Assignment 3: Policy-based Reinforcement Learning

Timo Vos, Louka Wijne, Chao Zhao

Abstract

In this assignment, we studied two deep policy-based reinforcement learning methods and their variations in the Catch environment. Detailed experiments have been conducted to tune their hyperparameters and to compare their performance in the environment. Finally, we conducted some experiments concerning the environment.

1. Introduction

In this section, we will give a short introduction to deep policy-based reinforcement learning and the environment that we use.

1.1. Reinforcement learning

Reinforcement learning is learning by doing and by making mistakes (Sutton & Barto, 2018). Reinforcement learning is also one of the three basic machine learning approaches (Sutton & Barto, 2018). It features an artificial agent and an environment. The agent follows some policy to determine its next action depending on the state that they are in. The goal is to find an optimal policy to get the agent to its goal. Every state has its own reward value, given by the environment. The environment is given by the transition model. This model is not known to the agent. Each state has a reward value. A trajectory is a sequence of states, actions and rewards from start to finish. The return is the cumulative reward of a trajectory. Two very important concepts are that of:

State value: $V(s_t)$ of a state s_t is the expected cumulative reward of a state by following the policy of that state.
State-Action value: $Q(s_t, a_t)$ of a state s_t combined with a certain action a_t is the expected cumulative reward of that state-action pair by executing the action from the state and following the policy further on.

1.2. Policy-based reinforcement learning

The field of classical reinforcement learning is that of tabular value-based reinforcement learning (Plaat, 2022). In tabular value-based reinforcement learning, an agent learns Q-values based on experience and it updates its knowledge using a table.

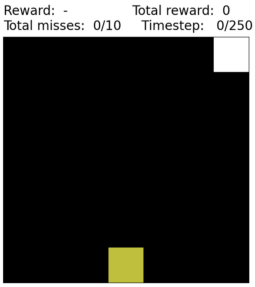


Figure 1. The Catch environment (Moerland, 2022). The paddle is always at the bottom, balls are dropped randomly from the top.

Deep value-based reinforcement learning. For high-dimensional problems with large state spaces, upholding a table filled with Q-values will not be realistic. That is when one needs to find an approximate solution using limited computational resources (Sutton & Barto, 2018). The agent learns how to act by generalizing from experiences in similar states. The key is to approximate the Q-values using function approximation in the form of deep learning. Deep reinforcement learning uses deep artificial neural networks. These networks reduce the number of neurons needed in comparison to more shallow networks (Liang & Srikant, 2016). The first deep reinforcement learning algorithm Deep Q-network (DQN) was introduced in 2013 by Mnih et al. (Mnih et al., 2013).

Policy-based reinforcement learning. Some reinforcement learning applications such as robotics have continuous action spaces as opposed to discrete action spaces (Plaat, 2022). In these action spaces, value-based reinforcement learning loses its meaning since there are an infinite amount of actions. Policy-based reinforcement learning does not use a separate value function but it models the policy directly. The policy function can be learned by policy gradient methods. They works well with deep neural networks by gradient ascent algorithm. And the policy function can also be learned by gradient free methods.

1.3. Environment

The Catch environment (Moerland, 2022) is extended from the Catch environment in Behavioural Suite (Osband et al., 2020). As shown in Figure 1, the agent needs to control a paddle to catch balls dropping from the top. The envi-

Table 1. Table of symbols and abbreviations

t	discrete time step
s_t	state at time t
a_t	action at time t
r_t	reward at time t
G_t	return at time t
τ	trajectory including states, actions, rewards
γ	discount rate
$\pi_\theta(a\|s)$	probability of taking a in s following policy π_θ
T	length of trajectory
V	state value function
Q	state-action value function
$\hat{Q}$	estimate state-action value function
$\hat{Q}_n$	n-step target computed by bootstrapping
A	advantage function
$J(\theta)$	performance measure of policy π_θ
$\nabla_\theta J(\theta)$	partial derivatives of $J(\theta)$ with respect to θ
$\nabla_\theta \pi_\theta(a\|s)$	partial derivatives of $\pi_\theta(a\|s)$ with respect to θ
θ	parameters of policy π
ϕ	parameters of critic V (state value function)
α	learning rate
ϵ	hyperparameter of clip-PPO
λ	strength of entropy regularization

ronment is customizable, meaning it can be made easier or more difficult. The width and the height of the state space can be adjusted. The observation space is either a vector with the locations of the paddle and the lowest ball or a binary two-channel pixel array, with the paddle location in the first channel, and all balls in the second channel. The paddle can move left, right or not at all at each timestep. All timesteps have a reward of 0, except for when a ball is caught $(+1)$ or when a ball is missed (-1). Balls randomly drop from the top, the speed of dropping can be adjusted. The task ends when the maximum number of misses or the maximum number of steps has been reached.

2. Methodology

In this section, we will introduce the methods we used to solve the problem. The symbols are listed in Table 1.

2.1. General methodology

The Catch environment is an episodic task. The goal is to catch as many balls as possible before the game terminates. Thus, we want to learn a policy to maximize the value of the initial state $V_{\pi_\theta}(s_0)$ because that is the expected number of balls caught by the paddle during the whole episode. Here, s_0 means the initial state of the current episode.

$V_{\pi_\theta}(s_0)$ can be used as a performance measure for different policies (Plaat, 2022; Sutton & Barto, 2018). We denote the performance measure as $J(\theta)$. Now we have:

$$J(\theta) = V_{\pi_\theta}(s_0) \tag{1}$$

Policy-based reinforcement learning aims to maximize $J(\theta)$. Therefore we can use the gradient ascent algorithm:

$$\theta_{t+1} = \theta_t + \alpha * \nabla_\theta J(\theta) \tag{2}$$

$\alpha \in \mathbb{R}^+$ is the learning rate. If we know the optimal action a^* in every state, it likes a supervised learning problem, the policy could directly learn towards the optimal actions:

$$\theta_{t+1} = \theta_t + \alpha * \nabla \pi_{\theta_t}(a^*|s) \tag{3}$$

The situation is that the optimal action a^* for state s is unknown. But it is possible to sample a trajectory and use estimates of the value of the state-action pair. the value of the state-action pair (s, a) is denoted as $\hat{Q}(s, a)$. And now we get:

$$\theta_{t+1} = \theta_t + \alpha * \hat{Q}(s, a) * \nabla \pi_{\theta_t}(a|s) \tag{4}$$

Note that based on the formula above, "good" actions are being improved double (i.e., good actions have higher $\hat{Q}(s, a)$ and higher probability $\pi_{\theta_t}(a|s)$), this is the root of instability. It can be corrected by dividing by the general probability:

$$\theta_{t+1} = \theta_t + \alpha * \hat{Q}(s, a) * \frac{\nabla \pi_{\theta_t}(a|s)}{\pi_{\theta_t}(a|s)} \tag{5}$$

Since we know of the identity $\nabla \ln x = \frac{\nabla x}{x}$, we can substitute it into Equation 5 and then we obtain:

$$\theta_{t+1} = \theta_t + \alpha * \hat{Q}(s, a) * \nabla_{\theta_t} \ln(\pi_{\theta_t}(a|s)) \tag{6}$$

The next step of obtaining an estimated Q network $\hat{Q}(s, a)$ and how to calculate the gradient $\nabla_\theta \ln(\pi_{\theta_t}(a|s))$ is the key difference among different policy-based methods.

2.2. REINFORCE

This algorithm optimizes the policy without bringing in Q-values (Plaat, 2022). REINFORCE learns directly from the trajectories obtained through sampling by following the policy π_{θ_t}.

Suppose we have a trajectory $\tau = \{s_0, a_0, r_0, s_1, ..., s_T\}$ sampled by following policy π_{θ_t}. T is the length of the trajectory, we can calculate the return at every timestep t:

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k \tag{7}$$

And we can get the estimated Q value at timestep t as:

$$\hat{Q}(s_t, a_t) = \gamma^t G_t \tag{8}$$

Substitute Equation 8 into Equation 6 and we get the update formula of the REINFORCE algorithm:

$$\theta = \theta + \alpha * \gamma^t G_t * \nabla_\theta \ln(\pi_{\theta_t}(a_t|s_t)) \tag{9}$$

Because REINFORCE estimates $Q(s, a)$ through sampling trajectory τ following π_θ, it is also called Monte Carlo Policy Gradient. The detailed pseudo-code can be found in Algorithm 1 (Williams, 1992; Plaat, 2022).

Algorithm 1 REINFORCE

Input: policy $\pi_\theta(a|s)$, learning rate $\alpha \in \mathbb{R}^+$, threshold $\epsilon \in \mathbb{R}^+$.
Initialize: θ;
while $\nabla_\theta J(\theta)$ does not converge below ϵ **do**
  Generate trajectory $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a|s)$;
  for $t \in \{0, \ldots, T-1\}$ **do**
    {For each timestep}
    $G_t \leftarrow \sum_{k=t}^{T-1} \gamma^{k-t} * r_k$;
    {Return from trajectory}
    $\theta \leftarrow \theta + \alpha * \gamma^t * G_t * \nabla_\theta \log \pi_\theta(a_t|s_t)$;
    {Update parameters}
  end for
end while
Return θ

2.3. Actor-Critic

This approach tries to improve upon the REINFORCE algorithm by combining the policy-based method with value-based elements (Plaat, 2022). Action selection in REINFORCE has low bias but high variance due to the sampling of full trajectories. The Actor-Critic method tries to achieve low bias and low variance by bootstrapping for better reward estimates and/or baseline subtraction for lower variance of gradient estimates. Both use the learned value function $V_\phi(s)$. It can use a separate neural network with parameters ϕ or a value head on top of the parameters θ. In the latter case the actor part (policy-based) and the critic part (value-based) share the lower layers of the network.

Bootstrapping: The return of a trajectory depends on many random choices. As a solution, we can bootstrap the value function. We use the value function to compute intermediate n-step values per trajectory. These are in-between full-trajectory Monte Carlo and single step temporal difference targets (Plaat, 2022):

$$\hat{Q}_n(s_t|a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n}) \tag{10}$$

Substituting Equation 10 into Equation 6, we get the update formula of the policy network of Actor-Critic:

$$\theta = \theta + \alpha * \hat{Q}_n(s_t|a_t) * \nabla_\theta \ln(\pi_{\theta_t}(a_t|s_t)) \tag{11}$$

For the critic network, we use the square loss to update it:

$$\phi = \phi - \alpha \nabla_\phi [\hat{Q}_n(s_t|a_t) - V_{\phi_t}(s_t)]^2 \tag{12}$$

Baseline subtraction. Subtracting a baseline reduces variance but does not affect the expectation (Plaat, 2022). We can use the value function as baseline. This gives us the advantage function:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{13}$$

This can be combined with bootstrapping to estimate the cumulative reward $\hat{Q}(s_t, a_t)$:

$$\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t) \tag{14}$$

Substituting Equation 14 into Equation 6, we get the update formula of the policy network:

$$\theta = \theta + \alpha * \hat{A}_n(s_t, a_t) * \nabla_\theta \ln(\pi_{\theta_t}(a_t|s_t)) \tag{15}$$

The update formula of the critic network can be re-written as:

$$\phi = \phi - \alpha \nabla_\phi [\hat{A}_n(s_t, a_t)]^2 \tag{16}$$

The pseudo-code of Actor Critic is shown in Algorithm 2.

Algorithm 2 Actor-Critic

Input: policy $\pi_\theta(a|s)$, value function $V_\phi(s)$, estimation depth n, learning rate $\alpha \in \mathbb{R}^+$
Initialize: θ and ϕ;
while $\nabla_\theta J(\theta)$ does not converge below ϵ **do**
  Generate trajectory $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a|s)$;
  for $t \in \{0, \ldots, T-1\}$ **do**
    if bootstrapping **then**
      $\hat{Q}(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$;
    else
      $\hat{Q}(s_t, a_t) = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$;
    end if
    $\hat{A}(s_t, a_t) = \hat{Q}(s_t, a_t) - V_\phi(s_t)$;
    {Baseline subtraction}
    $\phi \leftarrow \phi - \alpha * \nabla_\phi [\hat{A}(s_t, a_t)]^2$;
    {Descent the critic loss}
    if baseline subtraction **then**
      $\theta \leftarrow \theta + \alpha * [\hat{A}(s_t, a_t) * \nabla_\theta \log \pi_\theta(a_t|s_t)]$;
    else
      $\theta \leftarrow \theta + \alpha * [\hat{Q}(s_t, a_t) * \nabla_\theta \log \pi_\theta(a_t|s_t)]$;
    end if
    {Ascent the policy gradient}
  end for
end while
Return θ

2.4. Clip-PPO

Proximal Policy Optimization (PPO) methods are concerned with improving as fast as possible in one timestep using the current data without collapsing (Schulman et al., 2017). It uses trust regions to achieve adaptive step sizes in non-linear spaces (Plaat, 2022). Clip-PPO clips in the objective function to avoid the new policy getting too far away from the old one.

In the update formulas in Equation 9, 11, 15, in the REINFORCE algorithm or the Actor-Critic algorithms, the log probability of actions is used: $\nabla_\theta \log \pi_\theta(a_t|s_t)$. It can be

replaced with other items. For example, TRPO uses the the following formula to replace $\nabla_\theta \log \pi_\theta(a_t|s_t)$ (Schulman et al., 2015):

$$\nabla_\theta \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{17}$$

The problem here is that really big gradient steps are sometimes taken when an action is much more probable for the current policy. Clip-PPO stabilizes by clipping the objective function (Schulman et al., 2017):

$$\nabla_\theta \min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, clip(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)) \tag{18}$$

where ϵ is a parameter used to clip. The authors state that approximately 0.2 is a good value for ϵ. The clipping prevents the algorithm from getting greedy. So if the action was good and it became more probable the last time a gradient step was taken, we do not keep updating it too far. If it became less probable, we can undo that step. If the action was bad and it became less probable the last time a gradient step was taken, we do not keep updating it too far. If it became more probable, we can undo that step. Clip-PPO trains a stochastic policy in an on-policy way. It samples actions according to the current version of the policy.

We can combine Clip-PPO with REINFORCE or Actor-Critic by replacing the item $\nabla_\theta \ln \pi_\theta(a_t|s_t)$ in Equation 9, 11 and 15 with Equation 18.

2.5. Entropy regularization

Both REINFORCE and Actor-Critic are on-policy reinforcement learning algorithms. They learn a policy $\pi_\theta(a|s)$, which is a distribution over possible actions. But it can easily cause the model to fall into local optimum. That is to say, the probability of a certain action can rise to 1, and the probability of other actions can fall to 0. Thus, we need a method to encourage the model to explore.

For all of these variations, we use entropy regularization as exploration policy. Entropy is the lack of predictability of actions. It is calculated as $H(\pi_\theta(\cdot|s_t)) = -\sum_{a\in\mathcal{A}} p_a \log p_a$, where $\mathcal{A}$ is the set of possible actions following policy π_θ at state s_t, and where p_a is the probability of choosing action a at state s_t. We add a penalty to the loss function, which makes sure the entropy H of the probability distribution for the actions stays larger. Considering the general update formula Equation 6, we have the new update formula by adding entropy regularization (Plaat, 2022):

$$\theta_{t+1} = \theta_t + \alpha[\hat{Q}(s,a)\nabla_{\theta_t} \ln(\pi_{\theta_t}(a|s)) + \lambda \nabla_{\theta_t} H(\pi_\theta(\cdot|s))] \tag{19}$$

where $\lambda \in \mathbb{R}^+$ determines the strength of entropy regularization. Through Equation 19, we can get the new policy update formula by replacing different items of REINFORCE or Actor-Critic. For example, we replace $\hat{Q}(s,a)$ in Equation 19 with Equation 8, we can get the update formula with entropy regularization.

3. Experiments

In this section, we will experiment with the algorithms introduced in Section 2 and the environment as discussed in Section 1.3. For the implementation of the algorithms discussed, we use the open-source deep learning framework *PyTorch* (Paszke et al., 2019). The size of the environment in this section is 7×7, speed of the ball is 1.0, observation type is pixel, maximum steps is 250, and the maximum number of misses is 10.

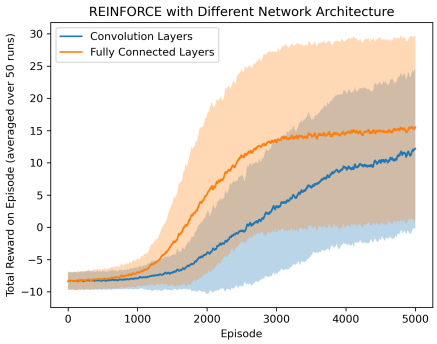
3.1. REINFORCE

In this subsection, we will explore the impact of parameter values on REINFORCE. In addition to the parameters to be explored, the default values of other parameters are as follows: learning rate $\alpha = 0.001$, discount rate $\gamma = 0.99$, regularization strength $\lambda = 0.01$. The policy network has one hidden layer with dimension 128 and the activation function that is used is *SiLU*. Weights of the network are initialized by the Xavier Uniform method.

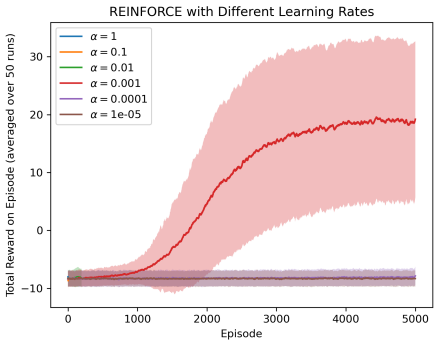
Network Architecture: Since the state s can be regarded as a 7×7 size picture with 2 channels, intuitively, we guess that the policy network of the convolutional neural network (CNN) structure may be helpful for agent learning. So we implemented a policy network with two convolution layers and one fully connected layer. The number of output channels of the first convolution layer is 32, kernel size is 4×4, stride is 1. The number of output channels of the second convolution layer is 64, kernel size is 4×4, stride is 2. And then we flatten the output and feed it into the fully connected layer to get the possibilities of possible actions.

We compare the performance of the network with CNN architecture and the network with the default fully connected architecture. The results are shown in Figure 2 (a). After 5000 Episodes, CNN failed to exceed the default fully connected network, but it was still on the rise, while the default network has converged at the end. The reason is that CNN has more parameters than the default fully connected network, resulting in that CNN is harder and slower to learn. Therefore, we can only say that within a budget of 5000 episodes, CNNs do not perform as well as the default fully connected networks.

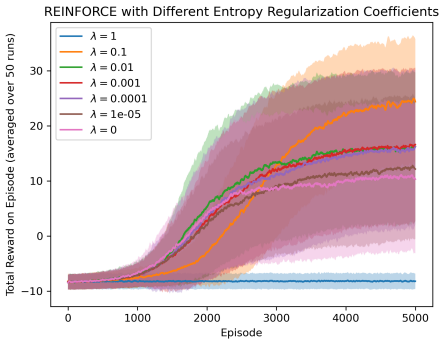
Learning Rate: We have experimented with different learning rates. The results can be found in Figure 2 (b). The best choice according to the graph is a learning rate α of 0.001. If the learning rate is too big, it is easy to cause the gradients to vanish or explode, as shown in the blue or green curves ($\alpha = 0.1$ or 0.01) in Figure 2 (b), and they stopped



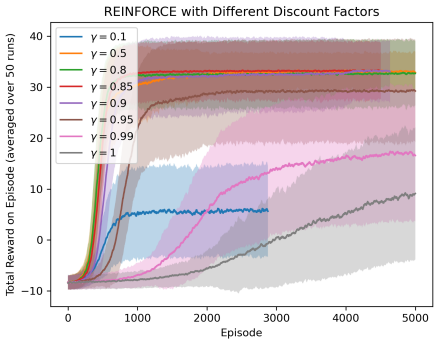
(a) Effect of the network architecture on learning performance of REINFORCE in the Catch environment. The graph shows learning curves for two network architectures. Optimal performance is achieved with fully connected layers.



(b) Effect of the learning rate α on learning performance of REINFORCE in the Catch environment. The graph shows learning curves for six values of α. Optimal performance is achieved for the learning rate of 0.001.



(c) Effect of the entropy regularization coefficient λ on learning performance of REINFORCE in the Catch environment. The graph shows learning curves for seven values of λ. Optimal performance is achieved for the entropy regularization coefficient of 0.1.



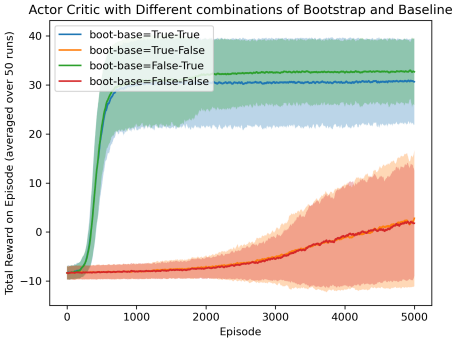
(d) Effect of the discount rate γ on learning performance of REINFORCE in the Catch environment. The graph shows learning curves for eight values of γ. Optimal performance is achieved for a discount rate between 0.5 and 0.9.

Figure 2. Experiments depicting the average reward per episode for REINFORCE. Note that some curves terminate after less than 5000 episodes. This is because the gradients are vanishing or exploding, which prevents the model from outputting correct results. All results are averaged over 50 repetitions. Shaded areas indicate 1 standard deviation confidence intervals of the mean. (a) Explores whether CNN can improve the performance of REINFORCE. (b) Finds the best learning rate for REINFORCE. (c) Explores the impact of the entropy regularization strength. (d) Finds the best discount factor for REINFORCE.

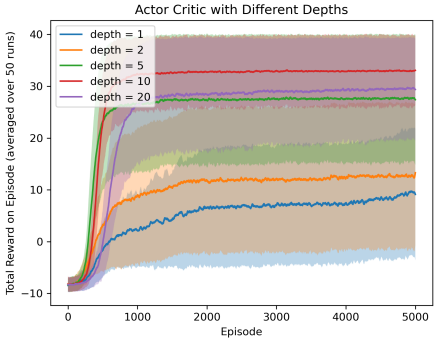
training after less than 200 episodes because of gradients vanishing or exploding. A learning rate that is too small results in too slow learning, such as the purple and brown curves ($\alpha = 0.0001$ or 10^{-5}), they did not learn anything. The learning rate of the order of 0.001 is the best learning rate in this case.

Entropy Regularization Coefficient: We have experimented with different entropy regularization coefficients. The results can be found in Figure 2 (c). The stronger entropy regularization gets (the larger λ is), the less predictable the choice of action is. This means that the exploration is stronger. A λ value of 1 does not perform very well, as to be expected. A value of 0 performs better, but $\lambda = 0.1$ outperforms all other values. This might be due to the fact that when $\lambda = 0$ or too close to 0, the model can fall into a local optimum while when λ gets too large, the actions are chosen too randomly.

Discount Factor: A discount factor γ between 0.5 and 0.9 seems to perform optimally, according to the results of the experiment found in Figure 2 (d). The discount factor γ determines how much value is placed upon future rewards. The higher the discount rate, the higher the future rewards are valued. This makes sure the agent is not too greedy and only focusing on immediate rewards. If γ is too large, however, immediate rewards that are important might be missed. The graph suggests that considering future rewards pays off, but we should focus slightly more on rewards that are closer since they represent the balls that are getting caught in the near future as opposed to those that are still far away.



(a) Effect of bootstrapping and baseline subtraction on learning performance of Actor-Critic in the Catch environment. The graph shows learning curves for four variation combinations. Optimal performance is achieved with baseline subtraction without bootstrapping.



(b) Effect of the search depth on learning performance of Actor-Critic in the Catch environment. The graph shows learning curves for seven values of the depth. Optimal performance is achieved for a depth between of 10.

Figure 3. Experiments depicting the average reward per episode for Actor-Critic. Results are averaged over 50 repetitions. Shaded areas indicate 1 standard deviation confidence intervals of the mean. (a) Explores whether bootstrapping and/or baseline subtraction can improve the performance of Actor-Critic. (b) Finds the best search depth for Actor-Critic.

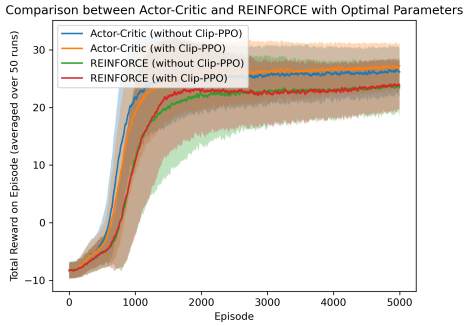


Figure 4. Actor-Critic or REINFORCE with optimal values of hyper-parameters and with or without Clip-PPO. Results are averaged over 50 repetitions. Shaded areas indicate 1 standard deviation confidence intervals of the mean.

3.2. Actor-Critic

In this subsection, we will explore the impact of parameter values on Actor-Critic. In addition to the parameters to be explored, the default values of other parameters are as follows: learning rate $\alpha = 0.001$, discount rate $\gamma = 0.99$, regularization strength $\lambda = 0.01$. The policy network has one hidden layer with dimension 128 and the activation function that is used is *SiLU*. Weights of the network are initialized by the Xavier Uniform method. And Bootstrapping and baseline subtraction are used by default, and the depth for bootstrapping is 1.

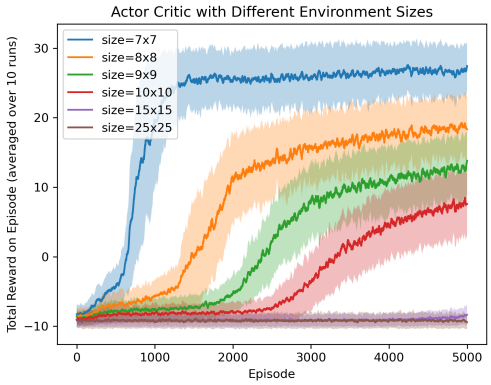
Learning Rate: We have experimented with different learning rates. Due to space limitations, plot is not shown here. The conclusion is that: the best choice according to the graph is a learning rate α of 0.001. The same reasoning as with the experiment for REINFORCE holds here too.

Entropy Regularization Coefficients: We have experimented with different entropy regularization coefficients. Due to space limitations, plot is not shown here. The conclusion is that: a λ value of 0.1 outperforms all other values. The same reasoning as with the experiment for REINFORCE holds here too.

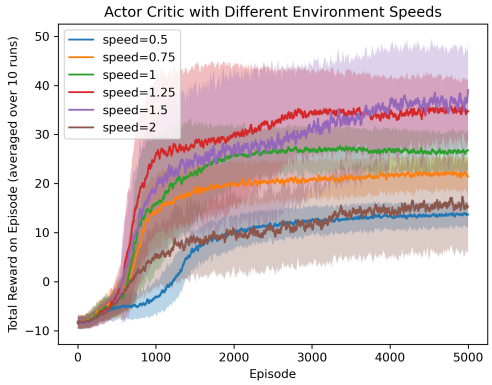
Discount factor: Due to space limitations, plot is not shown here. The conclusion is that: a discount factor γ between 0.5 and 0.85 seems to perform optimally. The same reasoning as with the experiment for REINFORCE holds here too.

Bootstrapping and baseline subtraction: As explained in Section 2, the Actor-Critic algorithm can be used with bootstrapping and/or baseline subtraction. We have experimented with multiple combinations of the two. At this time, we use search depth for 10 of bootstrapping. The results can be found in Figure 3 (a). Interestingly, bootstrapping without baseline subtraction does not seem to add a lot of value. Baseline subtraction does significantly improve the performance of Actor-Critic. In fact, bootstrapping seemingly does not improve much upon performance when already using baseline subtraction.

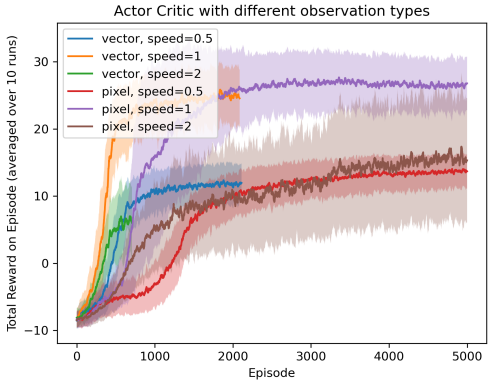
Search Depth: We have experimented with different depths of bootstrapping as well. The results can be found in Figure 3 (b). A depth of 1 or 2 does not perform very well, but neither does a depth that is too high. A depth of 10 seems to be optimal. The reason that a higher estimation depth does not perform better may be that the n-step targets start to resemble full-trajectory Monte Carlo, leading to high variance. This is exactly what we want to avoid.



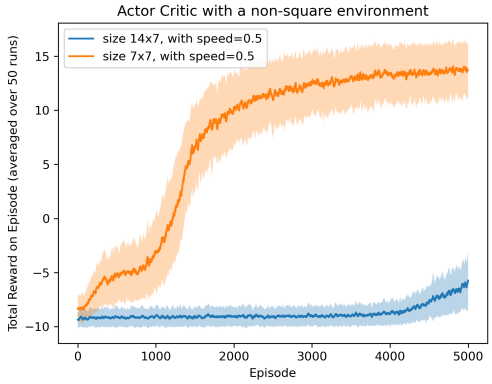
(a) Effect of the board size on learning performance of Actor-Critic in the Catch environment. The graph shows learning curves for six board sizes. Optimal performance is achieved with the smallest board of 7×7.



(b) Effect of the falling speed on learning performance of Actor-Critic in the Catch environment. The graph shows learning curves for six falling speeds. Optimal performance is achieved with a falling speed of 1.5.



(c) Effect of the observation type on learning performance of Actor-Critic in the Catch environment. The graph shows learning curves for six observation types. Optimal performance is achieved with the observation type "pixel", speed is 1.



(d) Effect of a non-square board on learning performance of Actor-Critic in the Catch environment. The graph shows learning curves for one square and one non-square board. Optimal performance is achieved with a square board of 7×7.

Figure 5. Experiments depicting the average reward per episode for Actor-Critic and different environment configuration. Note that some curves terminate after less than 5000 episodes. This is because the gradients are vanishing or exploding, which prevents the model from outputting correct results. Results are averaged over 10 repetitions. Shaded areas indicate 1 standard deviation confidence intervals of the mean. (a) Explores the effect of the board size. (b) Explores the effect of the speed of the apple. (c) Explores the effect of the observation type. (d) Explores the effect of non-square boards.

3.3. Clip-PPO and Models Comparison

Through the experiments, we find the optimal value of the hyper-parameters for both Actor-Critic and REINFORCE: 0.0001 of learning rate α, 0.1 of regularization strength λ, 0.5 of discount rate γ. And for Actor-Critic, it should adopt bootstrapping with step 10 and baseline subtraction. And in this subsection, we compared the performance of Actor-Critic and REINFORCE with the optimal values of hyper-parameters. And with or without Clip-PPO to validate that whether Clip-PPO is effective.

As shown in Figure 4, we can see that Clip-PPO almost has no impact on our problem. And Actor-Critic is better than REINFORCE. So we will use Actor-Critic with Clip-PPO for the environment experiments.

3.4. Environment

After doing the different experiments on the agent it was possible to find the best agent for the environment used in this report. The goal of this section is to experiment with this best agent in variations of the environment. All the experiments are done using the best agent configuration we discussed in Section 3.3 and the baseline configuration of the environment except if mentioned otherwise.

Board Size: The best agent had proven to work well on quite a small board size of 7×7. Figure 5 (a) shows the

performance of the agent for different sizes of the board. Increasing the board size increases the complexity of the environment, there are more states to learn. This is also represented in the results with an ever-lower mean reward for larger boards. The agent does learn so increasing the amount of times the agent can train will likely overcome this problem.

Speed of the Ball: Increasing the speed of the falling ball allows the agent less time to find the right position under the ball, but it also decreases the drop interval which means the agent can collect more balls. Decreasing the speed however allows the agent more time to find the right location for the ball, but also reduces the amount of ball dropped. This trade-off between less time to find the right location and the drop interval can be seen in Figure 5 (b). The best speed seems to be around 1.25 - 1.5, the agent can learn to catch the balls fast enough with this speed which leads to a higher reward, increasing more will not allow the agent enough time to catch the balls.

Observation Type: In the environment, it's possible to change the perceived input of the agent, from pixels to vectors. In Figure 5 (c), it can be seen that the performance of the vector compared to the pixel input results in a higher total reward. This can be explained by the complexity of the inputs, the vector is only of size 3. The pixel input however is of size $width \times height \times 2$, this input is way larger which results in a more complex input. A higher complexity would need more time to compute. **Why is it not recommended to use observation type "vector" with speed > 1.0?** because when using speeds greater than 1 there might be multiple balls on the board which are not all observable by the agent. Only the lowest ball will be known to the agent. And in Figure 5 (c) some of the runs stopped early due to vanishing/exploding gradients.

Non-square Board In the last experiment (results can be found in Figure 5 (d)), the board size was increased in width without increasing the height. The agent can't catch the ball initially, and then it can catch balls a little bit as we can see the blue curve in Figure 5 (d). Compared with the 7×7 environment, 14×7 is more difficult to learn, because the height is only 7, and the width is 14. When the distance between the agent and the column where the ball falls exceeds 7, there is no possibility of catching the ball at all. But the blue curve has an upward trend. If it has more training time, the agent may be able to catch more balls.

4. Discussion

In this assignment we implemented the REINFORCE algorithm and the Actor-Critic algorithm using *PyTorch* (Paszke et al., 2019). The Catch environment was used to do experiment (Moerland, 2022). We tuned hyper-parameters of REINFORCE and Actor-Critic and its variations. We found that the Actor-Critic algorithm with bootstrapping and baseline subtraction outperforms REINFORCE. We performed some experiments by adjusting the environment and found that smaller playing fields are easier to master, a higher falling speed might mean more rewards to some degree, a vector representation is less complex but also less complete and non-square boards can be hard to navigate.

Contribution: Timo was responsible for implementing REINFORCE, Chao was responsible for implementing Actor-Critic, and all of us co-designed the plan of the experiment part. We completed the report together while Louka made a huge effort on it.

References

Liang, S. and Srikant, R. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. 2013.

Moerland, T. Customizable catch environment, 2022.

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, C., Singh, S., Van Roy, B., Sutton, R., Silver, D., and van Hasselt, H. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygf-kSYwH.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Plaat, A. *Deep Reinforcement Learning*. Springer, 2022.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.