Assignment 3 - Deep RL

Lisa Samuelsson & Dylan Osolian

Group 19

Question 1

Select two deep reinforcement learning algorithms, where each of them has an implementation you can use. You should select two different algorithms, e.g., you are not allowed to select two different improvements of DQN (see Rainbow paper1). Tell us what algorithms you selected and provide references to the implementations you used.

We selected the algorithms *A2C (Advantage Actor Critic)* and *DQN (Deep Q-Network).* For both algorithms, we used the implementations from *Stable Baselines3* [1], [2].

Question 2

Write a description of each algorithm. What are the key components? Why are they in this way? Why is it a deep reinforcement learning algorithm? Mathematically define all important key concepts. Each algorithm should be explained in such a way that a person with basic knowledge of reinforcement learning, but lacking knowledge about this specific algorithm, should have a good understanding of the algorithm and be able to implement it after reading your description.

A2C (Advantage Actor-Critic)

The A2C algorithm uses an Actor-Critic architecture, which means it uses two networks: a value network (the critic), and a policy network (the actor) [3]. The actor decides how the agent behaves, updating the policy, while the critic checks how good the action is, updating the value function. A2C is a deep reinforcement learning algorithm as it uses neural networks as function approximators for both the actor and the critic networks. These networks are trained using stochastic gradient descent.

At every timestep, the current state is given from the environment. This state is then passed as input both to the actor and the critic. The policy takes this state, and outputs an action. This action is given as input to the critic, which along with the state is used to compute how good it was to take the given action at the particular state. This is the Q-value. Taking this action will result in a new state, and a reward. Using the Q-value generated by the critic, the actor will update its policy. With this update, the actor will do a new action. This process then continues on.

Described above is the underlying structure of the Actor-Critic process. However, in A2C another function is used as a critic instead of the action-value function: the Advantage function. This is another key component of the A2C algorithm. The purpose of using this function is to calculate the advantage of taking a certain action compared to all other possible actions at a state.

This function is defined as follows:

$$A(s,a) = Q(s,a) - V(s),$$

where Q(s,a) is the q value for action a in state s, and V(s) is the average value of state s. What this function calculates is essentially the extra reward given by taking action *a* at state *s* compared to the average reward given at the same state. This would require two value

functions, both Q(s,a) and V(s), but instead, the temporal difference error [4] can be used as an estimator of the advantage function:

$$A(s,a) = r + \gamma V(s') - V(s)$$

The A2C algorithm performs synchronous updates [5]. This means that the actor and critic are updated simultaneously with shared parameters. Another reinforcement learning related to A2C is A3C, which works similarly but asynchronous. According to a blog post by OpenAI [6], the fact that A2C is synchronous allows it to use GPUs more effectively, which makes it perform better with large batch sizes. OpenAI's implementation of the synchronous A2C algorithm had a better performance compared to the synchronous counterpart.

DQN (Deep Q Network)

DQN combines Q-learning with deep neural networks to approximate the optimal action value function [7]. Because neural networks are good at modeling functions, DQN uses a neural network to estimate Q. The main idea behind DQN is similar to Q-learning where we start with initializing the Q-table and then use an epsilon greedy policy to explore. To update the Q-table the algorithm uses the Q-value of both the current action and the next action.

There are three key components in the algorithm: two neural networks; the Q network and the Target network and a third component called Experience Replay. The Experience Replay communicates with the environment to get data which is used to train the Q network. It is the Experience Replay which handles the epsilon greedy policy and performs an action on the environment. The Q network can be anything from a simple linear network to a CNN (convolutional neural network) depending on the data, the Target network is the same kind of network as the Q network. The Q network is trained to predict the Q-value while the Target network predicts a target Q-value. This means that the Q network takes random data from all the available data which has been saved by the Experience Replay and uses the current state and action to get a predicted Q-value. The Target network instead predicts the target Q-value by taking the next state and estimating the best Q-value which can be reached with all possible actions [7].

When we have these two predicted values a loss can be calculated to train the Q network, the target network is not trained since we want the predictions to remain stable. However, after a certain amount of time steps, the weights of the Q network are copied to the Target network so that the predictions still improve. The loss can be computed in different ways but the most common one seems to be by calculating the squared error of the target Q-value

and the predicted Q-value [8]. The described process is then repeated and the experience replay once again interacts with the environment.

Question 3

Write a section comparing the algorithms.

A2C can handle both discrete and continuous action spaces, while DQN is designed for discrete action spaces [9]. However, in the implementation that we have done, we have used a discrete action space for both algorithms.

When it comes to the exploration vs exploitation trade-off, A2C handles this by using entropy regularization [10]. Entropy regularization encourages exploration by adding an entropy term to the loss function. It works by penalizing low policy entropy, meaning that policies that are more deterministic early on get penalized. In DQN on the other hand, exploration is handled separately from the policy itself. Instead, an epsilon-greedy approach is used to balance exploration and exploitation [10].

There is also a difference in the estimation of the values of the different actions. A2C uses the critic network to estimate the value function, while DQN learns the action-value function by using the Bellman equation for updating the Q-function.

Both algorithms are in some senses similar to “non-deep” algorithms covered in the course. Both networks used by A2C are related to “non-deep” methods: the way the critic network works resembles value iteration, and the actor network resembles policy iteration. DQN is based on Q-learning, but instead of the tabular representation a deep neural network is used to handle high-dimensional state spaces.

The performance of the algorithm depends on the environment and the hyperparameters used. Krishna and Yarram [11] implemented several reinforcement learning algorithms, including DQN and A2C, for three different environments: Cartpole, Space Invaders and Lunar Lander. They found that for Cartpole and Lunar Lander, the A2C algorithm was especially sensitive to changes in hyperparameters. However, for Lunar Lander, they found that A2C gave significantly better results in comparison to DQN. We also tried these two algorithms in both the Lunar Lander and the Cartpole environment, and we describe the differences that we observed further in the section below.

Question 4

Run each algorithm, i.e., train each agent from scratch, on two (appropriate) environments and answer the following questions for each of the algorithms:

(a) Describe each environment in a few sentences. What is the state space, rewards, action space, observations, etc.? Why is it appropriate to use a deep RL algorithm in this environment? Why is the selected algorithm suitable for this environment (preferably you should refer to your previous description of the algorithm)?

Lunar Lander

Lunar Lander is an environment that is part of the BOX2D environments. It is a game environment that revolves around physics control, rendered with PyGame [12]. The goal is to land a lander between two flags (the landing pad). Rewards are given based on where the lander is positioned, if and where it lands. Positive rewards are given for being in the air above the landing pad, and a larger positive reward is given for landing on the landing pad. If it lands outside of the landing pad or if it crashes, it receives negative rewards.

The lander starts in the air above the landing pad, and initially it has a random force applied to it. The action space is discrete as default, with the actions “do nothing”, “fire left engine”, “fire right engine” and “fire main engine”. The action space can also be set to be continuous, but we have not used this in our implementation. An 8-dimensional vector describes the state. It contains the lander's velocities in x and y, its coordinates in x and y, its angular velocities and one boolean for each leg that represents if the leg is touching the ground or not. An episode ends if the lander crashes, moves outside of the screen, or if it finishes the assignment of landing on the landing pad. As the state space is eight-dimensional, it is appropriate to use a deep RL algorithm as they handle high-dimensional state spaces well.

CartPole

This environment is a classic control environment where the goal is to balance a pole on a cart by applying forces on the cart [13]. The action space is discrete and there are 2 possible actions, applying force in the left direction or applying force in the right direction. The state space is defined by 4 values: Cart position, Cart velocity, Pole angle and Pole angular velocity. The observations consist of all of these values. An episode in CartPole ends when the pole is no longer upright (pole angle greater than +- 12 degrees.), so a reward of +1 is given for every timestep in the experiment. Deep RL is appropriate for CartPole considering

the complexity of the environment. Since the actions are discrete both DQN and A2C will work.

(b) Describe in detail how the training was done on each environment. How was the training performed and why? What were the values of the hyperparameters of the algorithm? For every function approximator, what network architecture was used (including activation function, parameters initialization, etc)? Preferably display the values of all hyperparameters in a table (one table for every algorithm-environment combination).

For both algorithms, we trained one environment at a time. We used the predefined parameters in the implementation, as described in the tables below. In the table, we also display some properties of the network For both algorithms and both environments, we trained for a predetermined number of timesteps: 10^6. Ideally, we would have liked to perform more runs with different hyper parameters and more time steps, but we were limited by the time it took to run the algorithms.

To train the agents, we began by importing the algorithms from Stable Baseline. We then created the environments and set up a logger to get a CSV file of the training results. After that, we defined the model, and selected the policy. For all combinations of the algorithms and environment, we used the MLP (Multi Layer Perceptron) Policy, which means that a state vector is given as input to our model. The other policy option is the CNN (convolutional neural network) policy, but this policy handles images which is not what we are working with in these environments. The next step is simply to call the learn function on the created model, sending in the selected number of timesteps as a parameter.

Hyper parameters DQN

Hyper parameter	**Value**
Activation Function	ReLu
Optimizer	Adam
Learning rate	0.0001
Exploration factor	0.1
Discount factor	0.99
Tau	1.0
Batch size	32

Exploration initial eps	1.0
Exploration final eps	0.05
Training frequency	4
Learning starts	50000

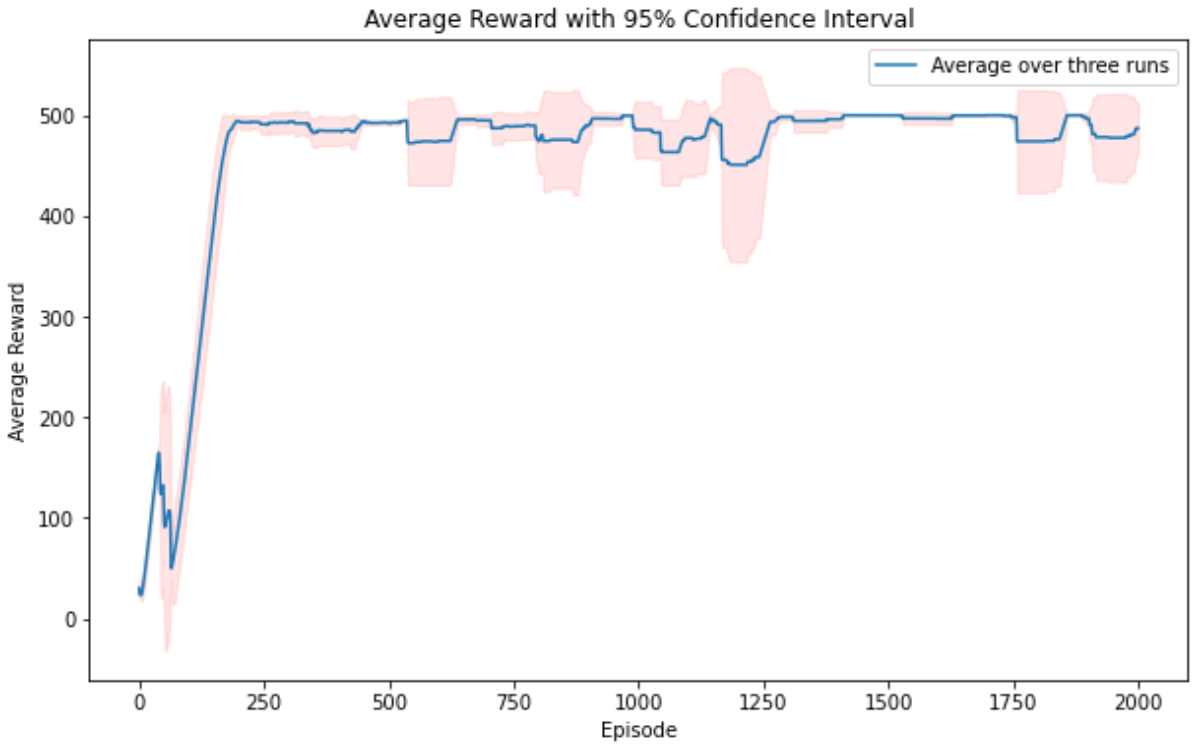
Hyper parameters A2C

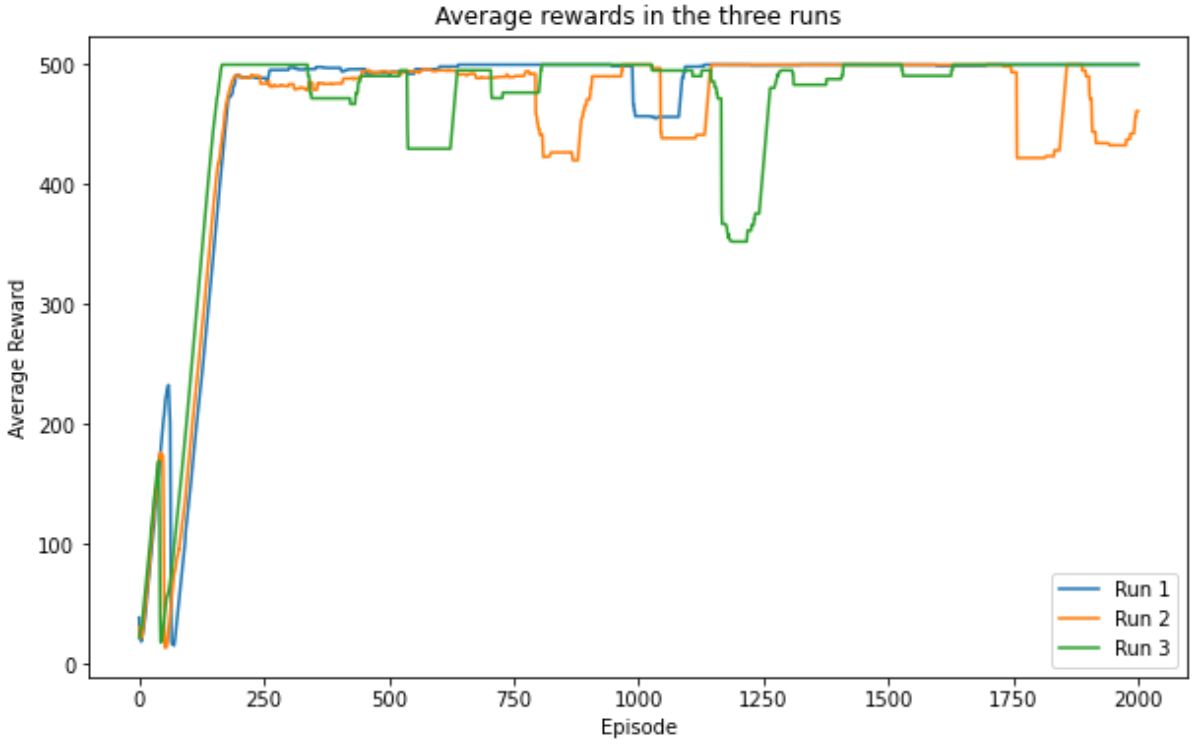
Hyper parameter	**Value**
Activation function	Tanh
Optimizer	RMSprop
Learning rate	0.0007
Discount factor	0.99
Entropy coefficient for loss calculation	0.0
Value function coefficient for the loss calculation	0.5
The maximum value for the gradient clipping	0.5

(c) Plot and briefly describe the results. For each environment, we want you to show and briefly describe how the agent improves while it interacts with the environment. You should show the average (together with some measure of variation) over at least 3 runs. For instance, plot the moving average of the rewards (together with some measure of variation) by episode, averaged over 3 runs, and describe the behavior of this curve.

A2C - CartPole

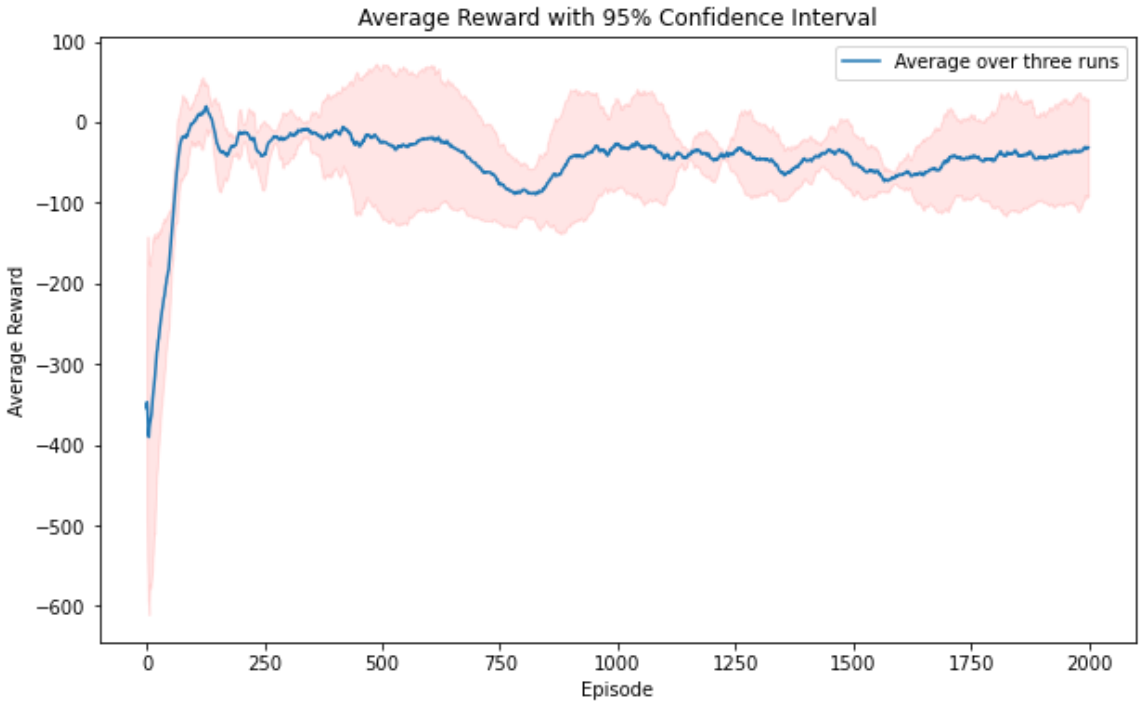
Below is the plot for the three runs of the A2C algorithm on the CartPole environment. The plot shows that the algorithm quite quickly learns and converges towards an average reward of 500 after about 150-200 episodes. The three runs perform quite similarly which results in a rather narrow confidence interval, which we can see in the figure below.

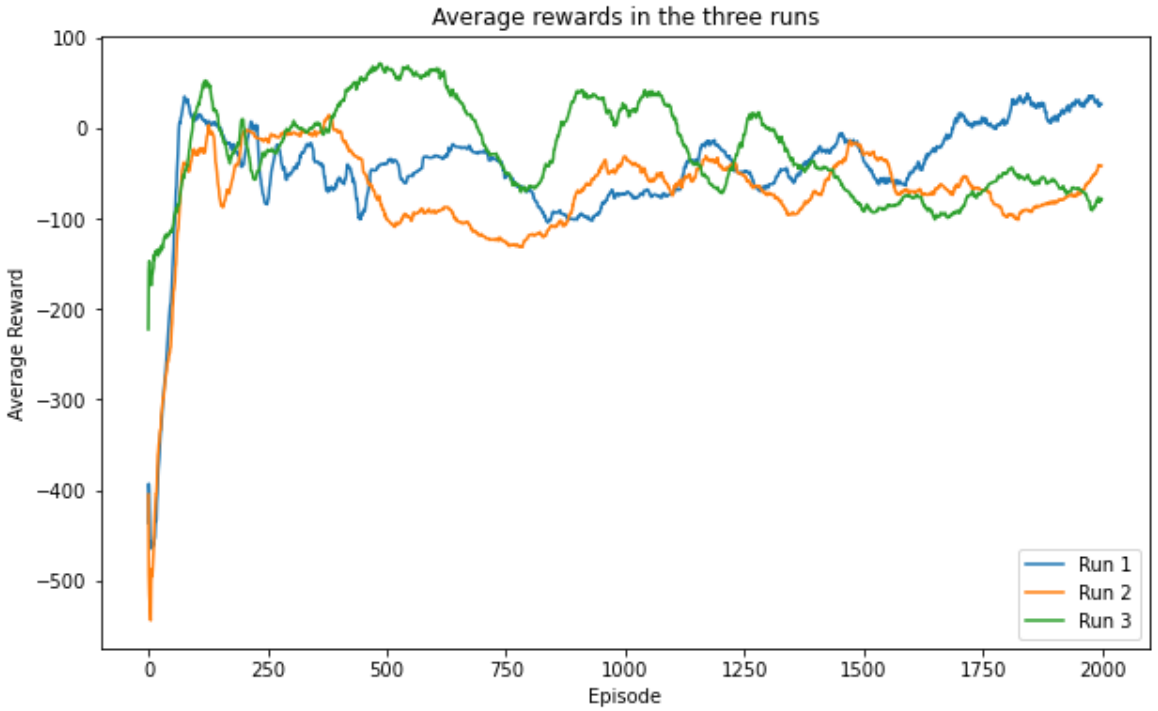

Average Reward with 95% Confidence Interval
Average over three runs
Average Reward
Episode
0
100
200
300
400
500
0
250
500
750
1000
1250
1500
1750
2000


Average rewards in the three runs
Run 1
Run 2
Run 3
Average Reward
Episode
0
100
200
300
400
500
0
250
500
750
1000
1250
1500
1750
2000

A2C - Lunar Lander

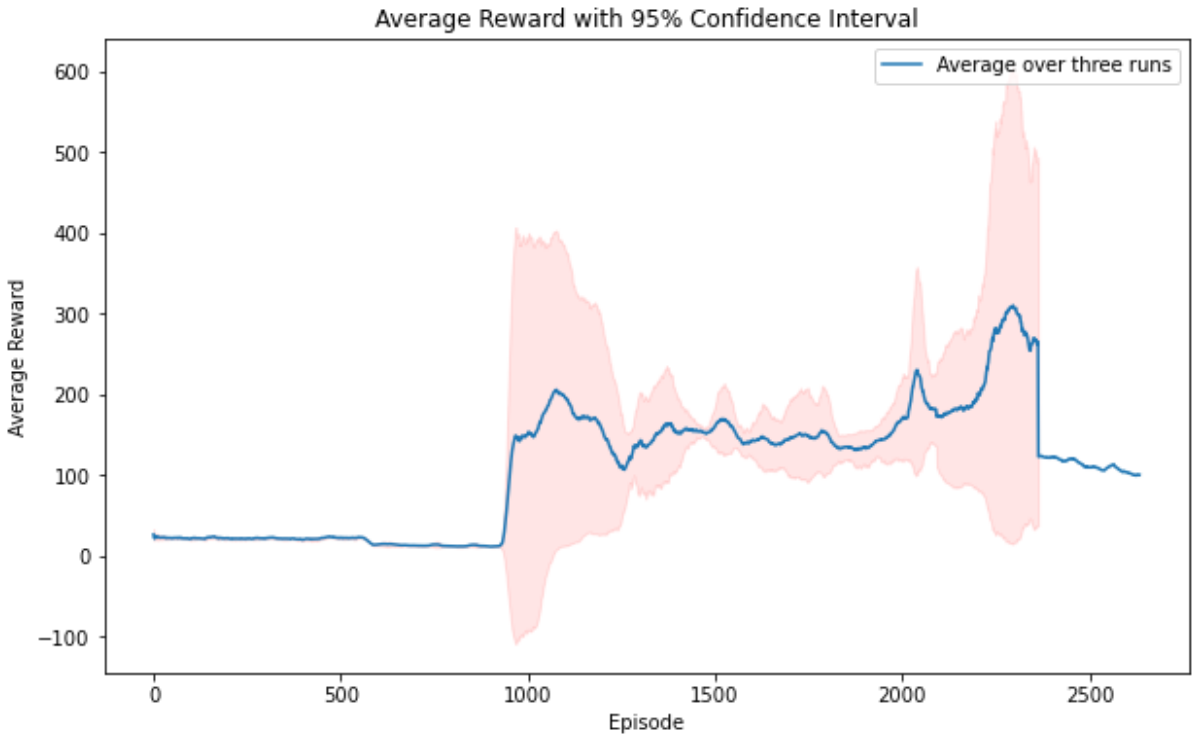
These plots show the performance of A2C on Lunar Lander, again the algorithm quite quickly learns and converges. However worth noting is that the rewards seem to be between 0 and -100 which indicate that the algorithm doesn't solve the game by landing the rocket in 2000 episodes. There is some variation between the runs, but the overall shape looks rather similar.

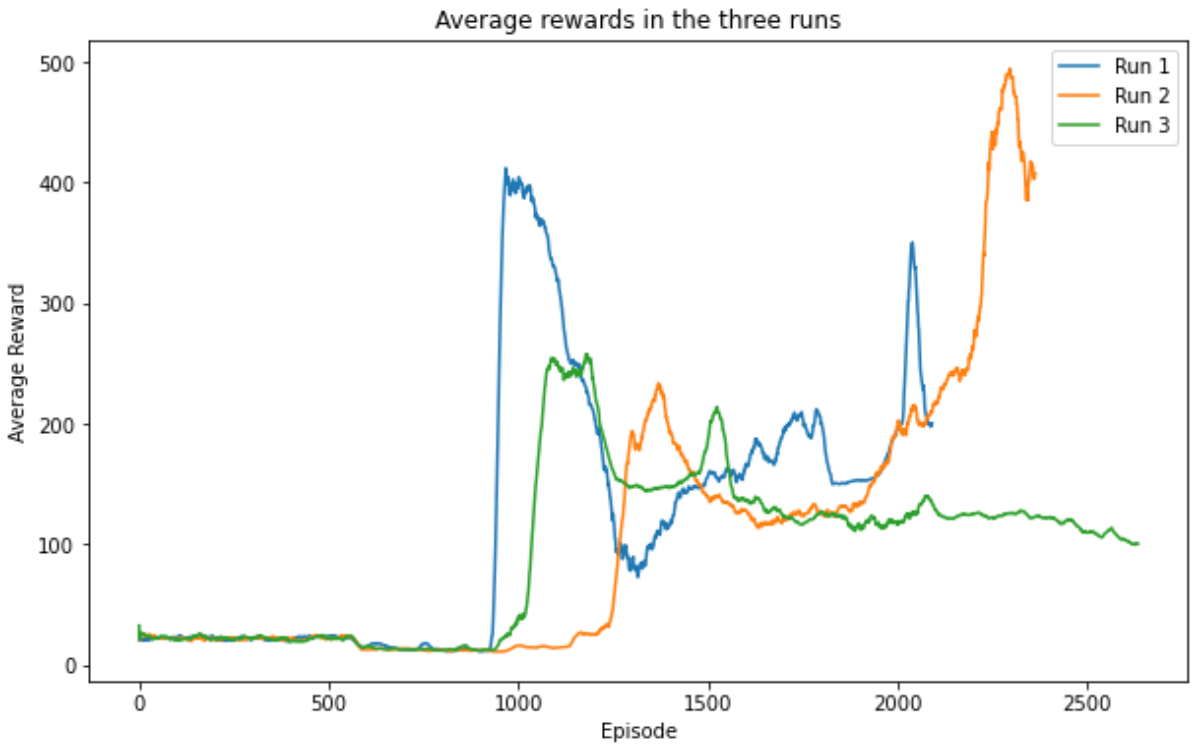




DQN - CartPole

This is the plot for the three runs of the DQN algorithm on the CartPole environment. In this case, we don't see any convergence. It is interesting that the average rewards seem to be quite low and constant for a long time before the algorithm learns how to do better. The variations between the three runs are rather big, which is clearly displayed in the figure that displays the confidence interval for the average rewards.

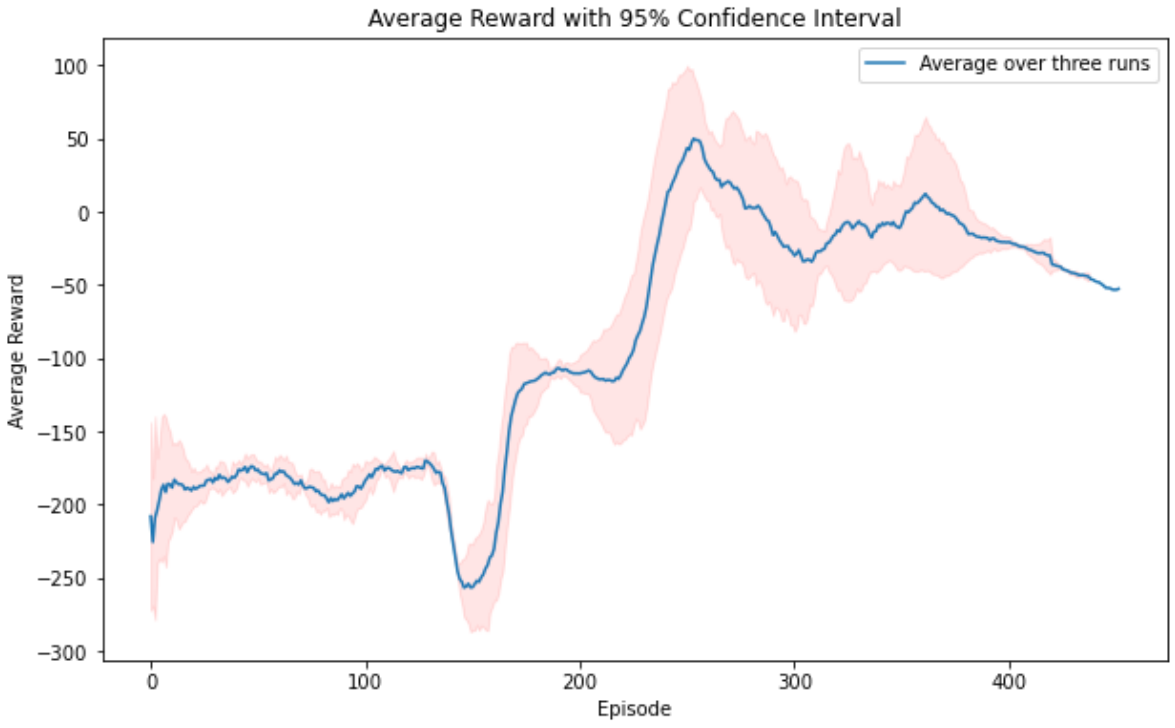


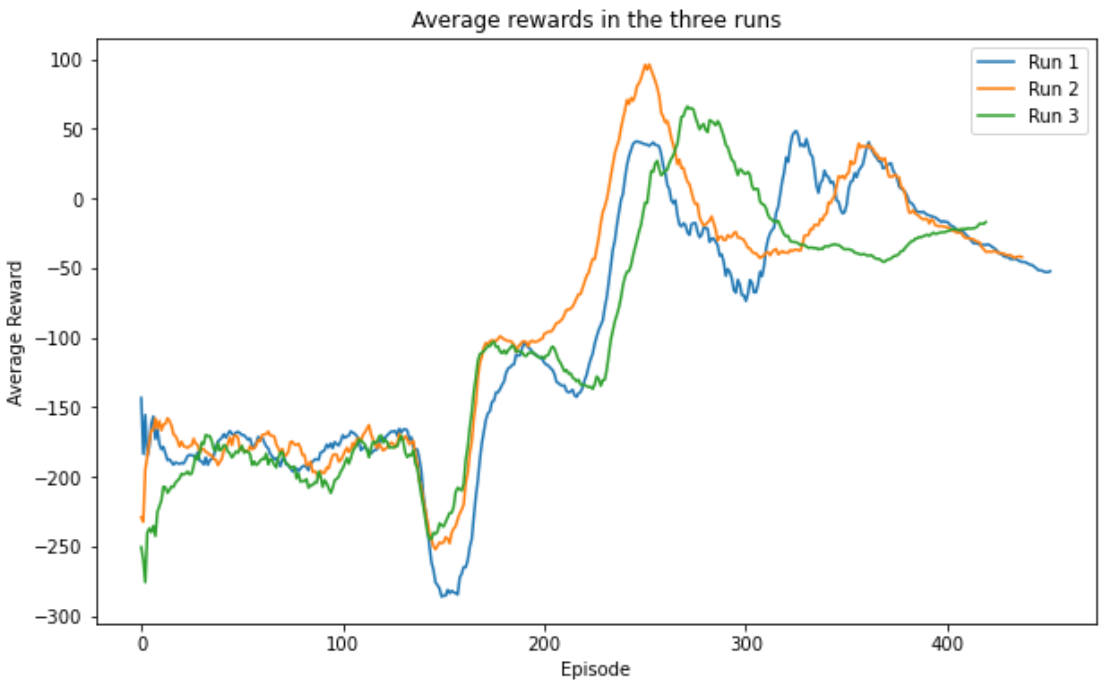


DQN - Lunar Lander

These are the plots for DQN on Lunar Lander. The algorithm was trained for one million time steps but this resulted in only 400 episodes which is less than the other trainings in this report. The algorithm doesn't seem to converge in this number of time steps however it is possible to see an improvement and that the algorithm learns.

The average rewards show a lot of variation, but interestingly the shapes are quite similar for all three runs. Especially notable is the dip in average rewards around 150 episodes, which appear on all three runs. We have no good explanation of why this pattern occurs, but it seems as if something in the combination of the environment and the algorithm makes this dip happen. It happens after a rather stable period, so it seems as if the algorithm starts exploring alternative solutions and doesn't stop until reaching a large negative reward. It would be interesting to carry out more runs to see if this pattern always appears. It is possible that the dip doesn't make the algorithm perform worse in the long run, but perhaps it could be eliminated by tuning the hyper parameters.




Average rewards in the three runs
Run 1
Run 2
Run 3
Average Reward
Episode
100
50
0
−50
−100
−150
−200
−250
−300
0
100
200
300
400

Question 5

Analyze and compare the results of the different agents and environments. What are your conclusions? What did you expect? What did you not expect? How does your (trained) agent perform in each environment? For the same algorithm, how does the performance differ between the environments? If you have run both algorithms in the same environment(s), how does the performance differ? You are encouraged to analyze anything that you find interesting. Clearly explain and motivate your conclusions. Try to explain why you observe differences and/or similarities in performance.

Looking at how the agents performed in the CartPole environment we can see that A2C performed better than DQN. A2C quickly converges and is quite stable with rewards around 500 while DQN is more unstable with lower average rewards. In Lunar Lander A2C still performs better than DQN. However here the difference seems to be a lot less. In the end of the training the performance of DQN is quite similar to A2C. Although again A2C seems to be more stable than DQN, but it is possible that DQN would also converge and become more stable with more timesteps.

We went into this assignment without many expectations, as we had limited prior knowledge about these algorithms and with the environments. It was however interesting to see how the algorithms differed in performance, although we felt that we haven't performed enough experiments to explain all of our observations.

When reading about the environments we selected, we believed that the Cart Pole environments would be easier for the agents to perform well in when compared to the Lunar Lander environment. In the beginning of the assignment, we looked at pretrained agents from the Stable Baseline implementation to get a grasp of how they performed in the different environments, and noticed that the Lunar Lander environment seemed to be more challenging than the Cart Pole environment. As mentioned above, the Lunar Lander environment has both a larger state space and a larger action space, which explains why it's more difficult for the agent to learn. Therefore, it wasn't surprising to us that the agents generally performed better in the Cart Pole environment. However, the DQN algorithm still had very varied results for this environment, and to draw more informed conclusions we would have to perform more runs. It would also be beneficial to run the algorithm longer as we don't see any convergence.

There can be many reasons for the differences in performance. One area which affects performance is hyperparameters. We only used the default hyperparameters from the implementation and did no tuning ourselves which will affect how the algorithms perform. Had we tuned the hyperparameters for each algorithm and environment it is likely that the outcome would be different. Another reason for the difference in performance between the algorithms could be the amount of timesteps we used for our training. It looked like DQN needed more time steps in both environments and it is possible that DQN simply needs more time to converge than A2C.

An obvious reason for the differences in performance between the environments is the environment's complexity. In general we see slightly better performance on the CartPole environment and this is likely due to the fact that CartPole has a smaller action space and state space than Lunar Lander, as mentioned above.

We think that it is hard to draw conclusions from the experiments we have performed, as we have only done a limited amount of runs for each environment, all with a limited number of time steps and without extensive hyper parameter tuning. If we would have reached convergence for all algorithms and environments, it would have been easier to draw conclusions about the performance. One conclusion is instead that there are many aspects that go into training agents and that careful considerations of hyper parameters, time horizons and algorithm selection is needed to reach good performance.

The assignment has given us experience in working with deep reinforcement learning algorithms and we feel like we have gotten a better understanding of some of the important concepts of deep reinforcement learning. We have looked at implementations of algorithms and also worked with some of the practical aspects like training and using different environments.

References

[1] A. Raffin, RL Baselines3 Zoo, 2020, GitHub, GitHub repository, https://github.com/DLR-RM/rl-baselines3-zoo

[2] A. Raffin, et. al., “Stable-Baselines3: Reliable Reinforcement Learning Implementations”, *The Journal of Machine Learning Research (JMLR)*, 22(268): 1−8, 2021, http://jmlr.org/papers/v22/20-1364.html.

[3] C. Yoon, “Understanding Actor Critic Methods and A2C”, *Towards Data Science*, [Online], Feb. 6, 2019. Available: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f (accessed on: 2023-05-17).

[4] B. Roy, “Temporal-Difference (TD) Learning”, *Towards Data Science*, [Online], Sep. 12, 2019. Available: https://towardsdatascience.com/temporal-difference-learning-47b4a7205ca8 (accessed on: 2023-05-17).

[5] Hugging Face community, “Advantage Actor-Critic (A2C)”, Available: https://huggingface.co/learn/deep-rl-course/unit6/advantage-actor-critic?fw=pt (accessed on: 2023-05-17).

[6] OpenAI, Y. Wu et. al., “OpenAI Baselines: ACKTR & A2C”, Available: https://openai.com/research/openai-baselines-acktr-a2c (accessed on: 2023-05-17).

[7] K. Doshi, “Reinforcement Learning Explained Visually (Part 5): Deep Q Networks, step-by-step”, *Towards Data Science*, [Online], Dec. 19, 2019. Available: https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b (accessed on: 2023-05-17).

[8] D. Karunakaran, *Medium* [Online], Sep. 21, 2020. Available: “https://medium.com/intro-to-artificial-intelligence/deep-q-network-dqn-applying-neural-network-as-a-functional-approximation-in-q-learning-6ffe3b0a9062” (accesed on: 2023-05-24.

[9] J. Zhu, F. Wu, J. Zhao, “An Overview of the Action Space for Deep Reinforcement Learning”, *ACAI '21: Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence*, Sanya, China, 2021, pp. 1-10, Available: https://dl.acm.org/doi/10.1145/3508546.3508598 (accessed on: 2023-05-17).

[10] Z. Ahmed et. al., “Understanding the Impact of Entropy on Policy Optimization”, *International Conference on Machine Learning (ICML) 2019*, Long Beach, US, 2019, Available: https://doi.org/10.48550/arXiv.1811.11214 (accessed on: 2023-05-18).

[11] University of Buffalo, Department of Computer Science and Engineering. Comparison of Reinforcement Learning Algorithms. V. Vamsi Krishna, S. Yarram. Available:

https://cse.buffalo.edu/~avereshc/rl_fall20/Comparison_of_RL_Algorithms_vvelivel_sudhirya.pdf (accessed on: 2023-05-18).

[12] O. Klimov, Lunar Lander, 2022, Gym Documentation,
https://www.gymlibrary.dev/environments/box2d/lunar_lander/

[13] Cart Pole, 2022, Gym Documentation,
https://www.gymlibrary.dev/environments/classic_control/cart_pole/