{(x)}$ is defined the same as in the MG.
    where $s^{(i)}$ is defined the same as in the MG.
  - Let T be defined using the T in the MG:

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

3.2.2 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.
- Assume full observability means "every agent sees other agents' states". Without this assumption, state aliasing can occur. Note that if using REINFORCE, then don't need this assumption; REINFORCE doesn't rely on Markov assumption, so it will handle state aliasing by essentially averaging the action to take (since it learns a non-deterministic policy).

4 Algorithms

4.1 Multi-Agent REINFORCE

[Last Updated: September 2020]

4.1.1 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 $i \in [1, \dots, m]$.
- 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 $r^{(i)}(s_t, a_t^{(1)}, \dots, a_t^{(m)})$ 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:

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

  and more generally:

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

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

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

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

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

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

4.1.2 Assumptions

- 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:

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

4.1.3 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 \\
&= \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\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 $p_\theta(\tau)$

4.1.4 Algorithm

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

4.1.5 Common Tricks

- **Causality trick**

$$
\begin{aligned}
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \right) \left(\sum_{t=0}^{T-2} r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right) \right] \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \left(\sum_{t=0}^{T-2} r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right) \right) \right] \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\cdots + \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_k^{(i)} | s_k) \right) \left(\sum_{t=0}^{T-2} r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right) + \ldots \right] \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\cdots + \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_k^{(i)} | s_k) \right) \left(\sum_{t=k}^{T-2} r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \right) + \ldots \right] \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)} | s_t) \right) \left(\sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \ldots, a_{t'}^{(m)}) \right) \right]
\end{aligned}
$$

  The causality trick means to remove terms because they don't matter due to causality. In the above, rewards at time $t < t'$ do not affect the policy at time t. Therefore, they can be removed from the corresponding sums. This decreases the variance of the expectation.

- **Using a baseline**

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

Subtracting a baseline b reduces the variance of the expectation. Note that the baseline can be anything, so long as it is not a function of $a_t^{(i)}$. One of the most common baselines is $V^{\pi_{\theta_i}}(s_t)$.

- **Value Function Approximation (VFA)**

  One trick is to substitute $\hat{Q}_t^{(i)}$ with $Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \ldots, a_t^{(m)})$ (or a Bellman rollout, see the bootstrapping section below):

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

  However, in non-tabular settings, $V^{\pi_{\theta_i}}$ and $Q^{\pi_{\theta_i}}$ cannot be calculated exactly. Therefore, they must be approximated via functions, i.e. $\hat{V}_{\mathbf{w}_i}$ and $\hat{Q}_{\mathbf{w}_i}$, where $\mathbf{w}_i$ are learned weights:

$$\hat{Q}_{\mathbf{w}_i}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) \approx Q^{\pi_{\theta_i}}(s_t, a_t^{(1)}, \ldots, a_t^{(m)})$$

  This is done by minimizing supervised loss:

$$\text{dataset} = \{\ldots, (s_{j,t}, y_{j,t}), \ldots | \text{sample } j \text{ and time } t\}$$

$$\text{loss} = \sum_{j,t} \left(y_{j,t} - \hat{V}_{\mathbf{w}_i}(s_{j,t}) \right)^2$$

  Where:

$$y_{j,t} = \sum_{t'=t}^{T-2} r^{(i)}(s_{j,t'}, a_{j,t'}^{(1)}, \ldots, a_{j,t'}^{(m)})$$

  Or (with bootstrapping):

$$y_{j,t} = r^{(i)}(s_{j,t}, a_{j,t}^{(1)}, \ldots, a_{j,t}^{(m)}) + \hat{V}_{\mathbf{w}_i}(s_{j,t+1})$$

  Note that using a VFA implies:

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

Though:

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

as

$$\hat{Q}_{\mathbf{w}_i}\rightarrow Q^{\pi_{\theta_i}}$$

This means $\hat{Q}_{\mathbf{w}_i}$ (and $\hat{V}_{\mathbf{w}_i}$) is initially *wrong* (i.e. bias), but it corrects over time.

- **Bootstrapping**

  Bootstrapping is about substituting $Q^{\pi_{\theta_i}}(s_t,a_t^{(1)},\ldots,a_t^{(m)})$ with its Bellman backup (only in Markov domains). For example, a one-step backup:

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

  Or in the case of a single-sample:

$$Q^{\pi_{\theta_i}}(s_{j,t},a_{j,t}^{(1)},\ldots,a_{j,t}^{(m)}) = r^{(i)}(s_{j,t},a_{j,t}^{(1)},\ldots,a_{j,t}^{(m)}) + V^{\pi_{\theta_i}}(s_{j,t+1})$$

  Note that VFAs can be used with bootstrapping:

$$Q^{\pi_{\theta_i}}(s_{j,t},a_{j,t}^{(1)},\ldots,a_{j,t}^{(m)}) \approx r^{(i)}(s_{j,t},a_{j,t}^{(1)},\ldots,a_{j,t}^{(m)}) + \hat{V}_{\mathbf{w}_i}(s_{j,t+1})$$

- **Bias - Variance Tradeoff**

  Intuitively speaking, when using a VFA with bootstrapping: a $k+1$-step backup has more variance but lower bias than a k-step backup. To reiterate, "bias" means that the term is a wrong/incorrect, but in the limit it should converge. The usefulness of introducing bias is that it decreases variance, which should increase sample efficiency. The risk of introducing bias is that because it is incorrect, it could lead the algorithm "down the wrong road" and possibly cause the algorithm to fail to converge or take too long to converge.

  Max Bias $\leftarrow$ ——————————————— $\rightarrow$ **Max Variance**

$\hat{Q}_{\mathbf{w}_i}$	1-step	2-step	...	$\hat{Q}_t^{(i)}$

Where:

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

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

$$\text{2-step} := r^{(i)}(s_t, a_t^{(1)}, \ldots, a_t^{(m)}) + r^{(i)}(s_{t+1}, a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)}) + \mathbb{E}_{s_{t+2} \sim p(s_{t+2}|s_{t+1}, a_{t+1}^{(1)}, \ldots, a_{t+1}^{(m)})} \left[V^{\pi_{\theta_i}}(s_{t+2}) \right]$$

$$\vdots$$

$$\hat{Q}_t^{(i)} := \sum_{t'=t}^{T-2} r^{(i)}(s_{t'}, a_{t'}^{(1)}, \ldots, a_{t'}^{(m)})$$

4.2 Multi-Agent REINFORCE with baseline

[Last Updated: August 2020]

4.2.1 Definitions

(Same as REINFORCE)

4.2.2 Assumptions

(Same as REINFORCE)

4.2.3 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 $V^{\pi_{\theta_i}}$

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

4.3 Multi-Agent Actor-Critic Q with baseline

[Last Updated: August 2020]

4.3.1 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:

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

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

4.3.2 Assumptions

- 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:

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

4.3.3 Algorithm Derivation

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

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

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

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

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

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

($\hat{Q}_{\mathbf{u}_i}$ 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 $V^{\pi_{\theta_i}}$

4.3.4 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 $y_{j,t} = \sum_{t'=t}^{T-2} r^{(i)}(s_{j,t'}, a_{j,t'}^{(1)}, \ldots, a_{j,t'}^{(m)})$.

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

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

[Last Updated: August 2020]

4.4.1 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:

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

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

4.4.2 Assumptions

- 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:

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

4.4.3 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:

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

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

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

4.4.4 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 $V^{\pi_{\theta_i}}$

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

4.5 Multi-Agent Actor-Critic SARSA with baseline

[Last Updated: August 2020]

4.5.1 Definitions

(Same as AC TD)

4.5.2 Assumptions

(SAME as AC TD)

4.5.3 Multi-Agent Bellman Rollout Derivation

(SAME as AC TD)

4.5.4 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 $V^{\pi_{\theta_i}}$

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

$\mathbf{u}_i \leftarrow \mathbf{u}_i + \alpha_3 \nabla_{\mathbf{u}_i} \text{loss}(\hat{Q}_{\mathbf{u}_i})$.

5 Analysis, Thoughts, Tips and Tricks

5.1 Context

5.1.1 Policy Network

For all PG algorithms, suppose the policy network learns a Gaussian distribution using a deep NN architecture as portrayed in `Figure 2`.

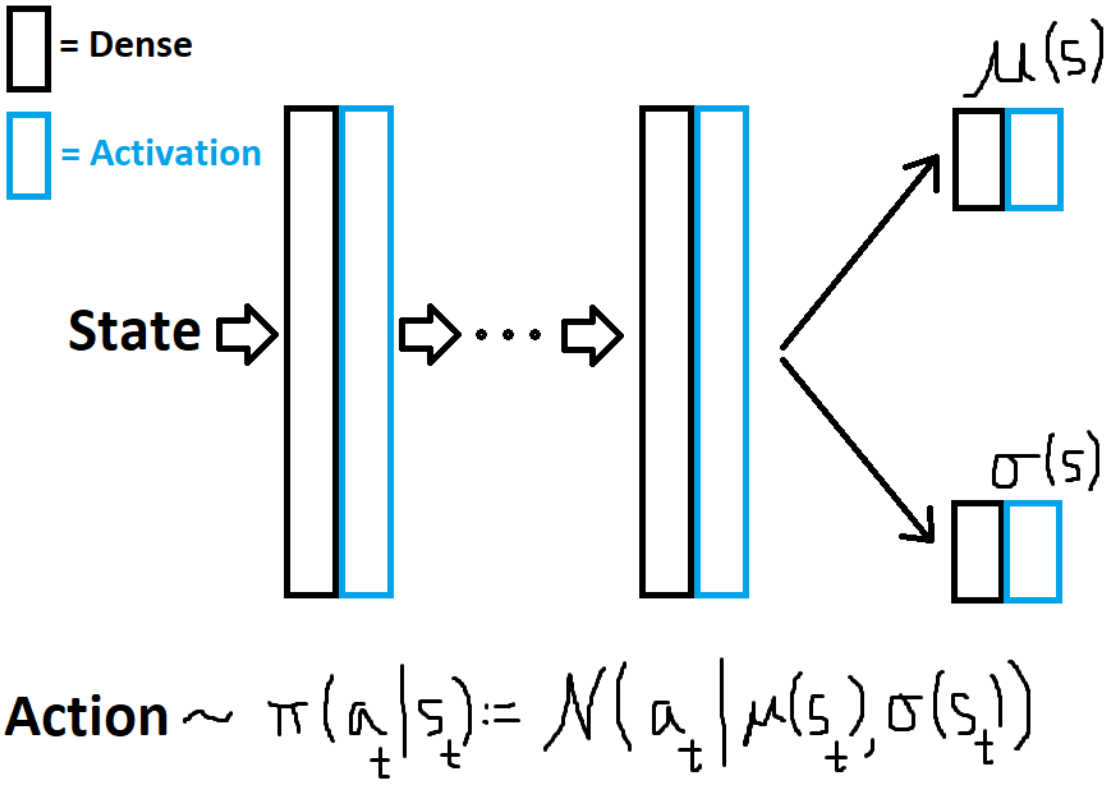

Figure 2: Policy Network Architecture

Where the activation layer is any non-linear function (e.g. ReLU, leaky ReLU, ELU, etc.). Furthermore, it is vital to do state and action scaling, otherwise the policy weights can blow up.

5.1.2 Value Network

For algorithms that learn Q or V, assume a standard setup: one or more dense layers taking state (and action for Q) as input and returning a Q or V value as output. Assume ReLU activation functions.

5.2 The Fundamental Problem, from an RL perspective

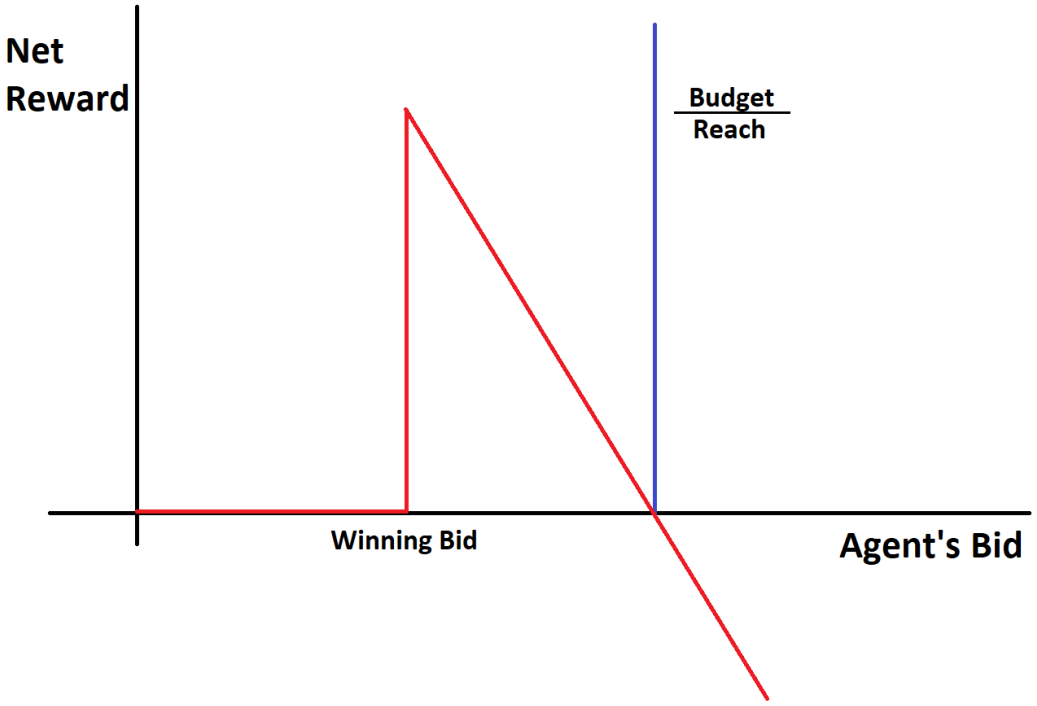

Figure 3: Fundamental Reward Curve

On any given day, the agent's reward is the payout that day minus the spend that day. On most days, though, the payout is 0 since campaign payout happens on the last day. However, for intuitive and visual purposes, suppose we split up the last day's payout evenly among all the days - thus making the daily payout meaningfully non-zero for all days. Let *net reward* be this modified agent reward.

The reward curve in `Figure 3` captures the essence of the agent's problem and has some notable aspects:

- If the agent spends less than the winning bid, then it gets no reward [5]. 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 [6]. Any higher bid only results in decreasing its reward [7]. The reward curve is **not flat** in the region [winning bid, $\frac{\text{budget}}{\text{reach}}$).

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

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

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

- 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, winning bid) and 2) the **not flat** region [winning bid, ∞). The two regions are "connected" via a discontinuity.

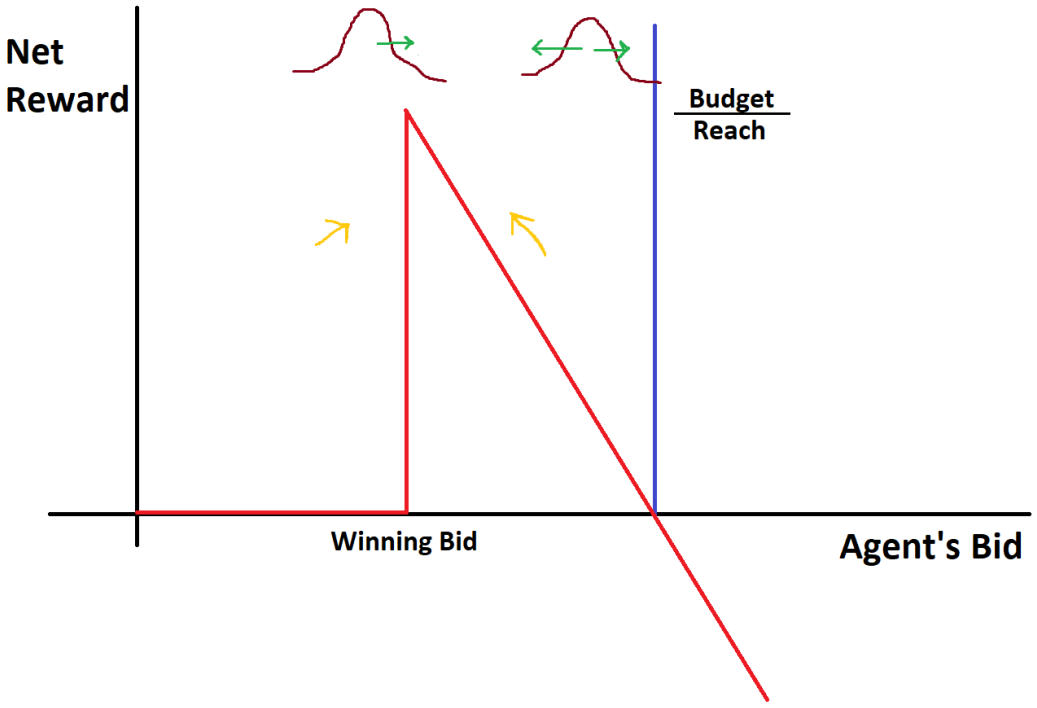

Figure 4: Learning a (Gaussian) policy

Recall that the core pieces of every PG algorithm are:

$$\nabla_{\theta_i} J(\theta_i) = \sum_{t=0}^{T-2} \nabla_{\theta_i} \log \left(\pi_{\theta_i}(a_t^{(i)}|s_t) \right) \hat{Q}_t^{(i)}$$
$$\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} J(\theta_i)$$

This is similar to cross-entropy loss in supervised learning, except it is weighted by the reward $\hat{Q}_t^{(i)}$.

`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.

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 $\hat{Q}_t^{(i)}$.
- Larger step sizes have higher risk of the policy "jumping off the cliff" once it nears the edge.
- The step size 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 domain are sensitive to the learning rate, which we will call the **zero collapse problem**. It further implies that it is worth trying techniques that decrease the effective step size as the policy climbs up the hill.

Although we do not know exactly what the reward curve would look like in higher dimensions, we believe the core aspects of the fundamental problem 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.

5.3 Tricks to combat zero collapse

Given the importance of the zero collapse problem, we experimented with several ways to alleviate the issue.

5.3.1 Policy network tricks

- Don't use random weight initialization

  Instead of doing random weight initialization, initialize the weights so that the policy is roughly centered around the $\frac{\text{budget}}{\text{reach}}$ threshold [8]. Doing so starts the policy off in the climbable region and will have minimal initial spend.

[8] Specifically, use samples from the state space to calculate the average budget per reach. Then pass this to the policy constructor.

- Don't use ReLU activation functions

  Although ReLU is the de facto activation function for neural nets, it is highly susceptible to the zero collapse problem. In particular, ReLU(x) has the property that its slope is constant as x goes from ∞ to 0. This means that the same $\Delta\theta$ near zero and not near zero move the output by the same amount. Or to put another way, the effective step of the policy is the same regardless of how close to $a_t = 0$ (i.e. bid $= 0$) the policy already is.

  In contrast, activation functions like softplus are better suited as the slope becomes shallower as they approach zero. See `Figure 5`.

  In general, I recommend softplus on the μ and σ layers and either leaky ReLU or ELU on all other layers. I recommend against tanh or sigmoid because they saturate too quickly.

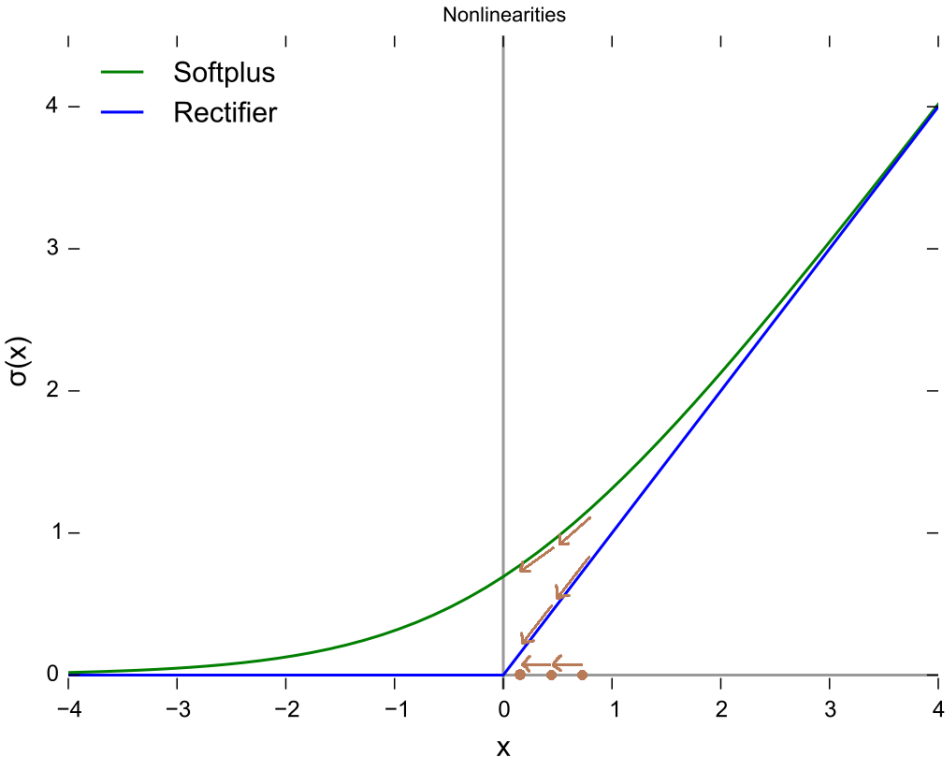

Figure 5: ReLU vs. Softplus

- Learn from an offset

  Let offset be the budget per reach threshold. Rather than use activation(x), instead use activation$(x - \delta \cdot \text{offset})$, where offset is such that activation$(-\delta \cdot \text{offset}) = 0$. This

trick works well with the softplus activation function, as it better utilizes the decreasing slope as $x \to -\infty$.

5.3.2 Use a baseline

Canonically, baselines are useful because they reduce variance. However, they provide further use in our problem domain.

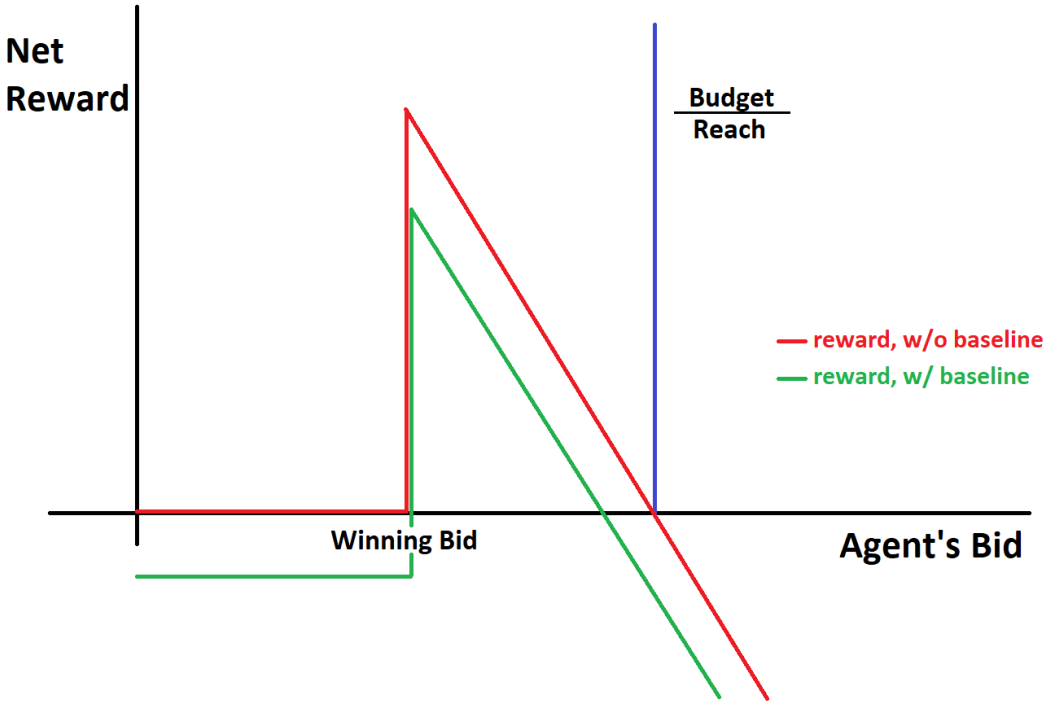

Figure 6: Reward curve with baseline

As seen in `Figure 6`, a baseline can turn the flat and zero region into a flat and negative region. This negativity causes the gradient to be negative, which in turn causes the policy to move away from the region. In other words, the negative value near the cliff acts like a nice buffer against zero collapse [9].

However, there is a hidden danger of using a baseline: it can lead to **thrashing**. As seen in `Figure 7`, a policy at position 1 might jump off the cliff to position 2. However, the now negative reward experienced at position 2 can lead the policy to jump back position 1 (or to the left or right of it). As one could imagine, the back-and-forth jumps between positions 1

[9]Keep in mind though that a policy wholly in the zero collapse region will not make any progress, as the flatness would result in the policy simply moving back and forth over the same spot (kind of like walking on ice).

and 2 can degenerate in different ways (e.g. equal amounts of back-and-forth jumps leading to forever oscillation, increasing back-and-forth jumps leading ultimately to zero collapse).

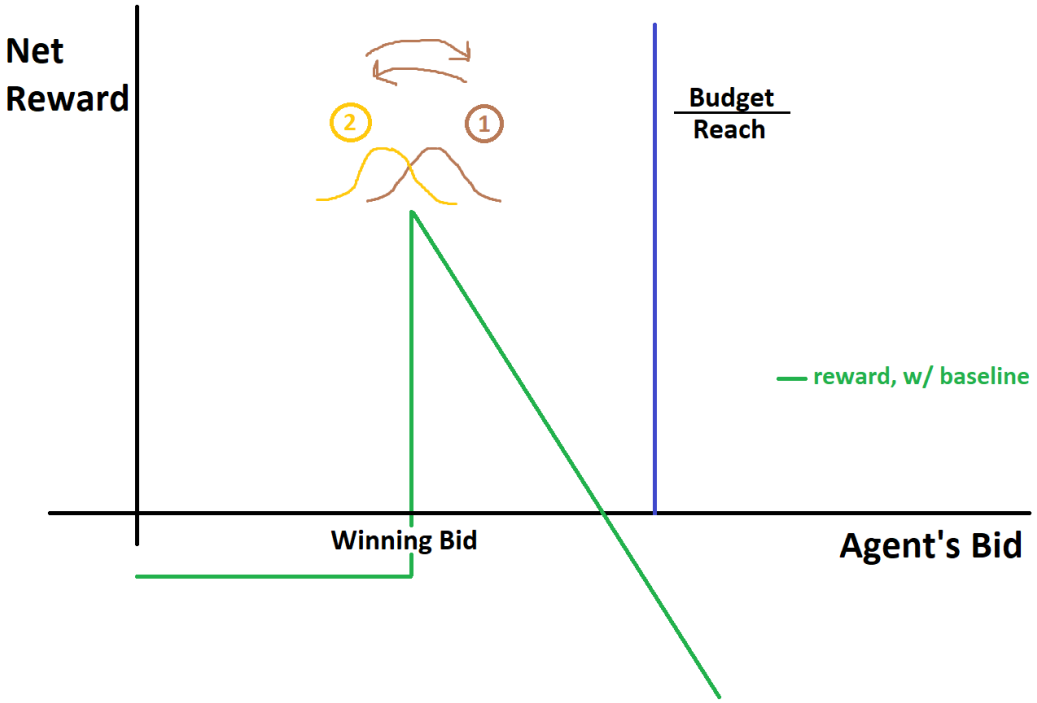

Figure 7: Baseline thrashing

5.3.3 Use an adaptive learning rate

Recall that a policy's parameters are updated as:

$$\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} J(\theta_i)$$

Another useful trick is to use an adaptive α instead of a fixed one. For example, use:

$$\alpha = \sqrt{\frac{\epsilon}{\nabla_{\theta_i} J(\theta_i)}}$$

For some fixed ϵ. This particular adaptive α has the property that $|\theta_{i,t+1} - \theta_{i,t}| \leq \epsilon$. Furthermore, the use of $\nabla_{\theta_i} J(\theta_i)$ in the denominator makes it agnostic to how high up the hill the policy is. Future work would be to use natural gradient methods, e.g. Trust Region Policy Optimization.

Along this vein, it is better to use SGD than ADAM; optimizers with momentum are very tricky to get working, as it seems the momentum exacerbates the zero collapse problem.

5.4 Actor-Critic Methods

While actor-critic methods are canonically beneficial because they reduce variance, their bias poses serious issues in our problem domain.

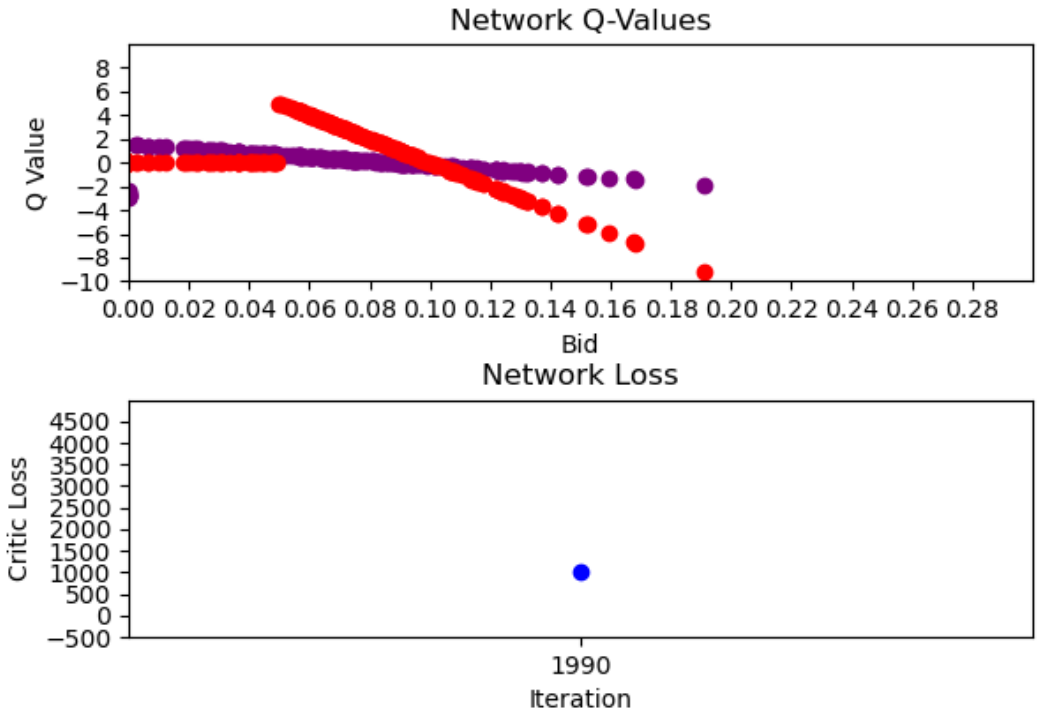

Figure 8: Fitting Q (purple) to actual reward (red)

As seen in `Figure 8`, even after roughly 2000 iterations, the Q value function was a poor fit to the actual reward curve. Furthermore, this early poor fit (i.e. bias) incentivizes the policy to zero collapse. In other words, this is the classic "bias leads down the wrong path" problem with value-based methods. And, unfortunately, the bias is particularly disastrous for our problem domain because of how difficult it is to recover from zero collapse [10].

[10]Empirically, we also see that the policy network converges far faster than the value network, implying that AC methods can only really work if the policy network is "throttled" in some way.

6 Results

Results are an ongoing collection, archived at:

`https://drive.google.com/drive/folders/1_UU7sV0vjmHkNzzO5EKoRPvbrpyqLnb-?usp=sharing`

Some sample results are included here.

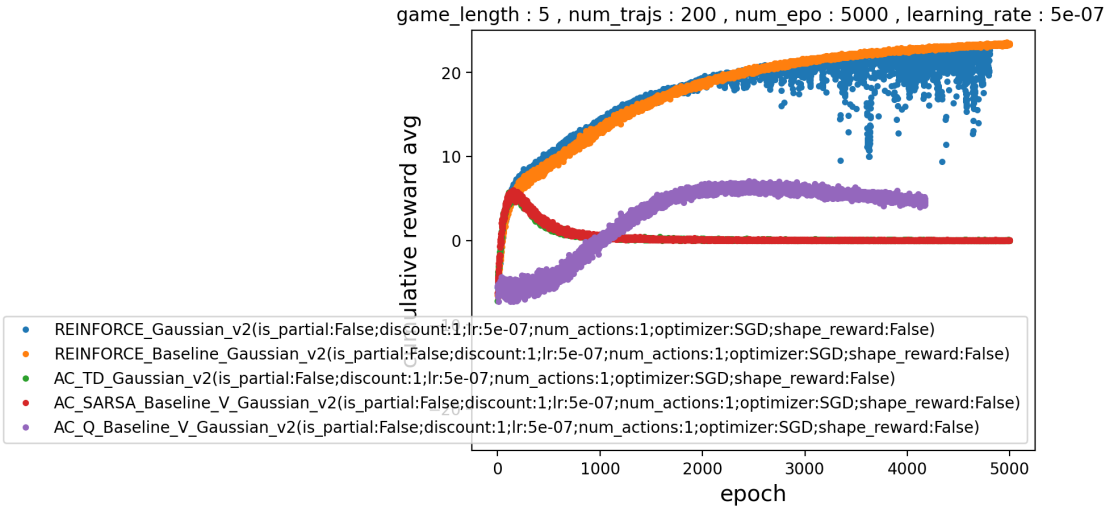

Figure 9: PG algos in 5 day game

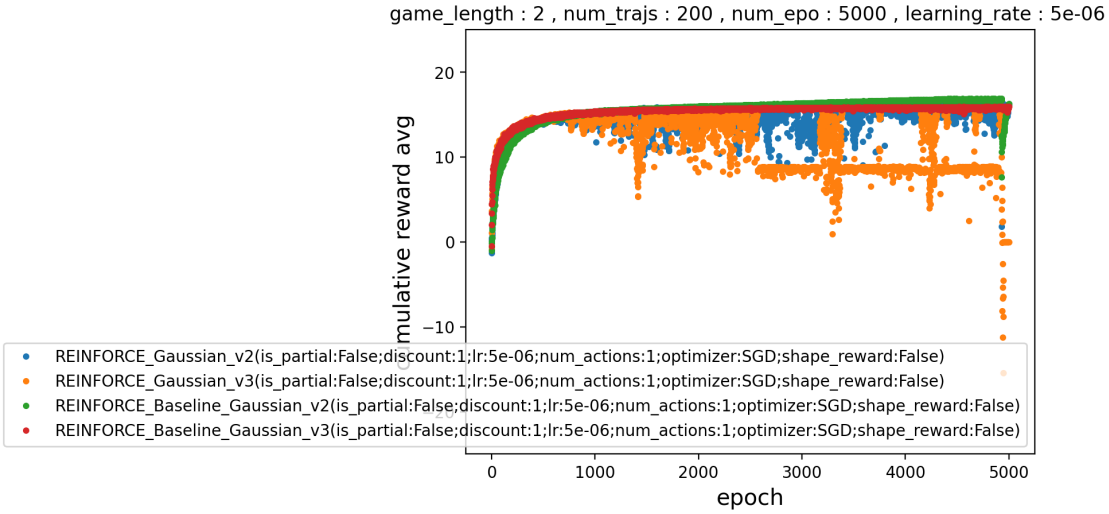

Figure 10: PG algos in 2 day game

7 Relevant Papers and Notes

- https://openreview.net/pdf?id=rkluJ2R9KQ
- https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec21.pdf
- http://128.148.32.110/people/gdk/pubs/fourier.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.5838&rep=rep1&type=pdf
- https://scholar.google.com/scholar?hl=en&as_sdt=0%2C40&q=Tabu+search+exploration+for+on-policy+reinforcement+learning&btnG=
- https://link.springer.com/content/pdf/10.1007/BF01720782.pdf
- https://ieeexplore.ieee.org/abstract/document/1224033?casa_token=mKZ2YXWZYlYAAAAA:eeLxrBG-dc00iULs8Of8w-YiAqqCaJtJejWBaT9SfherkrSCJ1YyQh6lAHZtv2BM3nB6FbYi96k
- https://arxiv.org/pdf/2002.01883
- https://arxiv.org/pdf/2003.04069.pdf
- http://proceedings.mlr.press/v19/slivkins11a/slivkins11a.pdf
- https://arxiv.org/pdf/2006.09585.pdf
- https://arxiv.org/pdf/1702.03037.pdf?utm_source=datafloq&utm_medium=ref&utm_campaign=datafloq http://www0.cs.ucl.ac.uk/staff/weinan.zhang/papers/ortb-kdd.pdf

8 Derby Framework

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

[NOTE: Derby is a private codebase and thus requires permission.]

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.