Deep Reinforcement Learning - assignment part 2

Thijs Lukkien [1] **Carlos Brito** [1]

Abstract

Reinforcement learning is a subfield of machine learning where the goal is to optimize a given policy to maximize the rewards returned over time. The goal of this report is to implement the PPO algorithm in the Catch environment previously provided to us. Then we will analyze the results and provide a conclusion of the performance of the algorithm.

1. Introduction

In this report, we will first describe the environment that was used to train and test our RL algorithms. Then we will provide a description of the PPO-clip algorithm that as implemented. Followed by describing the methods that we used to train and test out models. Additionally, we will provide the results that we obtained and discuss them. Finally, we will provide a conclusion to the results we obtained. Online repositories and fora have been helpful in implementing the algorithm. Especially

- Include all websites and github repos (especially mention the github from blobmaster as source of inspiration (https://github.com/zplizzi/pytorch-ppo/blob/master/train.py))

2. Environment

The environment that is used for this work is the *Catch* environment (Mnih et al., 2014). In the game of catch an agent performs certain actions with a paddle to catch a ball that is falling vertically. This is performed within an environment that is represented with a grayscale 21×21 grid, where the paddle is located on the bottom half of the grid. In this grid, the ball is falling with a vertical speed of $V_y = -1 \quad cell/s$ and has a vertical speed of $V_x \in \{-2, -1, 0, 1, 2\}$. The paddle may perform three possible actions: left, right, or no action. This environment has been implemented using the *Gymnasium* environment (Towers et al., 2024) as a base. Gymnasium is an open-source Python library that is widely used to study reinforcement learning algorithms. For this assignment, the grid has been scaled up by a factor of 4, resulting in a 84×84 grid. Also, frame stacking has been applied such that each observation contains 4 frames. This results in the input observation shape of $84 \times 84 \times 4$.

Figure 1. Representation of the game of Catch

3. Networks

In this section, we will describe the architectures that have been implemented within the code.

The main model architecture that we have used is the following:

- Convolution layer with: 4 input channels, 32 output channels, kernel size of (8,8) and a stride of 4
- ReLU activation
- Convolution layer with: 32 input channels, 64 output channels, kernel size of (4,4) and a stride of 2
- ReLU activation
- Convolution layer with: 64 input channels, 64 output channels, kernel size of (3,3) and a stride of 1
- ReLU activation
- Linear layer with: 3136 input channels, 512 output channels
- ReLU activation

After using the main model architecture we used two different networks for the heads, for the actor and the critic. The actor's head is as follows:

- Linear layer with: 512 input channels and 3 input channels

Finally the critic's head is as follows:

- Linear layer with: 512 input channels and 1 input channels

Both these heads were initialized with a semi-orthogonal matrix. However, the actor's head was initialized using a gain of 0.01 and the critic's head used a gain of 1.

The optimizer that was used was Adam with a learning rate of 1e-3 and an epsilon of 1e-5. Furthermore, in order to combine the losses of the actor, critic, and entropy we used the ittertools library to chain the parameters of each network and passed this to the Adam optimizer.

The hyperparameters that were used are the following:

- Clip value: ϵ = 0.2
- GAE Lambda = 0.95
- Gamma: γ = 0.99

The Generalized Advantage Estimation ($A_t^{GAE(\gamma,\lambda)}$) combines multiple steps of TD errors to reduce the variance. It introduces a parameter $0 \leq \lambda \leq 1$ that as the lambda is increased to 1, we can reduce the variance of the estimator but increase the bias. The equation is show in eq 1.

$$A_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{1}$$

3.1. PPO-clip

The PPO algorithm was developed as an improvement of TRPO. This algorithm maintains some of the benefits of TRPO while also being simpler to implement and more general. Furthermore, the sample complexity and the runtime complexity have been improved.

This algorithm simplifies the objective function by using a clip function. Also, this penalizes too large policy updates.

Equation 2 is the ratio of the of the probabilities.

$$r(\theta) \equiv \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} \tag{2}$$

The actor loss that we used is shown in equation 3.

$$J^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a)\right)\right] \tag{3}$$

The critic loss that was used is the MSE as shown in equation 4.

$$\mathcal{L}_t^{\text{VF}}(\theta) = \left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2 \tag{4}$$

The entropy regularization term that is used is shown in equation 5. This term is used to incentivize even more exploration. This was taken from the entropy of the actor probabilities.

$$\beta = c_2 S\left[\pi_\theta\right](s_t) \tag{5}$$

The final objective loss is shown in equation 6.

$$\mathcal{L}_t(\theta) = \left[J^{\text{CLIP}}(\theta) - c_1\mathcal{L}_t^{\text{VF}}(\theta) + c_2 S\left[\pi_\theta\right](s_t)\right] \tag{6}$$

The constant values that were used were : c_1 = 0.5 and c_2 = 0.01.

For our implementation, we have implemented a function where we obtain the logits of the observation. This is done by using the base architecture and the actor's head. Then we create a categorical distribution using these logits and then this is sampled to obtain an action. We also calculate the log probability using the action chosen, and the entropy of the probabilities, and finally we also calculate the critic value.

4. Methods

In order to test the implementation of the algorithm within the game of Catch we used the following experimental setup:

We have set the number of episodes that each run will have to 60000. Then every 200 episodes we take a copy of our base model and our actor head to test this model for 10 runs of the game of Catch. After running this test we averaged the rewards we obtained and then saved these results in a .pkl file. This method resulted in 600 average rewards of this test model. In the figure, we can see a solid line that represents the average value of the reward. Around this, we can see the standard deviation represented as a cloud around this line.

The code that was developed proved to be computationally intensive and for this reason, it was run within the Habrok HPC cluster.

5. Results

In this section, we will present the results that we have obtained after following the method presented previously.

The accuracies that we present in figure 2 correspond to the average reward per test run that was done every 200 episodes and did 10 runs of the catch environment.

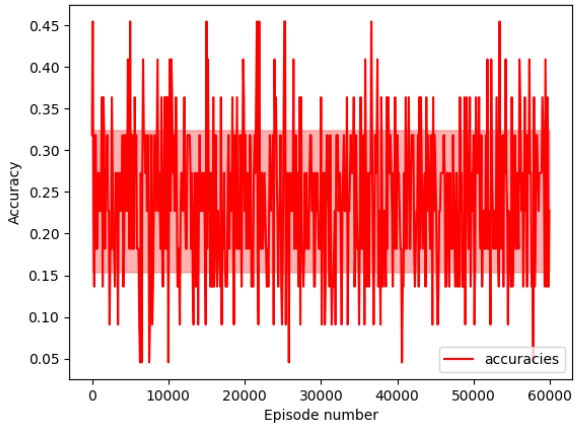

Figure 2. Accuracies obtained from running the PPO algorithm in 60000 episodes and testing every 200 episodes for 10 runs of the catch environment averaged out

6. Discussion

From figure 2 we clearly see that the model does not seem to learn and presents an erratic performance. However, with the loss that was observed, we could see that the network was learning as it was converging to zero. Due to this issue, we analyzed the models to determine where the issue could have originated from. We checked that all the parameters of the segments of the models were being updated. After confirming this, we started to observe the individual losses of the actor, critic, and the entropy factor. From these values, we saw that while the entropy converged, this caused either the actor or critic losses to also converge, however not at the same time. This means that either the entropy and the actor loss converged or the entropy and the critic loss converged. This indicated that the problem was either with the critic or the actor loss that we implemented. Nevertheless, this could also mean that the entropy was wrongly tuned so that there was not enough exploration.

Due to the issues with the results, we also decided to observe the logits that we obtained when choosing an action and we could see that the network would always favor one action above the other. With this, we wanted to experiment further with the entropy and tune its value. Furthermore, we also analyzed the way we obtained the GAE since this could also affect the actor and critic optimization. Finally, due to the chaining of the parameters in the optimization function there may be a strange interaction happening here when updating the base model and the head's parameters.

7. Conclusion

In conclusion, we implemented the PPO algorithm using the Catch environment. Then we evaluated the model by running the model for 60000 episodes. After every 200 episodes we copied the base model and the actor in order to test them in 10 runs of the Catch environment. These values produced 600 averaged rewards, these were used to produce the figure that was presented. We can see that the model that was developed was not able to properly learn. This could have been for a variety of reasons such as the parameters of the model not being correctly updating, the optimization function using the parameters of three models, and others.

References

Mnih, V., Heess, N., Graves, A., and Kavukcuoglu, K. Recurrent models of visual attention, 2014.

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, A. J. S., and Younis, O. G. Gymnasium, February 2024. URL https://doi.org/10.5281/zenodo.10655021.