SOLUTION OF A MODIFIED MULTI-ARMED BANDIT PROBLEM BASED ON REINFORCEMENT LEARNING

Xiang Zheng
21307110169
mango789

Jiaao Wu
21307130203
Julius Woo

Zihao Cheng
21307130080
football prince

December 1, 2023

ABSTRACT

This report introduces a Reinforcement Learning (RL)-based solution for the *Santa* competition on *Kaggle*, inspired by the multi-armed bandit problem. First, we provide an overview of the classic model and the specific context of the game. Our methodology is then explained in detail, divided into several crucial steps. We begin by describing the initialization and updating process of the `threshold` variable. Following this, we delve into our agent's selection policy, emphasizing how it utilizes historical data for decision-making. We combine various policies together whose spirits can be categorized into exploitation, exploration and imitation. Moreover, we demonstrate the effectiveness of our approach through the agent's performance in *Kaggle*. Finally, the report concludes with a discussion on the limitations encountered during the development of our agent, along with potential areas for future improvement.

1 Introduction

The multi-armed bandit problem stands as a classic model for decision-making in uncertain environments. The name of this problem is inspired by a lineup of slot machines, often referred to as "bandits", each with a lever or "bandit". Behind each machine exists a fixed yet unknown probability of yielding a payout. Choosing to pull a lever equates to sampling from this hidden probability distribution.

In successive gambling, the core challenge is to decide which machine to play, when we don't know which one offers the best chance of winning. This problem illustrates the fundamental trade-off between exploration (trying out different options to gather information) and exploitation (making the best decision based on the information gathered).

Santa is a competitive two-player game where participants strategically choose from 100 vending machines (the "bandits") to interact with, in order to maximize their gains in the form of candy canes over 2000 rounds. This game is fundamentally based on the multi-armed bandit problem. However, it adds a twist: players' choices are visible to each other. Consequently, every decision impacts not just the player's own reward prospects but also the strategic thinking of their adversary. Therefore, *Santa* not only tests the players' ability to make decisions under uncertainty but also their skill in predicting and countering their opponent's strategy.

In this report, we will first outline the *Santa* problem modeling and the method for initializing the `threshold` variable, along with the strategy for its update. Following this, we will delve into strategic considerations tailored for our agent's policy, particularly focusing on its adaptability to historical selections. A significant part of our approach involves integrating several empirical policies to strike a balance between exploration and exploitation as well as imitating our respectable opponent. We will then showcase the performance of our agent in *Kaggle* competitions, providing the algorithm's effectiveness. To conclude, we will discuss the limitations we encountered in the development of our agent and propose potential directions for future enhancements.

2 The initialization and update of threshold

The difficulty of the *Santa* lies in the fact that our agent is blind to the `threshold` variable in `observation`, which stands for probability values for each machine payout on the current step. Hence, an excellent initialization and updating method of the `threshold` guarantees an outstanding performance of the agent. The inspiration of the method derives from the Bellman equation in Reinforcement Learning. The core ideas of our method are classified into two parts: initialization and update.

2.1 Initialization

Initialization of `threshold` begins the instant we build the framework of our agent in step 0. To find proper initial values, we collected the information of `threshold` in the *Kaggle* agent log json file and visualized these points in figure 1. The blue points stand for `threshold` while the red line stands for the uniform distribution over $(0, 1)$. We observe that the distribution of `threshold` is quite close to Uniform$(0, 1)$. However, if we initialize `threshold` as the uniform distribution mentioned above, the biases of our initial approximation may be very large. Besides, the biases may be spread and amplified in later rounds, leading to a terrible behaviour of our agent. Taken into account these above factors, a more conservative initialization method is implemented: initialize `threshold` around 0.5, but with additional noises. Here we just simply adopt the random noise with decimal up to 3 (i.e 0.001).

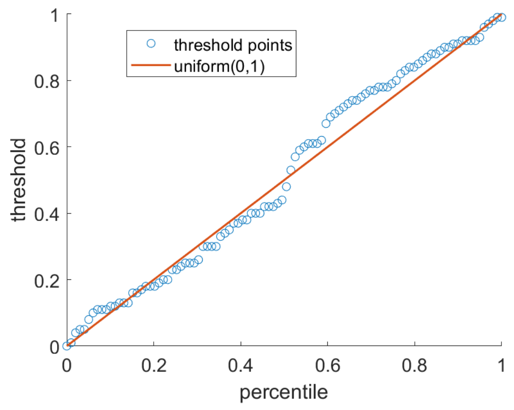

Figure 1: Initial distribution of threshold

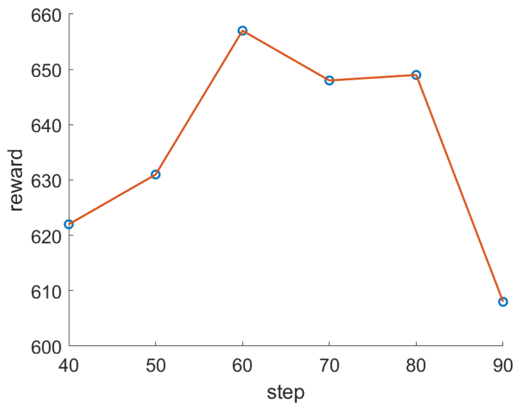

Figure 2: Reward of different look-back steps

2.2 Update

In the competitive interaction between the two agents, a relatively accurate approximation of `threshold` forms the basis of selecting an effective bandit, while the updating method proves vital in its estimation. Within our agent's framework, we have integrated the concept of the Bellman equation and modified it to suit this *Santa* problem. The update mechanism employed here can be considered a specialized adaptation of the value iteration technique commonly used in RL.

The Bellman update equation [1] is defined as

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s')$$

where a is the action, s' is the possible next state and γ is the discount factor.

In the specific *Santa* game, we take the value function V as the estimation of `threshold` and γ as the decay rate. For the reward function R, we replace it by the global variable `bandits_record` which records the historical information of reward and use it as the indicator of the best estimation rather than a correction factor.

At the beginning of each round, our agent records the information of last actions taken by both agents and judge whether our agent gets a reward or not. Then the agent stores these information in several global variables for the update of `threshold`. Unlike the *GridWorld* we encountered before in Lab 3, we choose to find the best estimation based on the information of `bandits_record` in this game. The estimation is found by minimizing the discrepancy from the expected value in a loop. After that, we multiply the estimation by the decay rate γ to get the next value of `threshold` in this bandit. Since the historical information of actions is already considered in the variable `bandits_record`, the result we get is well-defined. Notice that at most two bandits are chosen in a round, so we only need to update the `threshold` in the chosen bandit(s) each round.

3 Policy based selection

If an excellent initialization and update method of `threshold` is a guarantee of an excellent behaviour of our agent, an appropriate policy on the choice of a bandit in each round secures the execution of the full strength of our estimated `threshold`. Considering that there are a hundred bandits to choose from in a single round, we opt to use a passive learning based on some empirical results. Here in our agent, we hybrid several kinds of policies together to optimize our result. These policies can be categorized into three types: exploitation, exploration and imitation. Figure 3 and 4 show the experiment result in one single competition. Following are the throughout analysis of these policies:

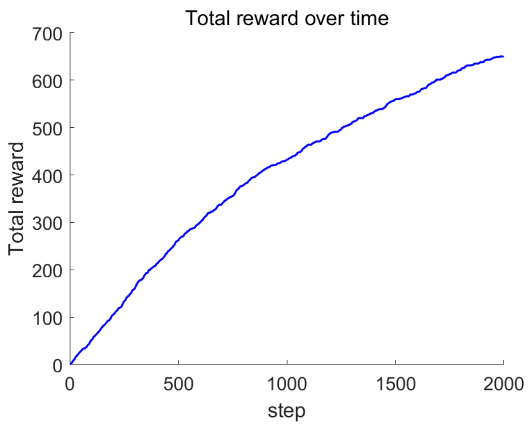

Figure 3: The total reward over steps

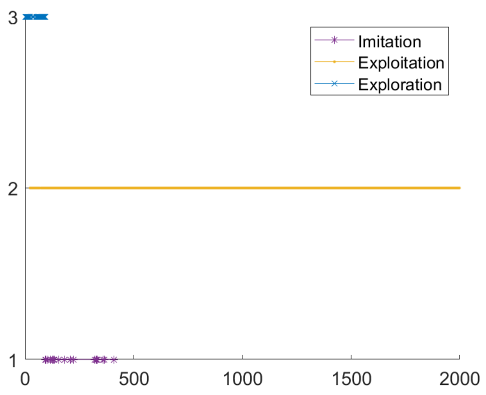

Figure 4: The type of policy chosen over steps

3.1 Exploitation

The estimation of the indicator variable `exploit` in each round is

```
exploit = (((2 * losses[my_best_bests] < wins[my_best_bests]) &
            (my_choices[my_best_bests] > 1) ).all()) or
            (thresholds[my_best_bests] > .63).all()
```

where `my_best_bests` is a list composed of the index of our top 10 thresholds.

This is a approximation based on empirical experiment results. It can be interpreted as our losses in these bandits are relative small than our wins or the estimation of `threshold` in these bandits are large enough, which indicates that the exploitation is already good enough. If the `exploit` is True, we'll find the next choice based on the exploitation policy: If the opponent's last action is already in our best choices, our agent will follow its action, otherwise the best choice with smallest last taken step is returned here. If the `exploit` is False, our agent will jump to the next policy.

3.2 Exploration

The exploration of our agent is relatively easy to understand by comparison with exploitation. Since the historical information of our opponent's action is considered in exploitation, we'll randomly select a bandit which is not chosen by both our agent and opponent's agent. If this kind of bandit doesn't exist, the best choice with smallest last taken step policy used in exploitation will be inherited here.

3.3 Imitation

Remember that *Santa* is not a single agent game but a fierce competition between two different agents. The opponent's agent we encounters in every competition can be stronger than, equal to or weaker than us. Since our goal of one single competition is to beat opponent's agent, why don't we learn from its strategy and imitate its behaviour to do better than it? Also notice that if we are stuck with our own policy during the whole competition, we are somehow confined to our own optimal "world", let alone beating our opponent's agent. Hence we image our opponent's agent as a very powerful agent and apply the following two policies:

```
# policy used in small step
if opponent[last_action] > 1 and my[last_action] <= 1:
    return last_action
# policy used in large step
if exists_any_not_chosen:
    if num_opponent_last_action > 1 and my[last_action] <= 4:
        return last_action
```

where the `last_action` in the above codes is the last action taken by our opponent's agent and the imitation is executed before the exploitation and exploration in each round.

To avoid the mislead by a weaker agent and the dependence of our agent on opponent's agent, we also develop a look-back correction mechanism: only the last 60 actions taken by our opponent's agent will be considered. The number 60 comes from our experiment results and you can justify it in figure 2. Since stronger agent chooses more wisely while weaker agent behaviours somewhat "weirdly" as the competition progresses, the weird action taken by weaker agent will be rejected by our agent while the wise action will be accepted and returned as the next action with reference to our own historical information of actions. The mechanism works pretty well in our agent and the result of performance will be shown in the following conclusion section.

4 Experiment results

The *Kaggle* scores of our different versions of agents are listed below:

version	characteristic	score
agent-test	UCB	603.1
agent-test-2	ϵ-greedy	1202.2
agent-2	traditional RL	1380
agent-7	RL with imitation	2824.1

We can find that there is a gigantic gap between the last submitted agent and other agents in score, which may due to the small number of teams in the competition. The score is not so accurate that it can only been viewed as a reference rather than the only standard for the evaluation of the agent.

5 Discussion

Based on the experimental results presented above, alongside insights gained through various trials and errors, we will now discuss some limitations of our current approach and outline potential directions for future work.

1. We notice that the ϵ-greedy method behaves relatively wiser in the early rounds of the competition. Hence the method can be hybrid with our current RL agent to achieve better performance. Here we'll use ϵ-decreasing strategy which is a natural variant of the method to prevent the sub-optimal from occurrence. [2]
2. Although a look-back correction mechanism is already developed for our agent to avoid the mislead by opponent's agent, sometimes it's still beaten by weaker agent. This forces us to develop a more effective mechanism to diminish these errors.
3. To eliminate the chances of wrong exploration when the exploitation is sufficient, the VDBE-softmax method can be added here. It'll outperform ϵ-greedy, Softmax and VDBE policies in combination with on- and off-policy learning algorithms such as Q-learning and Sarsa. [3]

References

[1] Russell, Stuart, and Peter Norvig. Artificial Intelligence: A Modern Approach. 4th ed., Pearson, 2020.

[2] Vermorel, J., Mohri, M. (2005). Multi-armed Bandit Algorithms and Empirical Evaluation. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds) Machine Learning: ECML 2005. ECML 2005. Lecture Notes in Computer Science(), vol 3720. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11564096_42

[3] M. Tokic, G. Palm, Value-Difference based Exploration: Adaptive Control between epsilon-Greedy and Softmax, in: KI 2011: Advances in Artificial Intelligence, Springer Berlin Heidelberg, 2011, pp. 335–346.