Policy Based Reinforcement Learning: REINFORCE and Actor-Critic

Josef Hamelink [1] **Koen van der Burg** [1]

Abstract

The Catch Environment is a custom-built environment that is used in this paper to test the performance of two widely used policy-based reinforcement learning algorithms, REINFORCE and Actor-Critic. The research analyzes how policy gradient variance, bootstrapping, baseline subtraction, and entropy regularization affect agents performance. The results reveal that the Actor-Critic agent beat REINFORCE in terms of convergence time and average performance that were found. But the fact remains that both agents' performance was significantly dependent on the hyper-parameters used to train them. The application of entropy regularization with fine-tuned hyper-parameter values improved policy gradient approaches by encouraging action space exploration. Meanwhile, bootstrapping with n-step: 3 had no effect on the final results, and combining bootstrapping and baseline subtraction had minimal variance reducing effect. Nevertheless, the study provides insights into the behaviour and strategy of the agent in this custom-world through various environment changes.

1. Introduction

Reinforcement learning is an popular concept that enables agents to learn how to interact with complex environments and achieve specific objectives. This study aims to examine the performance of two popular reinforcement learning algorithms, REINFORCE and Actor-Critic, on the challenging Catch Environment task.

1.1. Goals

The goal is to look into a number of reinforcement learning-related issues. Without regard to order, we would like to quickly summarize the primary study objectives. First, we test whether the commonly held assumption that the policy gradients employed by the REINFORCE method have significant variance is accurate. (**?**) In the context of an actor-critic agent, our second goal is to examine the effects of bootstrapping and baseline subtraction, both separately and jointly, on the variance of policy gradients and average performance. Third, we contrast the two agents' performances as well as their individual features. Fourth, we investigate how agent performance is affected by entropy regularization. Finally, we look at the agent's performance in various environment conditions and try to understand why it varies. We hope to shed light on the advantages and disadvantages of various reinforcement learning techniques as well as their possible applicability in real-world situations by addressing these important concerns.

1.2. Agents

As we know, reinforcement learning is a branch of machine learning concerned with teaching agents on decision making in varying and dynamic situations. One particular method for reaching this objective is policy-based reinforcement learning. Instead of maximizing the value function, which predicts the expected future reward of a certain state-action pair, this technique directly optimizes the policy function, which maps states to actions. REINFORCE and Actor-Critic are the two policy-based reinforcement learning algorithms used in this study.

The REINFORCE algorithm is a straightforward and widely used policy gradient technique. By altering the policy parameters in the direction of the expected reward's gradient, the algorithm directly optimizes the expected cumulative reward. REINFORCE, however, is supposed to have a number of drawbacks, including significant variance in its gradient estimations, which can cause sluggish convergence and unstable training.

The Actor-Critic algorithm, on the contrary, integrates components of value-based reinforcement learning with policy-based reinforcement learning to create a more sophisticated policy-based reinforcement learning strategy. An actor that learns the policy and a critic that learns the value function make up the algorithm. By using the estimated value func-

[1]LIACS, Leiden University, Leiden, Netherlands. Correspondence to: Thomas Moerland <LIACS>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

tion as a baseline, the actor adjusts the policy parameters, lowering the variance of the gradient estimations and resulting in more stable training. On the other hand, the critic learns the value function by calculating the anticipated future reward of a specific state-action pair. Thus, the actor is responsible for selecting actions based on the current policy, while the critic evaluates the quality of the actor's actions based on the estimated value function.

1.3. Environment

In this study, we use a custom-built environment called the Catch Environment, provided within the assignment, to evaluate the performance of our reinforcement learning agents. The agent is situated at the bottom of the grid and has horizontal motion capabilities in the Catch Environment, a straightforward two-dimensional grid world. Each drop interval, the environment generates a ball that drops in a straight line down the grid. The agent's job is to capture the ball by moving left, right or standing idle, before it hits the bottom of the grid. The agent gets a positive reward (+1) if it successfully catches the ball. But, it receives a negative reward (-1) upon missing the ball. Between the initial spawn, and the drop down, the agent is not compensated or punished while the ball falls to the bottom of the grid. The game ends on either of two conditions; after 250 time steps have passed since the beginning of the game, or the agent has failed to catch 10 balls.

In order to evaluate the robustness of our reinforcement learning agents, we also created different versions of the Catch Environment. We specifically change the environment's size and the ball's dropping speed. Additionally, we contrast how well our agents perform while interacting with the environment using a vector-based method and a pixel-based approach. We examine the generality of our agents and gauge their capacity for environmental adaptation thanks to these changes.

2. Agents

The two agents that we employed in our study will be covered in this part. To convey ideas found in both the REINFORCE and Actor-Critic agents, we will first introduce the Policy-based (basic) agent. The approach, hyperparameter tuning, and results of the REINFORCE agent will then be covered in detail. Following that, we'll go into detail on the Actor-Critic agent, including its methods, hyperparameter optimization, outcomes, and comparison to the REINFORCE agent.

2.1. Policy based

2.1.1. Methodology

2.1.2. Hyper parameter optimization

2.2. REINFORCE

2.2.1. Methodology

2.2.2. Hyper parameter optimization

2.2.3. Results

2.3. Actor-Critic

2.3.1. Methodology

2.3.2. Hyper parameter optimization

2.3.3. Results

2.3.4. Comparison REINFORCE & Actor-Critic

3. Environment variation - Actor Critic

In this section, we look at how changes in the environment affect the agent's performance. The portion was broken into two sections: Pixel approach variations & Vector approach variations. We chose the Actor-Critic agent because it performed more successfully than the REINFORCE agent. For each approach we explore how environmental differences effect performance in environment variable variation. The following variations are taken in consideration: size variation, speed variation, and size and speed variation combined. We evaluate the agent's performance and observe the strategies it learns and applies for each variation.

The state space represents the information an agent knows about its surroundings at any one time, and it serves as the foundation for the agent's decision-making process. When establishing the state space, one key choice is whether to represent it as a vector or as a collection of pixels. The vector representation is a numerical description of significant environmental data, such as position of paddle (x) and lowest ball (x,y), in a fixed-length vector. A pixel representation, on the other hand, includes capturing the state of the environment as a matrix of pixel values, in a binary-two channel pixel array. While vector representations are more compact and economical, pixel representations provide more detail and allow for more complicated decision-making. Pixel representations, however, might be computationally expensive and demand more processing power. In the following part we explore the results of the different representations on variations of the environment. Both agents were initially trained on a 7x7 grid at 1.0 speed unless specified else specifically. For example, due to the difference of state space's between the trained model and the environment, extra models were trained for the pixel representation with varying environment sizes.

Speed	Avg. reward	Max possible reward
0.5	17.24	18
1.0	33.70	35
1.5	35.5	62

Table 1. This table shows the results of the Actor-Critic (pixel) agent interacting with the environment (7 x 7) testing different drop interval speeds. Agent was trained with: α: 0.001, β: 0.1, γ: 0.9, δ: 0.995, batch size: 32 and 100 budget.

Size (rows x columns)	Avg. reward	Max possible reward
7 x 14	-9.22	36
14 x 7	14.14	17
4 x 4	60.60	63

Table 2. This table shows the results of the Actor-Critic (pixel) agent interacting with different sizes of the environment with speed 1.0. Agent was trained with: α: 0.001, β: 0.1, γ: 0.9, δ: 0.995, batch size: 32 and 1000 budget.

Size (rows x columns)	Avg. reward	Max possible reward
7 x 14	7.56	9

Table 3. This table shows the results of the Actor-Critic (pixel) agent interacting with a combination of changes in the environment. Agent was trained with: α: 0.001, β: 0.1, γ: 0.9, δ: 0.995, batch size: 32 and 1000 budget.

3.1. Pixel

3.1.1. Speed

The first variation applied to the environment is a change of speed. After loading the model we render its behaviour and log the average reward over 100 episodes. The speed variable alongside the number of rows, together determine the drop interval speed, which defines what the max attainable reward is in certain amount of time steps. Where the drop interval speed is: $\frac{\text{rows}}{\text{speed}}$, if the division is higher than 1, else it is 1. The max reward can be seen in Table 1.

Each different drop rate affected the agent differently. Overall, it is noticeable that the pixel agent tends to hug the wall where the majority of balls have dropped. Because, it seems more logical to see the agent return to the middle in order to reset its position for the next ball. At 0.5 speed we noticed that the agent waits shortly alongside the wall until the ball gets closer to the bottom before catching it. When the speed is increased the agent does not hug the wall idly anymore, instead it has to keep moving to catch the balls. When passing a certain threshold the agents switches to hug the other side, if enough balls have dropped there. Thus it seems that the preferred state to wait, when possible, is close to a wall. Hugging either side sometimes causes the agent to overshoot the position of the ball, when forced to travel to the opposite side. At an high speed of 1.5, the agent still performed decent, where the average reward was 57% of the max possible reward. This is impressive since there were many more targets to keep track off, on which the agent was not trained.

3.1.2. Size

Next we applied changes in the dimensions of the environment, and logged the results as well. As mentioned in 3.1 different models were trained for each environment due to differences in state-spaces. The agent was tested with three different environment sizes (rows x columns); 7 x 14, 14 x 7 and 4 x 4. The row variation was again taken into account whilst defining the max reward. The results from this experiment are seen in Table 2.

Observing the agent in the first environment (7 x 14) where the strategy is to stick to one side, without hugging the wall, but too far too catch balls on the other side. This was to be expected when the agent did not converge after 1000 episodes of training. On an environment where the agent had less horizontal space and more vertical space (14 x 7) we notice that the agent instantly goes into the right direction. The problem is that the agent does not seem to have learned how to properly use the idle function, causing it to shake around the position of the ball, sometimes missing it. On the smallest environment (4 x 4) the agent performed really well, catching over 96% of the balls. Because the area was small, the agent locked into the target and had no time to shake to another position.

3.1.3. Combination

After inspecting the effect of environmental changes on the agent separately, we now combine them together. Since the model for the narrow environment (14 x 7) was already trained, it was applied here as well. Table 3 displays the result.

It's strategy was to remain in the middle, but upon enough balls falling on one side, a bias was generated. The agent therefor anticipated the next ball to fall somewhere in that region, which worked out most of the time. Again due to the long wait of 14 rows, and few balls, the agent started shaking, missing some of the balls.

3.2. Vector

The second agent type we tested was the vector interpretation of states. Due to the unique property the vector has, only one model has to be trained and can be applied to multiple environment changes. This is possible because the xy position of only the lowest ball is visible to the agent. If we would increase the speed above 1, then multiple balls would be in the field, but only the lowest one is spotted. This would make it impossible for the agent to learn a strategy to

Speed	Avg. reward	Max possible reward
0.5	17.14	18
0.75	27.08	28
1.0	33.74	35

Table 4. This table shows the results of the Actor-Critic (vector) agent interacting with the environment (7 x 7) testing different drop interval speeds. Agent was trained with: α: 0.001, β: 0.1, γ: 0.9, δ: 0.995, batch size: 32 and budget of 1000.

Size (rows x columns)	Avg. reward	Max possible reward
7 x 14	7.98	36
14 x 7	16.28	17
4 x 4	59.34	63

Table 5. This table shows the results of the Actor-Critic (vector) agent interacting with different sizes of the environment with speed 1.0. Agent was trained with: α: 0.001, β: 0.1, γ: 0.9, δ: 0.995, batch size: 32 and 1000 budget.

catch two consecutive balls in different locations.

3.2.1. Speed

Similar to the pixel agent the first variation applied to the environment is a change of speed. Again the change in the speed together with the amount of rows determine the drop interval speed. In Table 4 we compared the results of three different speeds; 0.5, 0.75 & 1.0.

Each different drop rate affected the agent differently. Overall the agents again tend to hug the wall, but at 0.5 speed we noticed that the agent would this time idle and move towards the ball when spotted. The agent sometimes reacted to slow due to an opposite movement, or overshot the target. When the speed is increased to 0.75 the agent tends to hug the wall shortly before moving to catch the balls. After catching it moves towards the preferred wall, until a new target shows up. At a speed of 1.0, on which the agent was trained, the agent does not hug the wall anymore and keeps itself more centered. This explains the behaviour of the agents on slower speed, where more time is available between balls, causing them to be attracted slowly to one wall.

3.2.2. Size

To compare the results, the vector agent is also tested on the three different sizes (rows x columns); 7 x 14, 14 x 7 and 4 x 4. Here again no new agent has to be trained, and a speed of 1.0 was used to render the environment. Just as the pixel calculation, the row changes were taken into account for the max reward. The results from this experiment are seen in Table 5.

On the first environment (7 x 14) the agent performs relatively well for the size of the world. It managed to obtain a positive average reward of 8 over 100 episodes. It is constantly busy targeting the next dropping ball, but sometimes the distance to travel is to great. On an environment where the agent had less horizontal space and more vertical space (14 x 7) we notice that the agent instantly goes into the right direction and hoverd around it. The problem is that the agent again shakes around the position of the ball, sometimes missing it. On the smallest environment (4 x 4) the agent performed really well, where hugging a side is no problem if the agent instantly moves when a ball is spotted.

Size (rows x columns)	Avg. reward	Max possible reward
7 x 14	8.7	9

Table 6. This table shows the results of the Actor-Critic (vector) agent interacting with a combination of changes in the environment. Agent was trained with: α: 0.001, β: 0.1, γ: 0.9, δ: 0.995, batch size: 32 and 1000 budget.

3.2.3. Combination

To finish the vector experiment, we also tested the combination of environment and speed changes. The results can be found in Table 6. The agent maintains its position in the middle as much as possible, and over time in the episode creates a small bias for one side. Also shakes slightly under the position of the ball, making it almost perfect.

4. Conclusion

4.1. Key takeaways

After comparing the performance of both agents, it was notice-able that the Actor-Critic agent performed better. Where the Actor-Critic converged earlier, with higher average rewards. The performance of both agents highly depended on the set of hyper-parameters used to train the agent, not just one set of hyper-parameters was correct.

Entropy regularization can help to improve the performance of policy gradient methods by encouraging exploration of the action space, by penalizing policies that have low entropy (i.e., that are too certain about their actions). The regularization hyper-parameter (β) thus showed to be of great importance to fine-tune. A high entropy regularization did not perform well overall, there were some outlier hyper-parameter sets, but these could also be lucky.

Alongside all the existing hyper-parameters, the Actor-Critic also had the option to use bootstrapping and/or baseline subtraction. We decided not to experiment with different n-steps with bootstrapping, since we already had many different configurations. A bootstrap with n-step: 3 seems to be to low for the chance of having a significant effect on the final results. The baseline subtraction, although the impact was low, it was noticeable that the variance was smaller.

When combining both, we saw no direct difference to the behaviour of just the baseline subtraction.

The REINFORCE agent, unlike the Actor-Critic, has no second network to regulate the first one. Here we indeed see that the policy gradients suffer from high variance, unless entropy regularization is applied, helping with said variance.

The final comparison was that of the two Actor-Critic (pixel & vector) on different environments and speeds. To start of, the flexibility of the vector network, without suffering too much information loss, worked great for this problem. It was able to traverse with relative ease through different environment size settings and ball drop intervals, when compared to the pixel agent. Speed changes were no problem for both, until multiple balls are in the playing field, on which both agents were not trained to handle. The vector agent did handle it better than the pixel agent was. Wide environments made it hard for the REINFORCE agent to even converge before 1000 episodes, causing it to do quite badly in comparison to the vector agent.

4.2. Future Work

4.3. Contributions

References