CS7642 Project 2 Report

Github hash: `01c6c69520ce8984d02b121bf59801f809fe1321`

IMPORTANT: all files are in Project 2 folder

INTRODUCTION

The goal for this project is to train an agent that could successfully land a lunar lander, using the 'LunarLander-v2' environment from OpenAI Gym. Compared to the grid-world problem that was solved before, this 'LunarLander-v2' environment is challenging in 2 ways: First, there are infinitely number of states, but only an limited number of states can be stored in computer memory. Second, since stored states are a subset of all states, there needs to be a mapping mechanism from stored states to actions. None of these challenges have been addressed before in the class.

DEEP Q-LEARNING

An effective algorithm that can address the challenges is Deep Q-Learning. It is a combination of Deep Neural Network and Q-Learning. In Deep Q-Learning, instead of maintaining a Q-table as in those finite state problems, Q-table is 'stored' in a trained neural network model: for any given state, a pre-trained neural network model predicts Q values for each corresponding actions, and the correct action is selected with the highest Q value. Compared to conventional Q-Learning algorithm, the biggest change occurs at the step where an action is selected from a state, which is tabulated below.

Q-Learning	**Deep Q-Learning**
action = argmax(corresponding Q of a state)	corresponding Q of a state = model.predict(state) action = argmax(corresponding Q of a state)

In Deep Q-Learning, for each state the Q values corresponding to each action is predicted using a trained neural network, which excludes the necessity of keeping and updating a Q-table. Therefore for Bellman equation, all Q-values are calculated as such (Figure 1).

$$Q(S_t, A_t) \leftarrow \underbrace{Q(S_t, A_t)}_{\text{model prediction}} + \alpha\Big[R_{t+1} + \gamma \underbrace{\max_a Q(S_{t+1}, a)}_{\text{model prediction}} - \underbrace{Q(S_t, A_t)}_{\text{model prediction}}\Big]$$

Figure 1. Bellman Equation Q value updates in Deep Q-Learning

At each step of an episode, a batch of randomly selected states are the input features and updated Q values are the input targets for neural network training. As weights for each node are updated every time a model is re-trained, it is expected that rewards from preferable actions will eventually guide the agent to learn to achieve the target (reward > 200 for 100 consecutive episodes, and 'target' is short for this definition in the following discussion).

RESULTS AND DISCUSSION

In the currently Deep Q-Learning setup, 3 key hyperparameters and 1 model architecture, play an important role in the learning process, which are Bellman equation learning rate (α), Bellman equation reward discount (γ), learning rate for Adam optimization method of the Neural Network (*lr*), and neural network design. The base case hyperparameter settings are $\alpha = 0.1, \gamma = 0.99, lr = 0.0005$, and 3 layer

fully connected neural network (with 64 nodes in the 2 hidden layers) which allows agent to achieve target within reasonable number of training episodes (proximately 1200 episodes).

For all experiments, the maximal number of episodes is 2000, and the maximal number of steps for each episode is 1500 (just in case an episode never ends). When reward reaches target, another 20 episodes are run before training stops.

Note: *Figures 2 and 3 show rewards for each episode. Figures 5, 6, 7, and 8 show average rewards with rolling window 100 episodes, for better visualization.*

Base Case

Figure 2 shows that reward does not linearly increase with episode, but in a sigmoid fashion. For the first 500 episodes, the trend for rewards has little change: most of the rewards are around -200, with variance of ± 200. In some rare cases, the reward falls below -400. An accelerated increase of reward occurs between episode 500 and 800, where reward increases, on average, from -200 to 100. Rewards suffer some big losses around episode 900, where it drops back below -200 again. Starting from episode 1000, rewards seem to have little variation on average value, even though the variance still vacillate between 0 and 200. The last 200 episodes see a quickly decrease in variance, which allows reward to reach the target.

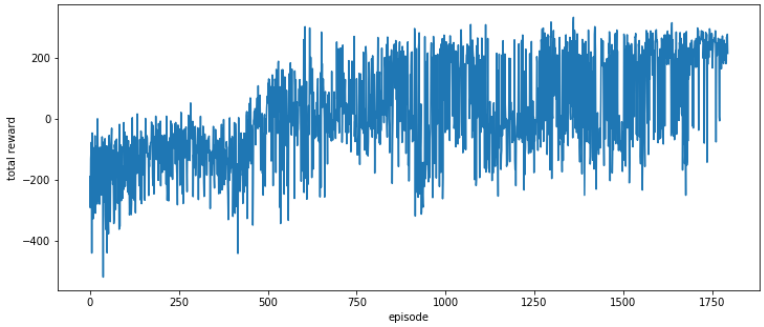
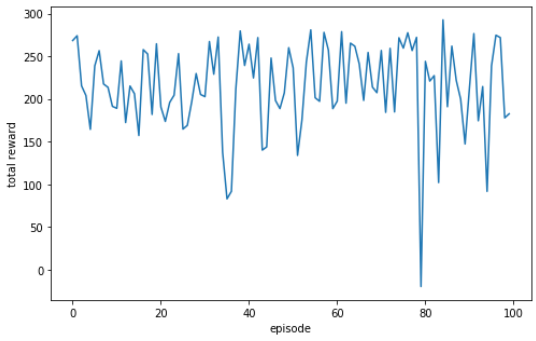


Figure 2. total reward for each episode during training process $\alpha = 0.1, \gamma = 0.0005, lr = 0.99$

The trained model from the above experiment is tested for 100 episodes, and the total rewards are shown in Figures 3 (reward per episode) and 4 (reward histogram). 65% of the episodes have a reward above 200, but still there are about 10 episodes with a reward below 150. A well-trained model only guarantees rewards on average are above the target, but for each individual episode, there is still possibility that the reward falls below 200, or even lower. In the last training episodes in Figure 2, there are still cases where rewards drop below 0, and there is also one case in Figure 3 where reward is below 0. Although further training could reduce the chance of low rewards, it demands much more computational time.



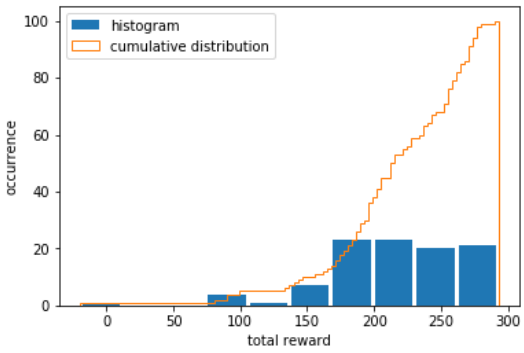
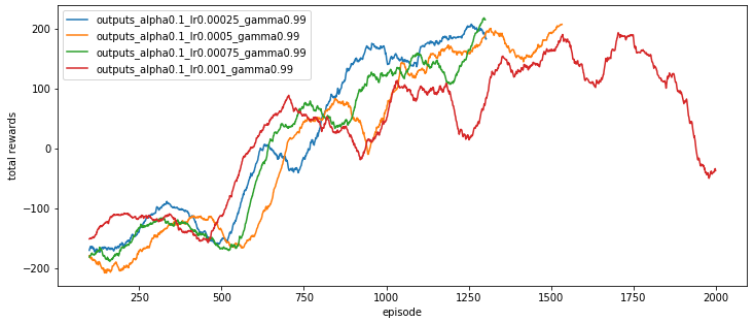


Figure 3. total reward for 100 consecutive episodes on a trained model

Figure 4. total reward distribution for 100 consecutive episodes

Effect of lr

This learning rate is used in Adam optimization of neural network. Higher learning rate facilitates a more aggressive update of the weights based on each batch of new inputs, so that the recent episodes will have more influence on model training. Although higher learning rate might expedite training process, it could also increase potential of divergence. A good choice of learning rate needs to balance training speed and capability to converge to global optima.



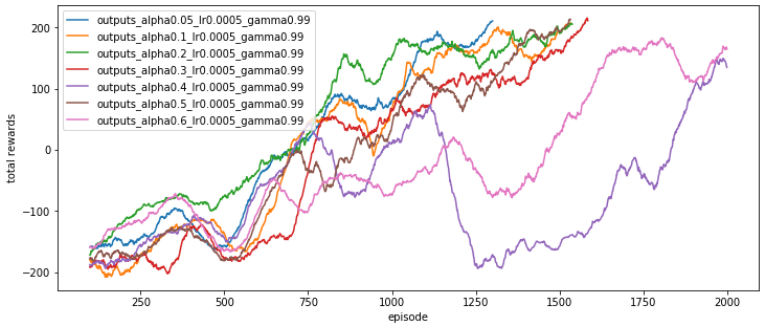
4 learning rates are tested in this experiment, and average reward history are shown in Figure 5. When $lr \leq 0.00075$, rewards all converge to the target. But reward never reaches the target for $lr = 0.001$ (the red curve). For that case, rewards deteriorate for the last 200 episodes.

Figure 5. total reward for each episode for 3 different γs

One interesting observation is that reward does not always reach the target with $lr = 0.00075$ (more thorough tests suggest that convergence/divergence is about 50/50). It implies that randomness plays an important role in model training at learning rate=0.00075. In current model architecture, randomness occurs at 2 steps: 1) action selection using epsilon-greedy method, and 2) input data selection for neural network training. Rewards for each step and episode is determined by whether good actions could be selected from a given state. If randomness leads to less preferable actions, rewards might never converge to the target. When *lr* is further increased to 0.001, the reward does not reach the target regardless how many experiments are performed. In this case, 0.00075 is the borderline learning rate between convergence and divergence.

Effect of α

Learning rate α in Bellman Equation controls the speed of Q value updates. A smaller α puts more weight in the old Q value estimate, so that the update relies more on the old value. On the other hand, a larger α allows more influence of new Q value. In theory Q-Learning always converge to the optimal solution, but larger learning rate could make model unstable and increate the number of episodes needed for training.



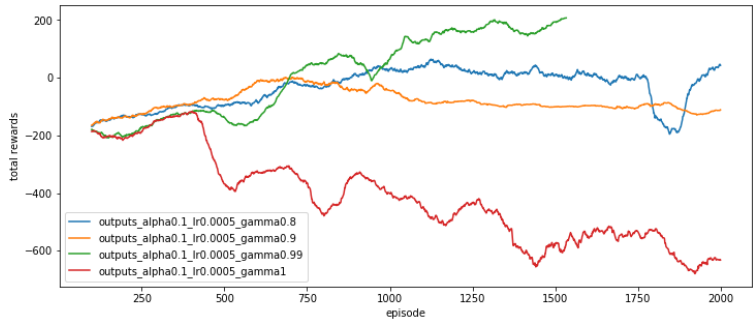
7 different αs are tested, with only rewards for 2 αs fail to reach the target within 2000 episodes. 5 αs are able to reach target. For this experiment, the borderline α appears t be around 0.4 to 0.5, where model with α below it can converge and above it will diverge. Note that 'diverge' means model fails to reach the target in 2000 episodes.

Figure 5. total reward for each episode for 7 different αs

For $\alpha = 0.1, 0.2, 0.3, 0.4$, models all reach the target at similar episodes around 1500. The model with $\alpha = 0.05$ converges much faster than others, which occurs at episode 1250. It shows that a smaller learning rate reduces the reward variance for any episode windows. As Figure 5 suggests, reward variances in a given window for $\alpha = 0.1, 0.2, 0.3, 0.4$ are in general larger than that of $\alpha = 0.05$, showing that smaller learning is quite effective to improve model stability. Such observation also holds true for cases with $\alpha = 0.5, 0.6$: the reward history for these two cases has the largest variance in any window and fails to reach target when 2000 episodes run out.

Effect of γ

Bellman equation reward discount factor γ determines how fast future reward is discounted to the current state. $\gamma = 1$ means there is no reward discount, and rewards in the infinite horizon has the same effect as the immediate reward, while $\gamma = 0$ means no rewards have impact other than the immediate one.



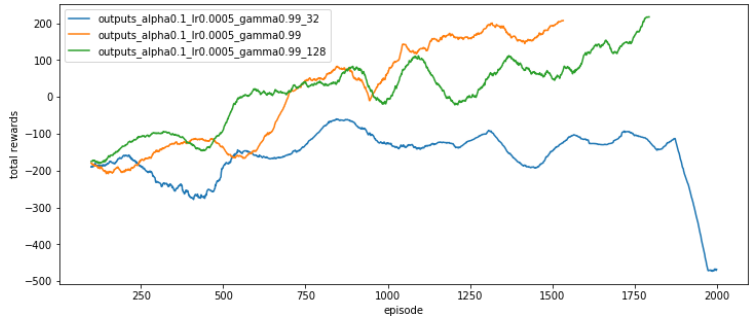
With 4 γ values tested in the experiments, only experiment using $\gamma = 0.99$ manages to converge. Both smaller (0.8, 0.9) and larger (1) γ values fail to converge. For $\gamma = 1$, the reward history shows a downward trend, showing that the agent only learns to deteriorate reward.

Figure 6. total reward for each episode for 3 different γs

γ values, except 1, all show upward trend between episode 500 to 800. The difference happens beyond episode 1000. For $\gamma = 0.8$ the average reward converges to ~ 0 and for $\gamma = 0.9$, it converges to -100. Both cases suggest that the model converges in a premature fashion.

Effect of neural network architecture

Another quite interesting factor that deserves study is the neural net architecture. The base case architecture is a 3-layer (including output layer) neural network, with 2 hidden layers of 64 fully connected nodes. For every training step, 64 data points are fed into the model as inputs.



In this experiment, two other neural network architectures are tested. To minimize effects from other factors, only the number of nodes in the hidden layers are changed. The number of input data points is set to be the same as the number of nodes.

Figure 8. total reward for each episode with 3 different neural network architectures

The number of nodes for each layer of neural network has a significant effect on model learning. More nodes (and along with that, more input data points) facilitate model convergence. With 32 nodes, model does not seem to learn anything for the first 1800 episodes and suffers from a negative learning for the

last 200 episodes. Both 64 and 128 node designs allow for model convergence. However, more nodes do not imply faster learning speed. As can be seen from Figure 8, it takes more episodes to learn for 128 nodes than 64 nodes.

It should be noted that comparison above is simple and does not employ any other potential influencing factors. For instance, it does not consider the case where number of nodes varies but input data points are fixed. As a result, it does not eliminate the possibility that the cause for divergence with 32 nodes is due to the number of input data points. This could be done in future study.

SUMMARY

In this project, a Deep Q-Learning model is developed to train the agent for 'LunarLander-v2' environment from OpenAI Gym. 3 hyper parameters, and deep neural network architecture, are investigated to learn their impact on model training.

It is found that the learning rate for Adam optimization method needs to be low to ensure model convergence. All models converge when $lr \leq 0.0005$, and could diverge when $lr = 0.00075$, and will diverge when $lr = 0.001$. 0.00075 is the borderline learning rate that separates divergence and convergence.

A similar borderline value is also observed for Bellman learning rate α, which has a value between 0.4 and 0.5. Models with α smaller than borderline value will converge and with α bigger than borderline value will diverge in 2000 episodes. This observation agrees with the purpose of learning rate, which is used to control model stability during training.

The model is very sensitive to the reward discount factor γ in Bellman equation. For all different γ values tested, only $\gamma = 0.99$ ensures that model converges. The model converges in a premature fashion with $\gamma = 0.8, 0.9$. Future study can focus on alternating epsilon decay function for slower decays, which might help model convergence.

Neural network architecture is another important factor for model training. More input data points and more fully connected nodes in each neural layer allows for better updates in weights, and therefore help model convergence.