Agent-based Reinforcement Learning in Colonel Blotto

Joseph Christian G. Noel

1 Introduction

Models and games are simplified representations of the world. There are many different kinds of models, all differing in complexity and which aspect of the world they allow us to further our understanding of. In this paper we focus on a specific instance of agent-based models, which uses reinforcement learning to train the agent how to act in its environment. Reinforcement learning agents are usually also Markov processes, which is another type of model that can be used. We test this reinforcement learning agent in a Colonel Blotto environment[1], and measure its performance against a Random Agent as its opponent.

2 Colonel Blotto

Colonel Blotto is a constant-sum game proposed by Émile Borel in 1921[1]. In the game two or more players distribute resources (or coins) over several fronts in a battlefield. A player wins a front if they have allocated more resources to it than the other player. The objective of the game is to win more fronts than your opponent.

A notable thing about Colonel Blotto is that there is no optimal best strategy that will beat all other possible strategies. This means that the best strategy can change in a given game depending on the opponent and what the strategy the opponent is using. One way to play Colonel Blotto would be to update your strategy over time as you play the same opponent multiple times. If the opponent's strategy is constant or doesn't change much, it must be possible to eventually "learn" a strategy that would win against this opponent more often than it loses. One way to do this would be to create an agent-based model that would update itself over time as its plays an opponent repeatedly.

3 Reinforcement Learning

Reinforcement Learning (RL) is a form of agent-based modeling. In RL an agent learns by performing actions which changes the state of an environment. After each action, the agent

[1] Source code for the experiments can be found at https://github.com/jinonoel/BlottoRL

may also receive a "reward" whose value depends on how close the agent is to what it wants to achieve. The goal for the agent is to maximize the cumulative reward that it receives over all the actions that it takes. After a series of actions, the agent eventually reaches a goal state or terminal state, which signifies the end of an episode. The environment is then reset and the agent starts again from an initial state and the process then repeats itself.

Formally, an RL model is a set of states S, a set of actions A, and transition rules between states depending on the action taken. For state $s \in S$ at time t, an agent performs an action $a \in A$, moves to a new state s' and receives a reward r_t. The goal of the agent is to maximize the expected reward Rt,

$$R_t = \sum_{k=0}^{\infty} \lambda^k r_{t+k} \tag{1}$$

where $0 \leq \gamma \leq 1$ is a discount factor for handling infinite horizons.

$\pi(s, a)$ is a probability mapping of an agent taking action a while in state s. A proper policy is one where there is a non-zero probability of reaching a terminal state. There is always an optimal policy π^* that is better than or equal to all other policies when it comes maximizing the cumulative rewards.

3.1 Markov Decision Process

Calculating the optimal policy in general is a hard problem, especially if we need to keep track of the history of all states, actions, and rewards. To simplify things, reinforcement learning problems are usually constructed as Markov models, such that the information from the history of all states, actions, and rewards before time t is encapsulate in the current state at time t. This is the Markov property, and tasks which exhibit this property are called Markov Decision Process (MDP). Formally, the Markov property states that for transition probability function Pr,

$$Pr(s_{t+1}, r_{t+1} | s_t, a_t) = Pr(s_{t+1}, r_{t+1} | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_0, a_0, r_0) \tag{2}$$

3.2 Value Function

The state-action value function Q is the estimate of the expected rewards the agent will receive by being at state s, at time t, taking action a, and then following policy π from $t + 1$ onwards.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}(R_t | s_t, a_t) \tag{3}$$

The reinforcement learning problem can therefore be reduced to finding the optimal policy π^* by simply choosing the action a that maximizes $Q(s, a)$ for all $s \in S$

$$\pi^*(s) = \arg\max_a Q(s, a) \tag{4}$$

3.3 Q-Learning

Q-Learning is one of the basic reinforcement learning methods, and takes full advantage of the Markov property of the RL model.. It approximates the state-action value function $Q(s, a)$ in a recursive manner using a weighted average of the old value and the new information retrieved from the action and subsequent reward.

At each time step Q-Learning approximates the optimal policy by iteratively updating Q in the following manner, where $\alpha > 0$ is the learning rate, and $0 \leq \gamma \leq 1$ is again the discount factor:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{5}$$

4 Experiments

We now show how a Q-Learning agent performs against a random agent in Colonel Blotto.

4.1 Experimental Setup

We setup a 2-player Colonel Blotto game with 3 fields and 10 coins per player. This provides a total of 66 possible actions a (ways to distribute the 10 coins on the 3 fields) that the players can choose from. The players aim to distribute their 10 coins among the 3 fields in such a way that their coins are greater than their opponent's coins in each field. The winner of the game is the player that wins more fields than their opponent, and received a reward of $r = 1$ from the environment. The loser of the game receives a reward of $r = -1$. A single episode in reinforcement learning constitutes one game. We count how many games each player has won over time and show the results. .

Our reinforcement learning agent uses the Q-Learning algorithm for approximating the optimal policy. We use $\alpha = 0.1$ as the learning rate and $\gamma = 1$ as the discount factor. We also set $\epsilon = 0.2$ as the exploration rate for the agent. For the opponent, we use a RandomAgent that will select one of the 66 possible actions at random with equal probability for all of them. The experiments were run using the OpenSpiel[2] environment simulator.

4.2 Results

We run 1000 games of Colonel Blotto to see the performance of the reinforcement learning agent. As shown in Figure 1, it takes almost 200 games before the RL agent really begins to start defeating the random agent regularly. In fact, if we look at the just the first 100 games (Figure 2), both agents win games at practically similar rates. However after 1000 games, the improved win rate of the RL agent can already be clearly seen.

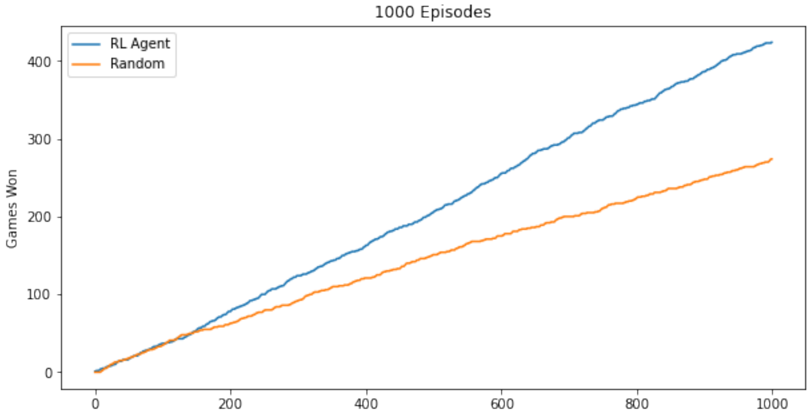

Figure 1: Performance of RL Agent vs Random Agent

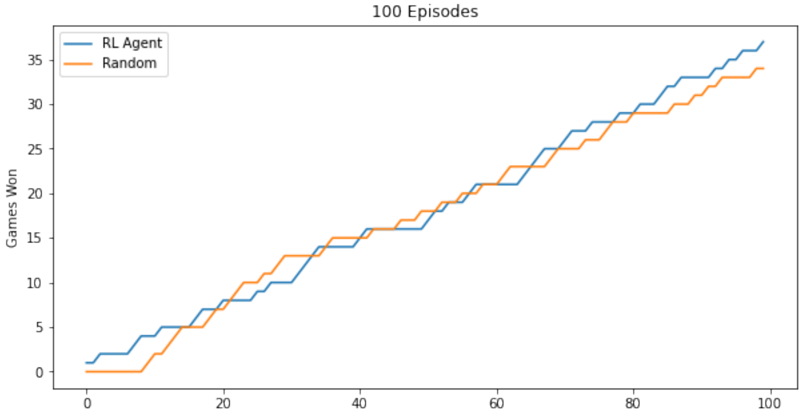

Figure 2: First 100 games of RL Agent vs Random Agent

We also dive deeper and look at the Q-values that the agent has assigned to each of the 66 possible actions, to better understand what strategy it has arrived at. We continue running the episodes to 1 million games to optimize the Q-values further before showing the following results. Below we show the top 5 and the bottom 5 actions based on their

Q-value scores. The 3 numbers for each action signify the number of coins that the RL agent assigns for each of the 3 fields.

Top 4 Actions
[4, 0, 6]
[5, 0, 5]
[0, 4, 6]
[6, 1, 3]

Bottom 4 Actions
[0, 0, 10]
[0, 8, 2]
[0, 10, 0]
[10, 0, 0]

As can be seen, the actions with the lowest Q-values are the ones where the agent puts all their coins in just one front, guaranteeing a loss when their opponent easily wins the other two fronts. On the opposite end, the best actions are the ones where the agent splits their coins almost evenly among just two fronts, with 0 coins allocated to the third front. By sacrificing one of the fronts, they give themselves the best chance to win the other two fronts they focused on, and have a high probability of winning the game in the process. Against a Random Agent, this results in the high win rates that were shown in the earlier graphs.

5 Conclusion

We have shown how a RL agent using Q-Learning performs against a Random Agent in 1000 games of Colonel Blotto. The RL doesn't do at well at first, and actually has quite similar win rates to the Random Agent over the first 100 to 150 episodes. It takes almost 200 episodes for the RL agent to start differentiating itself to the Random Agent, and by 1000 episodes the RL agent has started to clearly show its dominance.

We also investigated what strategies the RL agent has learned through its repeated playing, and find that its preferred strategy is to sacrifice one of the fronts and focus on the other 2 fronts, splitting its coins almost evenly between them. This was shown empirically to be a a winning strategy against the Random Agent

References

[1] E. Borel. The theory of play and integral equations with skew symmetric kernels. *Econometrica*, 21(1):97–100, 1953.

[2] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. D. Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and

J. Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.