Project 2

Nicolas Cofre
nicolas.cofre@gatech.edu

Git hash of last commit

be47afb24ad2c59b586931bdb36e9c708
2a6e5f5

Abstract—**This project consists in the solution of the Lunar Lander task from OpenAI Gym environment. The objective is to safely land the spacecraft between two flags that determine the ideal landing position. The use of the engine, crashing and missing the target are costly. Using the Deep Q Network reinforcement learning methodology, I look for a policy that maximizes the reward.**

I. The problem

The problem state has 8 dimensions. 6 continuous dimensions that give information about position, angle and speed. The other 2 dimensions indicate whether each of the two legs of the spacecraft are touching the ground or not. The action space is a discrete 4 dimensional state: fire right engine, fire left engine, fire main engine or nothing.

II. Learning algorithm used: Deep Q Network

Deep Q Network (DQN) [1] is closely related to the well-known tabular Q-learning algorithm. The DQN implementation used here can be briefly described as follows:

- The Q(s,a) function is approximated using a neural network, that maps the state space S to the action space A. Q(θ): S → A.
- The Q(θ) has a lagged copy or target network Q(θ^{-}).
- There is a memory buffer or experience replay memory which is used to draw samples from it for training. This buffer stores the transitions states, action and reward: s,a,s′,r.
- Using a sample D from this transition memory buffer, uniformly drawn, we update the parameters of the network in order to minimize $L(\theta, D) = Q(\theta, D) - r - \gamma Q(\theta^{-}, D)$, for the observed rewards transitions and a given discount factor γ. Note that the lagged target network is not affected in this optimization, it is not a function of θ. The same as r.

The detailed algorithm can be found in [1]. One of the main differences with Q-learning is that now we are not using the last observation for the immediate update. Instead, we are storing this last observation in a memory buffer and then sampling from that buffer to update the model using a batch of observations.

III. Hyperparameter tuning

We have the following hyperparameters to tune in this model, and the best model uses the value indicated here:

1. Learning rate for the network parameters update lr, fixed at 1e-3.
2. Discount rate for the rewards γ, 0.99.
3. Iterations between target network updates TNU, 10.
4. Memory buffer size, 1000.
5. Batch size for the training set D, 50.
6. Design of the neural network used. For this case I have used ReLu activations for the only hidden layer and a linear activation for the output layer. 50 units in the hidden layer.
7. Exploration rate used to decide between the greedy policy or random action, $\epsilon = 0.99$.
8. Decay rate of the exploration rate applied in the following way $\epsilon' = \epsilon \cdot decay. decay = 0.99$

A. Gamma (γ)

Hovering was observed if γ was set too high, for instance $\gamma = 1$. Intuitively, this means that the spacecraft is very patient, meaning that it prefer to delay the reward of the landing at the expense of more fuel consumption but avoiding to crash. If we decrease the value of gamma, the spacecraft would be more in a hurry to get the landing reward before the time decreases the payoff due to discounting. In some cases, the spacecraft prefered even to go and fly away rather than crashing or attempting to land.

B. Memory Buffer

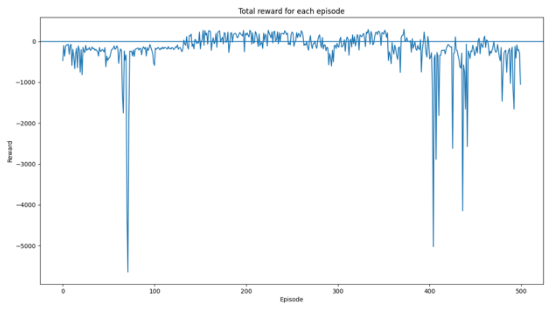
The size of the memory buffer seems to play and important role. I have tested a very small 100 which usually ended up in the model overfitting to the last observations. This implies, for instance, forgetting what was learned in previous interations.

C. Size of the training batch

Another parameter that seems to greatly affect the performance is the size of the batch used for training, the size of the D set. Setting this value in a relatively small value of 50 seemed to work better than bigger values like 100. But it seems to be an interaction between this value and the size of the memory buffer. My intuition is that the smaller the sample of the batch relative to the memory buffer size, the lower the correlation between the samples in D, which improves the performance of the training.

D. Agent during training

We can see how the agent learns how to properly solve the problem after the episode 150 approximately, but at the end close to 400 it starts to get worse.

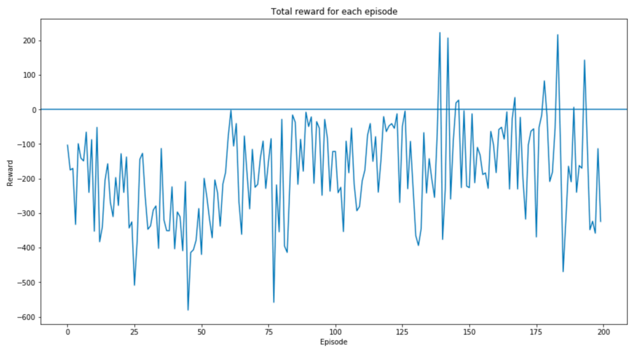


IV. Effect of different hyperparameters

Due to computational time restrictions, here we ilustrate the effect found over the parameters stated previously with a exaggerated change in the values, intead of using a fine grid to see the effect.

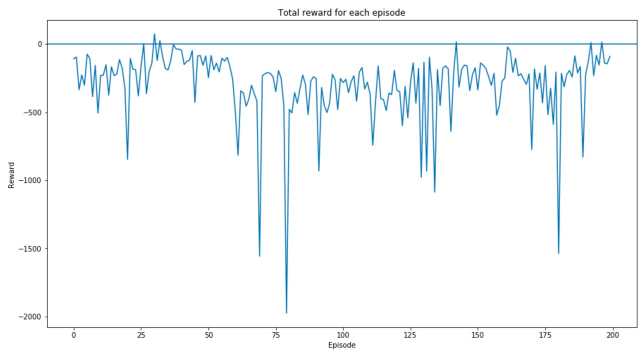
A. *Decreasing gamma*

In this case, the agent is not very patient about getting the positive reward of landing correctly and also cares less about crashing due to discounting. $\gamma = 0.6$.



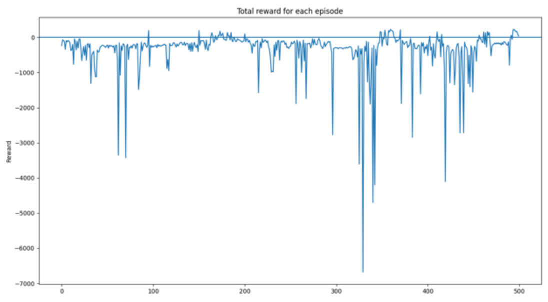
B. *Decreasing memory buffer size*

In this case, the agent quickly forgets previous experiences, and overfit to recent experiences. Memory buffer size used is 100.



C. *Size of the training batch*

In this case, I have increased the training batch to 1000. We completely remove the independence of the observations as the agent is using the whole buffer to train.



V. Conclusion

During the last runs of the model, the agent managed to land properly in many episodes. Even small changes in the hyperparameters can cause the model to completely fail. If I had more time for this task, I would like to explore on the hyperparameter tuning in the reinforcement learning literature.

[1] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." *Nature* 518, no. 7540 (2015): 529-533.