CSE510 Intro to Reinforcement Learning
Final Course Project

Utkarsh Behre, Nikhil D Kamath
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
ubehre@buffalo.edu, ndkamath@buffalo.edu

Abstract

In this project, we were asked to explore and research on the different deep reinforcement learning algorithms. We were given options of environments to choose from. The focus of our research was more on the algorithms and their behavior on different environments. After doing our research and project work, herein lies our final report.

1 Introduction

In this project, we have implemented various deep reinforcement learning algorithms to see their behavior on different environments. The goal was to see the results and compare them for each algorithm on different environments. We have implemented 5 different deep reinforcement learning algorithms: DQN, Double DQN, Advantage Actor Critic, Reinforce, and PPO. We coded different versions of the algorithms with no convolution layers for non-image-based environments and other versions with convolution layers being 2,3, and 4. However due to computation limitations we will focus more on non-image-based environments with minor discussions on image-based environments.

2 Motivation

We have been implementing different algorithms throughout the course so far in the course. However, when we implemented those, we were new to them and were unaware of some of the algorithms introduced later on. So, the motivation for this project is to compare up the algorithms we learnt so far along with a new one we learnt recently.

We wanted to see which algorithm stands out amongst the other, or say which one is the best. The aim was to check if some algorithm can do better on 1 environment but do worst on other environment when compared to other algorithms. This would help in understanding how the algorithms work better and ultimately would help in choosing the appropriate algorithm given a new

environment or situation.

2 Algorithms

We have used 5 different algorithms: DQN, Double DQN, Advantage Actor Critic, Reinforce, and PPO. We'll first go over each algorithm and see how they work. Then we will go over how we have implemented them for our testing.

2.1 DQN and Double DQN

Deep Q Networks makes use of deep neural networks along with the idea of Q learning. For this assignment we have implemented the vanilla DQN and also have the modified version of the DQN, specifically used double DQN algorithm. Both of these algorithms are implemented in the same class in the code as the differences between implementation of these 2 algorithms is very minor and only requires a different updating model technique.

We use experience replay in DQN. We store all the transitions that happen between states given an action, which we sample and use to train the model. This way the agent doesn't use sequence of steps taken on the states, but rather uses these different transitions which have random states with actions taken with the rewards. This way the agent can generalize what it's trying to learn.

DQN uses 2 different models while training. These are evaluation and target networks. The main purpose of using a target network and not just going with a single model is to make sure that the agent has a stable model that it can use while predicting for future rewards. If it uses the same model which its training to predict the future rewards for the next state as well, it would just be too unstable overall. This might work for really simple environments but it will not do well in more complex environments. Target Network helps us to get the rewards of even the next state which that helps punishing the bad Q values even further.

2.2 A2C

There are different versions of Actor-Critic algorithms out there. For this assignment I have chosen the advantage actor-critic algorithm also known as A2C. The algorithm as the name suggests involves an actor and a critic. The actor makes use the policy network to decide which actions to take at any given state. Then the critic makes use of the value functions to help the actor in learning.

So, actor-critic algorithm generates state-value functions, which is given by the critic, and policy, which is given by the actor. Every time when training

happens, it has to happen for both the actor and the critic networks.

Even though both networks may look alike there is a huge difference In the output of each network. The actor network outputs the softmax output on available actions which basically gives probabilities on different actions. But for the critic network the output is a single value. And since the output dimensions serve different purpose for both of these networks we use softmax activation for the actor network while we use linear activation for the critic network. This is why we use categorical_crossentropy as loss function for our actor network while we use mean_squared_error as loss function for the critic network.

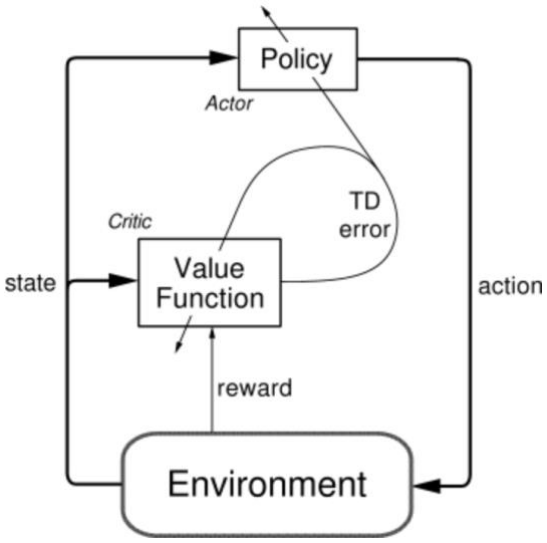

Figure 1: The actor-critic architecture

2.3 REINFORCE

REINFORCE otherwise also known as Monte-Carlo policy gradient uses concepts from Monte-Carlo in its implementation. Just like Monte Carlo, REINFORCE collects the states, rewards, actions taken throughout during an episode. It then uses the rewards to find out the gradient which is basically the returned overall discounted rewards referred to as G_t that we got throughout the episode. The algorithm is pretty straight forward as in below figure taken from the Reinforcement Learning book.

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$

The algorithm involves a network that can have multiple hidden layers that is chosen based on the complexity of the environment. In the network we feed the environment states we got throughout the episode as input, we pass the G_t as the sample weight for the batch and the output of the network is probabilities of different actions to be taken for which a softmax activation function is used.

2.4 PPO

PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

  by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

end for

3 Implementation

We have implemented different versions of all the algorithms. Since the environments can be image-based or non-image-based, we have coded a simple non-image-based version of all the algorithms. This version does not have any convolution layers as the input is different number of parameters. Apart from this we have made 3 different versions for image-based each. These are named based on the number of convolution layers it has in the model. We have 2c, 3c, and 5c

which have 2,3, and 5 convolution layers respectively. So, in the files we have 0c for non-image-based versions, and, then we have 2c, 3c, and 5c for image-based versions.

We have separate notebook files for different environments. We tried to train the models on Space Invaders, Assault, DemonAttack that are image-based. Even though we got the upwards trending graph for each of the algorithms. It was hard to differentiate between different algorithms as they take a lot of time to run. The only thing we noticed was that the models with 3 convolutional layers performed well enough with 2 cl being very similar and 5c being slightly better.

So, we trained our models on the classic CartPole and also Acrobot environments. We did this for all 5 algorithms to see the comparisons. We have a notebook for each environment where we ran all algorithms and plotted graphs for them. The exception being PPO for which we created a separate notebook since it is a new algorithm we tried.

5 Results

We train all 5 algorithms on CartPole-v1 and Acrobot-v1 to see how well they perform and how they make the progress. Both environments had a different learning curve overall which we can observe from all the graphs. First, we'll see the rewards vs episodes graphs for each of the algorithms on both environment then we'll compare them all together environment wise.

5.1 DQN results

In case of DQN, we notice that the agent is able to learn the CartPole environment really well and acrobot environment as well to some limits.

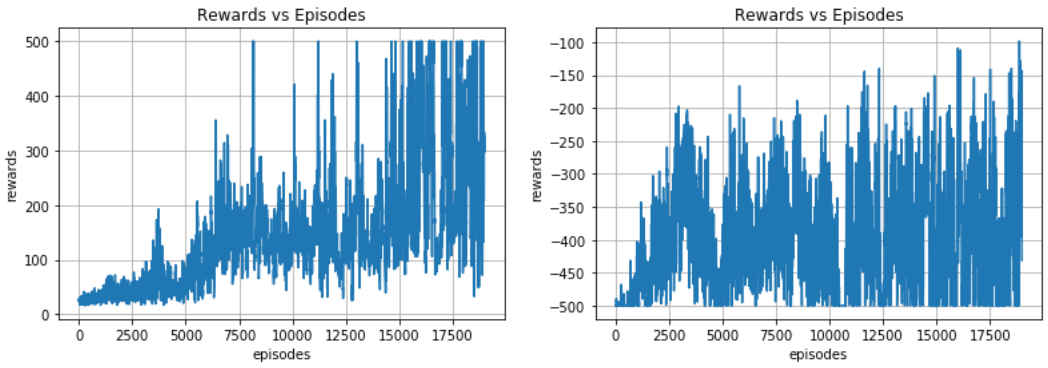

Figure 1: DQN Agent's plots (left is cartpole, right is acrobot)

5.2 Double DQN results

In case of double DQN the results are very close to vanilla with only a minor

difference. Acrobot environment's rewards graphs that we got after running it multiple times varied in terms of at which point the graph stopped going up. But in general it goes up which means it is able to learn but not as good as the vanilla version.

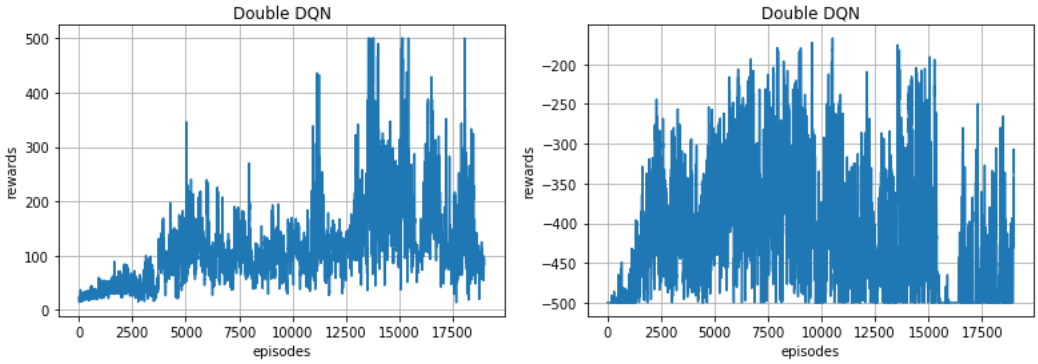

Figure 2: Double DQN Agent's plots (left is cartpole, right is acrobot)

5.3 A2C results

In case of Actor-Critic algorithm, we notice that for a long time from the start of training the rewards are relatively low and towards the end the rewards go up very fast and the agent learns quickly to hit the high score. A2C seemed to struggle with Acrobot environment slightly more than other algorithms.

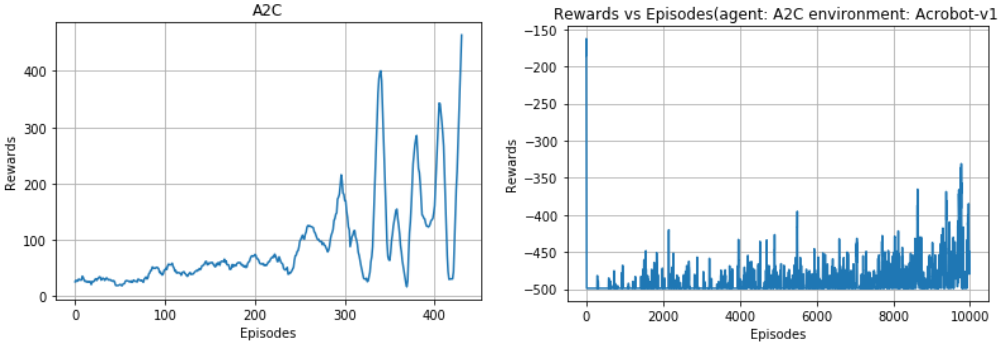

Figure 3: A2C Agent's plots (left is cartpole, right is acrobot)

5.4 Reinforce results

After running REINFORCE on both CartPole-v1 and Acrobot-v1 we have their respective rewards vs episodes graph below. We can clearly notice that this algorithm shows a steady rise in the rewards in both the environments which means that the agent learns steadily over time.

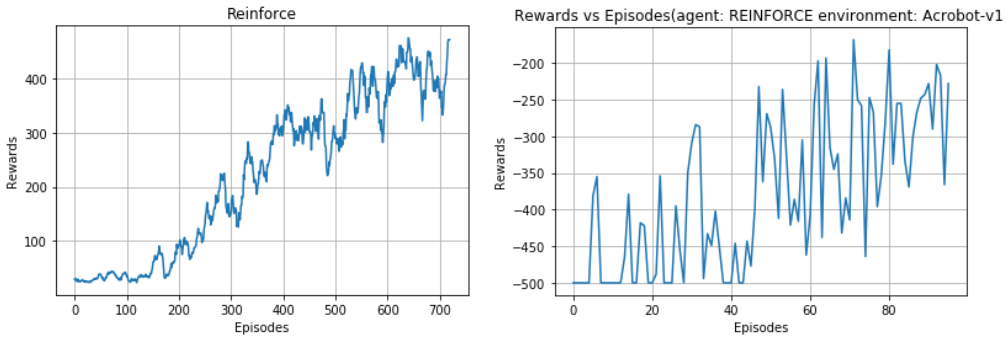

Figure 4: REINFORCE Agent's plots (left is cartpole, right is acrobot)

5.5 PPO results

In case of PPO algorithm, we can see that this algorithm did the best on both environments. We can see clear upward trending graphs on both the environments.

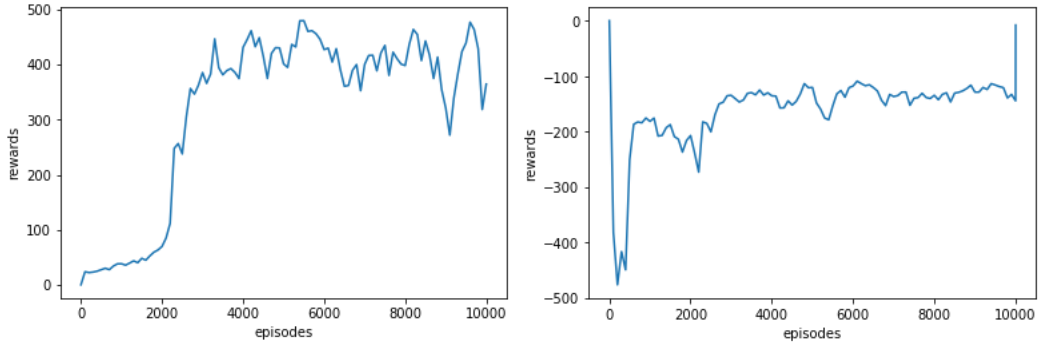

Figure 5: PPO Agent's plots (left is cartpole, right is acrobot)

6 Comparisons of algorithms

When we compare the algorithms based on Cartpole environment, it is hard to say which algorithms did best as all algorithms were able to hit the peak at the end. The only difference is how fast some algorithms were compared to others. A2C seemed to be the fastest one for this environment.

But when we compare the algorithms based on Acrobot environment, it is clear that PPO did the best compared to the rest of the algorithms and was able to reach even a higher peak for rewards than others.

7 Conclusion

All the algorithms performed really well on the cartpole environment however not all did so well on the acrobot environment. A2C did the best on Cartpole environment while PPO did the best on acrobot environment. This shows that PPO should be a good starting point to try when dealing with any complicated environment. While A2C should be a good starting point to try when dealing with a simple environment.

Acknowledgments

I am grateful to Professor Alina Vereshchaka for helping me understand the concepts of different deep reinforcement learning algorithms.

References

[1] http://incompleteideas.net/book/first/ebook/node66.html
[2] UB 510 Reinforcement Learning lecture slides
[3] http://gym.openai.com/docs/
[4] https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py
[5] https://nips.cc/Conferences/2018/PaperInformation/StyleFiles
[6] https://towardsdatascience.com/learning-reinforcement-learning-reinforce-with-pytorch-5e8ad7fc7da0
[7] https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
[8] https://www.endtoend.ai/envs/gym/atari/space-invaders/
[9] https://github.com/openai/gym/wiki/Table-of-environments
[10] https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
[11] http://gym.openai.com/docs/
[12] https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py
[13] https://nips.cc/Conferences/2018/PaperInformation/StyleFiles
[14] https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
[15] https://medium.com/@jonathan_hui/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12
[16] https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf