Policy-gradient based Reinforcement Learning

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 reduced variance and 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 no noticeable 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 a 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. All code written for this project is publicly available on GitHub[1].

1.1. Goals

The goal is to explore 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 (Sutton & Barto, 2018). 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 function as a baseline, the actor adjusts the policy parameters,

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

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

[1]https://github.com/Josef-Hlink/BasedRL

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

Both the REINFORCE and Actor-Critic agents are policy-based agents, i.e. they directly optimize the policy function. This policy function is a mapping from states to actions, and can be represented by a neural network. The technique to optimize this policy function is called policy gradient *ascent*, as it happens to be a maximization problem. This is in contrast with the more common policy gradient *descent* approach seen in most neural network applications, where the loss is defined as the gradient of the policy function, and the policy function is updated in the direction of the gradient's negative. The policy gradient ascent approach is used because the policy function is a probability distribution, and thus its gradient can not be viewed as a loss function.

2.1. REINFORCE

2.1.1. Methodology

The high-level procedure of REINFORCE is laid out in Algorithm 1.

Algorithm 1 REINFORCE

```
**Input:** $env, \pi_\theta, \alpha, \gamma$
Initialize policy parameters $\theta$
**repeat**
    Under policy $\pi_\theta$,
    sample episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$
    **for** $t = 0$ to $T-1$ **do**
        $G_t \leftarrow \sum_{i=t+1}^{T} \gamma^{i-t-1} R_i$
        $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \log \pi_\theta(A_t|S_t)$
        $\theta \leftarrow \theta - \beta\gamma^t (G_t - b(S_t)) \nabla_\theta \log \pi_\theta(A_t|S_t)$
    **end for**
**until** convergence
**Output:** $\pi_\theta$
```

One important thing to note is that we do not actually iterate of over each transition $(S_t, A_t, R_{t+1}, S_{t+1})$ separately. Rather, we use batch training, where we sample a batch of episodes, and then perform the gradient update on the batch. While taking larger batch sizes is computationally more efficient, it also results in a higher variance in the gradient estimates, which is something to be taken into account when setting other hyperparameters.

2.1.2. Hyperparameter optimization

All hyperparameter optimization was done using wandb's sweep functionality. Our sweep was set up to randomly explore different combinations of the following hyperparameters:

Hyperparameter	**Symbol**	**Search space**
learning rate	α	Uniform($10^{-4}, 5 \cdot 10^{-3}$)
entropy reg. coef.	β	Uniform(0.0, 0.5)
discount factor	γ	Uniform(0.7, 0.999)
α decay rate	δ	Uniform(0.99, 0.9999)
batch size	-	[$2^i \ \forall \ i \in [2, ..., 8]$]

Table 1. REINFORCE hyperparameters

A total of 200 runs were performed, each with a budget of 2500 episodes, after which performance was evaluated on 500 episodes. All of our hyperparameter sweeps are publicly available in one of our wandb projects [2].

[2] https://wandb.ai/basedrl/pbrl-sweeps-old

2.1.3. Results

The result of our sweep over REINFORCE's hyperparameter space is shown in Figure 1. This was not the first sweep we performed, as we iteratively tried to minimize the weight each individual hyperparameter had on the rewards the agent was able to achieve on the evaluation episodes. Stating that one hyperparameter is inherently more important than another is a difficult task, as the hyperparameters are not independent of each other, so multiple sweeps had to be done to find a good balance.

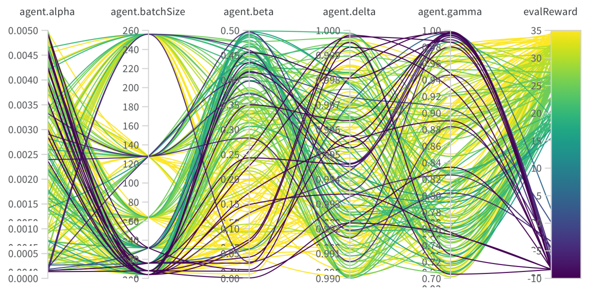

Figure 1. REINFORCE hyperparameter sweep results for our REINFORCE agent using wandb.

For the learning rate α, we see that generally, values between 5e-4 and 5e-3 result in the most consistent performance. We also see that employing higher batch sizes also tends to be more, not less beneficial when compared to their lower counterparts, which should in theory be more granular in their gradient estimates, as the network of the agent is updated more frequently. Perhaps the most interesting observation we can make from this figure is the relatively homogeneous section of the plot depicting regularization coefficient β with values between 0.05 and 0.25.

To further investigate the effect of this hyperparameter in isolation, we also ran the same agent for 10 independent iterations with a fixed learning rate of 0.0025, a discount factor of 0.8, a batch size of 64, and a learning rate decay factor of 0.99. The rewards over the episodes and the policy gradients runs are shown in Figures 2 and 3, respectively.

The first observation to make is that completely leaving out entropy regularization exhibits an unpleasantly high variance in the rewards the agent is able to achieve; some runs converge extremely early, while others never manage to perform above chance level. The way these curves are plotted is also important to note: the shaded area represents the standard error of the mean as min and max values were too noisy. When a single run is converged, the data is not stops appearing in the plot, which is why the mean drops down to the worse performing runs that never converge. Our agents with $\beta > 0$ do not exhibit this behaviour, and manage to achieve stable learning across all 10 runs.

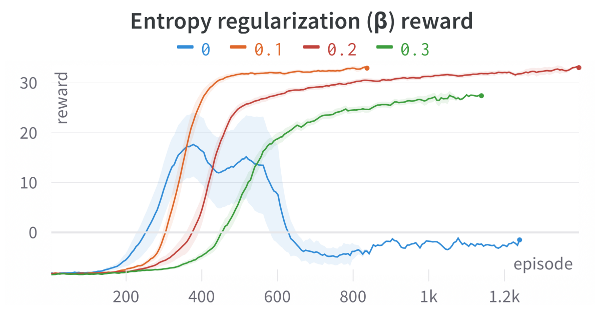

Figure 2. Entropy regularization reward learning curve.

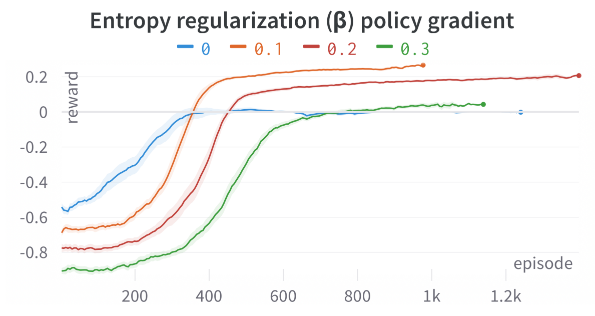

Figure 3. Entropy regularization policy gradient learning curve.

Again, we see that the optimum for this hyperparameter lies somewhere around 0.1, as it converges the fastest and achieves the highest mean reward while training. The policy gradient also supports this observation, as these curves resemble the reward curves.

2.2. Actor-Critic

2.2.1. Methodology

An actor-critic agent is very similar to a REINFORCE agent, in the sense that they both actively approximate an optimal policy through interaction with the environment using a policy gradient method. The main difference between the two is that the actor-critic agent uses an additional function approximator, called the critic, to estimate the value of the current state. The actor is used to select actions, and the critic is used to evaluate the actor's performance during training. A high-level overview of the actor-critic's training loop is described in Algorithm 2.

In order to assist the performance of the agent during training, we implemented two features; bootstrapping and baseline subtraction. Bootstrapping is a technique that allows the agent to learn from incomplete episodes, by using the critic's value function to estimate the value of the state at the end of the episode. A strategy for reducing the variation of the projected value of an action or state is baseline sub-

Algorithm 2 Actor-Critic

```
  **Input:** $env, \pi_\theta, V_\phi, \alpha_\pi, \alpha_V, \gamma$
  Initialize policy parameters $\theta$ and value function parameters $\phi$
  **repeat**
    Under policy $\pi_\theta$,
    sample episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$
    **for** $t = 0$ to $T - 1$ **do**
      $G_t \leftarrow \sum_{i=t+1}^{T} \gamma^{i-t-1} R_i$
      $\delta_t \leftarrow G_t - V_\phi(S_t)$
      $\phi \leftarrow \phi + \alpha_V \gamma^t \delta_t \nabla_\phi V_\phi(S_t)$
      $\theta \leftarrow \theta + \alpha_\pi \gamma^t \delta_t \nabla_\theta \log \pi_\theta(A_t|S_t)$
    **end for**
  **until** convergence
  **Output:** $\pi_\theta$
```

traction. This is done by subtracting the average value of the state from the projected value of the state.

2.2.2. Hyperparameter optimization

For our actor-critic agent, we used the same hyperparameters as our REINFORCE agent. While we would have wanted to experiment with using different learning rates for the actor and critic, we were unable to do so due to time constraints. For simplicity's sake, we performed nearly identical hyperparameter sweeps as we did for the REINFORCE agent, but with the addition of turning on and off the bootstrapping and baseline subtraction features.

2.2.3. Results

We compared the performances of the various agents after the bootstrapping and baseline subtraction features were implemented. The impact of bootstrapping on Actor-Critic performance is depicted in Figures 4 and 6. Figure 4 illustrates the variance of the various rewards received, where no discernible difference can be found. The average reward obtained is seen in Figure 6, where the bootstrapped agent does not always perform better. However, it does not necessarily perform worse in the long run. It appears to be more unstable than the non-bootstrapped agent.

Figures 5 and 7 show the effect of baseline subtraction on Actor-Critic performance. Figure 5 shows the variance of the different rewards gained, where some difference can be seen. The variance of the baseline subtracted agent is not lower than the variance of the non-baseline subtracted agent. It seems to even create some more variance into the system. Figure 7 also points this out, where the rewards are more stable, but not necessarily higher.

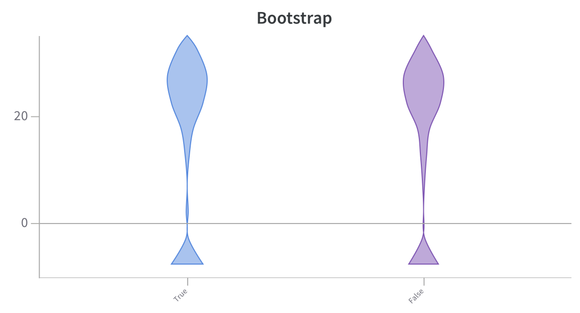

Figure 4. Effect of bootstrapping on Actor-Critic performance.

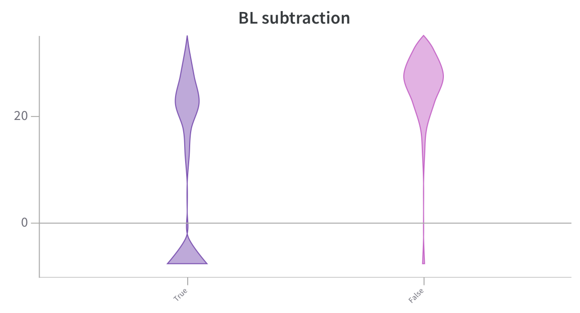

Figure 5. Effect of baseline subtraction on Actor-Critic performance.

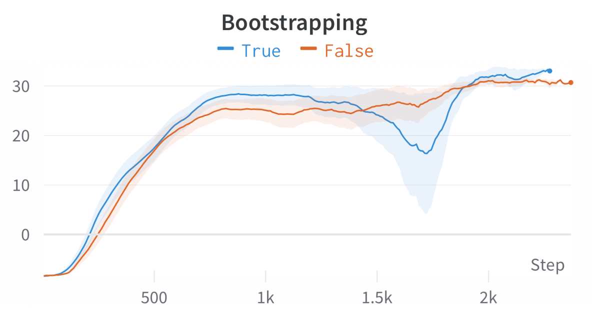

Figure 6. Effect of bootstrapping on Actor-Critic performance.

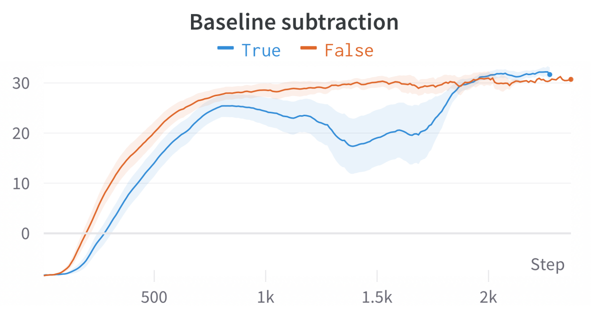

Figure 7. Effect of baseline subtraction on Actor-Critic performance.

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.

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: $^{\text{rows}}/_{\text{speed}}$, if the division is higher than 1, else it is 1. The max reward can be seen in Table 2.

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.

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

Table 2. 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.

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 3.

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

Table 3. 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.

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" left and right 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 rushed to 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 combine them together. Since the model for the narrow environment (14 x 7) was already trained, it was applied here as well. Table 4 displays the result.

The strategy of the agent was to remain in the middle, but upon enough balls falling on one side, a small 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.

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

Table 4. 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.

3.2. Vector

The vector agent was the second type we examined. Because of the vector's unique characteristic, only one model needs to be trained and can be applied to various environment modifications. This is feasible because the agent can see only the xy position of the lowest ball. It is therefor not affected by the amount of rows, or the speed of the balls. One issue with this approach is that the agent can not see the balls that are above the lowest ball. If the speed variable is larger than one, numerous balls will appear in the field, but only the lowest is taken into consideration. This would make learning a strategy, for catching two successive balls in different spots, extremely difficult for the agent. Depending on the speed and the rows, the second ball can already be halfway the playing field, unnoticeable for the agent.

3.2.1. Speed

The initial modification applied to the environment, similar to the pixel agent, is a change in speed. Again, the drop interval speed is determined by the change in speed as well as the number of rows. In Table 5 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 acted similar and tended 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 even 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. The agents are again expecting the next ball to fall in the same region, causing them to miss some balls. At a speed of 1.0, on which the agent was trained, it 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 slowly move to one wall.

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

Table 5. 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.

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 no new agent had to be trained, a speed of 1.0 was used to render the environment. Alike to the pixel calculation, the row changes were taken into account for the max reward. The results from this experiment are seen in Table 6.

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

Table 6. 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.

On the first environment (7 x 14) the agent performs relatively well for the size of the world. The environment is pretty wide, and not tall enough to reach the ball in time. Despite this, tje ahem managed to obtain a positive average reward of 8 over 100 episodes. It is constantly moving towards the next dropping ball, but sometimes the distance is to great to traverse in time. In 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 hovered 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.

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 7. 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. Sadly, sometimes shakes were observed when the agent was positioned under the ball.

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

Table 7. 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.

4. Conclusion

This study has conducted a comprehensive comparison of the performance of two policy-based reinforcement learning algorithms, namely REINFORCE and Actor-Critic. We investigated the impact of various hyper-parameters, the effect of bootstrapping, baseline subtraction, and entropy regularization, on the performance of both agents. In addition, we compared the performance of the pixel agent and the vector agent on different environments and speeds. From this research we have concluded the following key takeaways.

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, meaning that multiple sets of hyper-parameters were correct.

One parameter, 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 (that are too certain about their actions). The regularization hyper-parameter (β) showed to be of great importance to fine-tune. Leaving out this regularization altogether did not perform well at all; the variance was high between episodes when compared to values between 0.1 and 0.3. Only when the entropy regularization was set to 0.1, the agent started to perform better, but decreasing at higher values.

Alongside all the 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. After testing we found out that a bootstrap agent with n-step 3 seems to be too low to have a significant effect on the final results. The baseline subtraction, seemed to increase variance in rewards, where not using baseline subtraction concentrated the rewards upwards. This might be due to the implementation of a baseline subtraction less suitable for this environment. When combining both, we saw no direct difference in comparison to only 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, counteracting the variance allowing for faster and stable convergence.

The final comparison was that of the two Actor-Critic agents (pixel & vector) on different environments and speeds. To start of, the flexibility of the vector network, without suffering too much from information loss, worked great for this specific problem. It was able to traverse with relative ease and high rewards, through different environment size settings and ball drop intervals, when compared to the pixel agent. Speed changes were no problem for both, until it caused multiple balls to be in the playing field. Both agents were not trained to handle this, but the vector agent did adapt better to the situation than the pixel agent could. Wider 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. The “shake” behaviour of the vector agent was also observed in the pixel agent, but on a much smaller scale. This could be due to the fact that the entropy regularization was still letting the vector based agent explore the action space. The pixel agent did prefer to hug the side of the environment, which was not observed as much in the vector agent. Also, when performing in more extreme environments the agent started to hug the side of the environment.

4.2. Future Work

During the research, there were some ideas that we did not have the time to test and document properly. We would like to explore different speeds and environments, to see how the agents react to these changes. Does the agent need to train longer, or do we need to create a smaller environment to make it work?

Furthermore, the shaking behaviour seemed to stop at certain speeds, which is interesting to research further. The shaking behaviour might also be due to the fact that the agent is not trained for long enough, and that it will disappear after more training. We would also like to test the Actor-Critic agent with different n-steps, to see if this has a significant impact on the performance of the agent. During some late testing we noticed the effect of n-step (10) bootstrapping, but we did not have the time to test and document this properly.

We would also like to explore different baseline subtractions, since many implementations exist. We have seen that the baseline subtraction did not have a significant impact on the performance of the agent, but we would like to see if this is due to the implementation or the environment.

A last point for the future could be to further pinpoint the correct entropy regulation hyper-parameter. We have seen that the entropy regularization is of great importance, and would like to find out if we can find a better value than 0.1. Any value between 0 and 0.1 could theoretically perform better than values above 0.1, since we have seen that the performance decreases on higher values.

4.3. Contributions

Contribution	**Josef**	**Koen**
Code REINFORCE and Actor-Critic agent	V	V
Paper	V	V
CLI scripts & wandb integration	V	X

Table 8. This table shows the contribution dispersion of the project. Where each member has contributed fairly to the project.

References

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction (2nd ed.).* MIT Press, 2018.