Solving the LunarLanderContinuous-v2

Eran Shmuel (ID. 301796561), Moshe Binieli ID. 311800668)

Submitted as final project report for the RL course, BIU, 2021

1 Introduction

Deep neural network (DNN) is a revolutionary algorithm for supervised and unsupervised learning. In the last few years, researchers were able to use DNN with reinforcement learning to solve complicated problems, including games. In this project, we used some of the latest Deep Reinforcement Learning algorithms in order to solve the Lunar Lander Continues V2 game. We also solved a more complicated problem by adding Gaussian noise to the observed state. We ended up solving the Lunar Lander Continues V2 game in 58 iterations, and in 60 iterations in the noisy environment.

1.1 Related Works

Q-learning is an important algorithm in the field of reinforcement learning, the algorithm tries to maximize the expected reward when choosing an action a with the state s, by using the update rule:

$$Q(s,a) \leftarrow Q(s,a) + a \cdot (r(s,a) + \gamma \cdot max(Q(s',a)) - Q(s,a))$$

In the last few years, due to the rise of neural networks, the algorithm can be done in environments where the state is continuous: instead of using the traditional tabular methods, neural networks allow to compute Q(s, a) where s is continuous[1]. In order to achieve stable training, it was found that saving replay memory as a tuple of (state, action, reward, observation) is essential for training[1]. Later it was discovered that adding a target network, which is the same network from a few iterations ago, is good for computing the target Q-Values of the neural network. Later on, more algorithms were founded, such as Double Deep Q-Learning[2] that uses the target network for only computing the Q-Value of the selected action, which is chosen by the original network. Dueling neural network [3] uses a more complex network architecture by dividing the network to compute the state value V(s) and every action advantage A(s, a), only to return the sum A(s, a) + V(s) - mean(A(s)), in order to help the back-propagation process. Expected Sarsa is another algorithm very similar to Q-learning, the only difference is that instead of choosing the max value of the next state, it chooses the expected value of the next state according to the policy. This algorithm also improved some of the models[4] .

2 Solution

2.1 General approach

We used some of the most successful Deep Reinforcement Learning algorithms including DQN, Double DQN, Dueling DQN, Double Dueling DQN, Expected Sarsa and Double Expected Sarsa. We also tried to add different features such as replay memory, epsilon greedy, discount factor and target network. We wanted to compare the networks performance while also trying to solve the game as quickly as possible.

2.2 Design

Action Space

Since the task is a continuous task and we want to convert it to a discrete task, we introduce the concept of Action Space. The action space of the algorithm takes an action index and returns a tuple of values: a value for the main engine and a value for the left-right engine. Because any value between -1 to 0 in the main engine and any value between -0.5 to 0.5 in the left-right engine is equal, we took only 1 action that represents those values.

Networks

For DQN and Double DQN models, we used a deep neural network architecture that contains 3 hidden linear layers with sizes 128, 256, 256 with ReLU activation function, which outputs the Q-Values.

For Dueling DQN, Double Dueling DQN and Expected Sarsa models , we used a deep neural network architecture that contains 3 hidden linear layers with sizes 128, 256, 256 with ReLU activation function to compute the features vector X. We then used another linear layer to computer the "Advantage" vector and the difference linear layer to compute the value. The network returns:

$$advantage + value - mean(advantage)$$

Hyper parameters:

- Epsilon - We used epsilon with a start value of 1, and a decay factor of 0.975, which means that after each iteration we multiplied epsilon by 0.975.
- Batch size - We used a batch size of 256. Since we're being tested by the number of iterations and not by running time, a big batch size worked best for us, otherwise we would've decrease the batch size.

- Target reset - We copy the model parameters to the target model every 200 steps (a step is done after every action), except the models which had no discount factor effect, in which we copy the parameters every 5 iterations.
- Gama - We used a discount factor of 0.99.
- Learning Rate - We used a learning rate of 0.00005.
- Replay Memory Size - The maximum size of the replay memory is 20,000.
- Number of Actions - The number of actions is 30: 2 for the main engine (-1,1), and 15 for the left-right engine.
- Number of frames - In most of the regular models (which didn't operate in a noisy environment), we used only the current state (figure 1). In the noisy environment, we used the last 4 states.

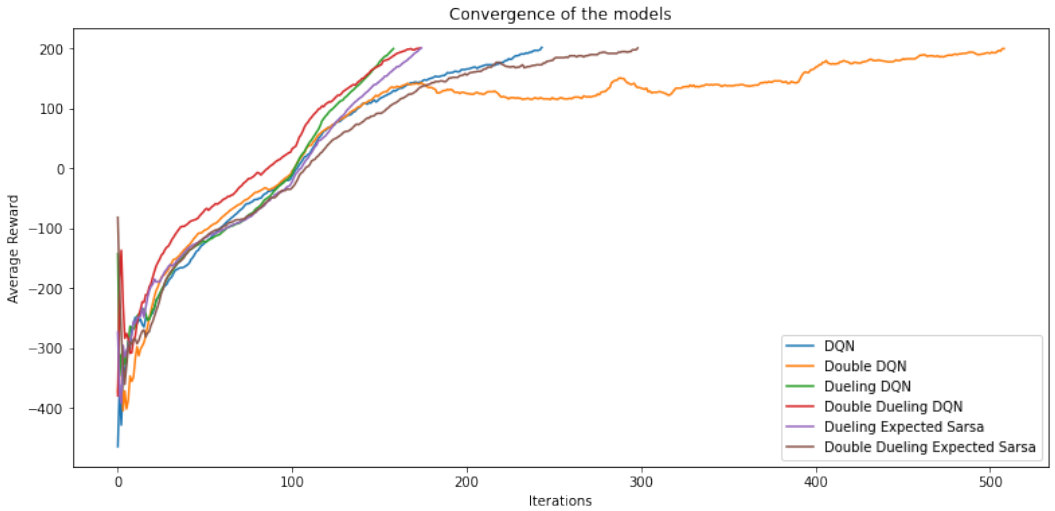
3 Experimental results

We built the DQN model with different kinds of features: replay memory, epsilon greedy, discount factor and target network. We tried to run the model with different combinations in order to see how each feature effects the results. The model converged at each of those combinations, except the ones without any features, as you can see in Figure 1.

Model	Frames	Experience replay	Target network	Epsilon greedy	discount factor	Solved after x iterations
DQN	1	x	x	x	x	failed
DQN	1	✓	x	x	x	3821
DQN	1	✓	✓	x	x	661
DQN	1	✓	✓	✓	x	488
DQN	1	✓	✓	✓	✓	143
Double DQN	1	✓	✓	✓	✓	408
Dueling DQN	1	✓	✓	✓	✓	**58**
Double Dueling DQN	1	✓	✓	✓	✓	73
Dueling Expected Sarsa	1	✓	✓	✓	✓	74
Double Dueling Expected Sarsa	1	✓	✓	✓	✓	198
DQN	4	✓	✓	✓	✓	235
Double DQN	4	✓	✓	✓	✓	169
Dueling DQN	4	✓	✓	✓	✓	62
Dueling Expected Sarsa	4	✓	✓	✓	✓	60

Experimental results of Lunar Lander Continuous game without noise

We also built double DQN, Dueling DQN, Double Dueling DQN, Dueling Expected Sarsa and Double Dueling Expected Sarsa models with all the features above, and compared them to the DQN model. The Dueling DQN model provided the best results, as you can see in figures 1,2.

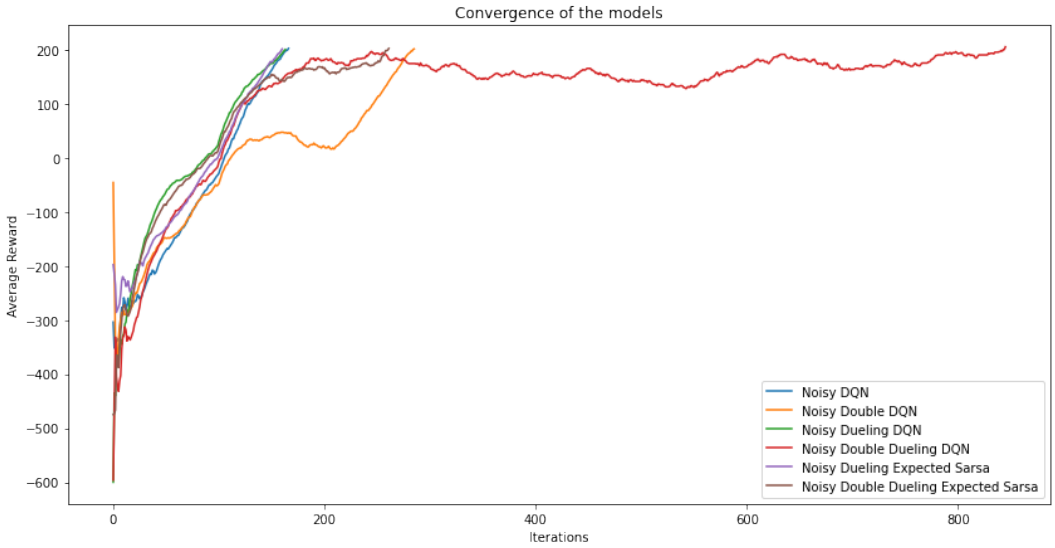


Convergence of the models in the normal environment

In the noisy environment, we tried to run all our main 6 models with the same parameters but they didn't converge in a reasonable time. After we increased the number of frames to 4, all of the models converged, with Dueling Expected Sarsa providing the best results (figures 3, 4). We tried to add frames in the normal environment, but while it did improve some of the models (figure 1), none of them outperformed the Dueling DQN model.

Model	Frames	Experience replay	Target network	Epsilon greedy	discount factor	Solved after x iterations
DQN	4	✓	✓	✓	✓	66
Double DQN	4	✓	✓	✓	✓	185
Dueling DQN	4	✓	✓	✓	✓	63
Double Dueling DQN	4	✓	✓	✓	✓	745
Dueling Expected Sarsa	4	✓	✓	✓	✓	**60**
Double Dueling Expected Sarsa	4	✓	✓	✓	✓	161

Experimental results of Lunar Lander Continuous game with noise



Convergence of the models in the noisy environment

4 Discussion

From the experiments, we came to a conclusion that the experience replay feature is essential for the convergence of the models, as the model without it didn't converge after more than 15,000 iterations. We could also see that the target network is very important for the model, as it speeds up the convergence up to 593%. Epsilon greedy feature also helped the model, but it wasn't important as the previous two. Discounting factor feature was very important when we copied the target network parameters more often. We could also see that the Dueling network provided the best results, while the double networks seems to be worst in the DQN, the Dueling DQN and the Expected Sarsa models. In the noisy environment, adding previous states to the input was essential for solving the problem, and Dueling expected Sarsa provided the best results.

5 References

1. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou Daan Wierstra, Martin Riedmiller: Playing Atari with Deep Reinforcement Learning, 2013

2. Hado van Hasselt, Arthur Guez, David Silver: Deep Reinforcement Learning with Double Q-learning, 2015

3. Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas: Dueling Network Architectures for Deep Reinforcement Learning, 2016

4. Michael Ganger, Ethan Duryea, Wei Hu: Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning, 2016

6 Code

Code can be found in the private repository:
https://github.com/eran88/Lunar-Lander-Continuous2