CSci 5512 - Artificial Intelligence II

Jiafeng Li(li001029), Thomas Tang(tang0376)

Final Report

Introduction

The goal of this project is to design an agent that can play the lunar-lander game in OpenAI Gym, test the agent and find the best parameters in a certain range that fit the model best. Originally the plan was to implement Dynamic Bayesian Network (DBN), however, soon after researching on this problem, it became clear that DBN is not quite feasible for this project since we could not obtain a conditional probability table, and the calculation can be hard. Then, we discovered an algorithm called Deep-Q Network (DQN) is quite popular in this kind of problem. We learned the theory basis of this algorithm, and finally solved the problem with DQN. In the following parts, we’ll first briefly introduce what we learned about DQN, how we implemented DQN in Python, and parameter tuning of our model.

Abstract

DQN is actually implementing a Q-learning algorithm with a deep neural network. The basic idea of Q-learning is similar to the Bellman equation. In our video game case, we input states S and action space A, and through the equation Q(s,a)=r+gamma\*maxQ(s’,a’) we calculate the “quality” of each action, Q(s,a). “r” in the equation is the reward of current state, and gamma is a discount coefficient.

In DQN, we want to input state space S and action space A, and calculate the “quality” of action by neural network.

How to calculate Q(s,a) through a neural network? Here we can consider many algorithms, including one of the simplest, the feed forward network. The basic idea is that we want to construct a function mapping our input states to “quality” of actions. And the function is given as f=sigmoid(W\*I+B), where W is the weight matrix mapping each input to next hidden layer units, and I is the input matrix, B is the bias matrix. The sigmoid function is to make W\*I+B a number on [0,1].

So, the main idea is to output the value of each action (do nothing, left, right, up) through a neural network.

Python Implementation

There are many machine learning related packages in Python. We researched on several, include tensorflow, torch, Qlearner. We decided to use keras at the end, as it is a very commonly used toolkit. Open AI Gym has provided us a very friendly way to interact with the environment, we only need to use “env = gym.make('LunarLander-v2')” to load the environment. Our agent contains several parts: a neural network model, action choosing part, and replay.

After research, we found that a very general problem on action choosing is “Exploration-exploitation trade-off”. In Yu’s (2019) paper, he suggested that we should neither randomly choose next action nor totally follow the prediction of neural network model. Instead, we should find a way to balance. So we use a value epsilon in [0,1] to determine whether random choose an action or follow model’s prediction. Epsilon decays over iterations, (we set a decay rate) so gradually we will choose model’s prediction with higher probability. We believe that this method could be very efficient, so we used it in our code and later on we tested different epsilon decay rate to find the best one.

Another idea we found useful is called experience replay. We learned this idea in both Yu’s paper and video by Deeplizard (2018). The video introduced how to construct a buffer memory to store old experience and use it to train the model. Based on that we designed a replaymemory class to store old experience. However, we had no experience and cannot successfully implement experience replay, so we referred to Verma’s (2019) code and used the replay part.

Our python implementation could be found here <https://github.umn.edu/tang0376/ai2_project/blob/master/lunarlander.py>.

Agent Test

We tested on two parameters: number of units in each layer and epsilon decay rate, among certain sets of parameters, and found the best parameter in a certain range. (units number in range [25,500], epsilon decay rate in range [0.8,0.999]) Due to limited time, we didn’t test on other parameters like gamma and learning rate, we set them to be the best parameters based on Yu’s paper. We know that these parameters from Yu’s paper may not work well for our model, so testing on these parameters can be a future improvement.

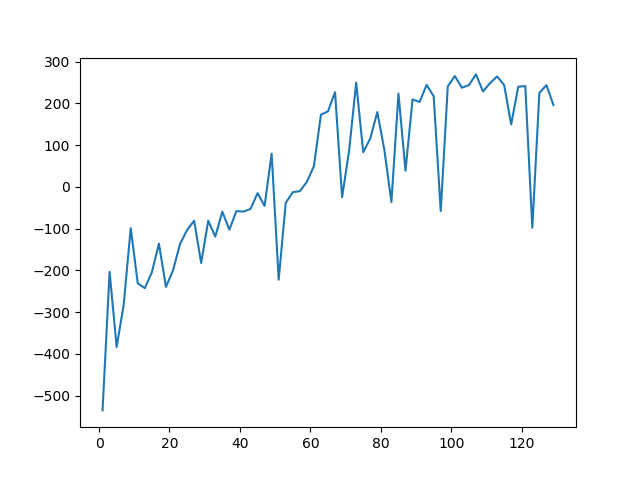
First we conducted test on number of units.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Units in 1st layer | Units in 2nd layer | Epsilon decay rate | Result |
| 1 | 50 | 25 | 0.996 | used more than 200 episodes and diverges, never reach our goal of 200 points |
| 2 | 500 | 250 | used **178** episodes |
| 3 | 200 | 100 | used more than 200 episodes |
| 4 | 200 | 200 | used more than 200 episodes |
| 5 | 150 | 75 | used more than 200 episodes |
| 6 | 250 | 125 | used more than 200 episodes |
| 7 | 750 | 375 | used more than 200 episodes |

We conducted test 7 based on test 1-6 to see if it is generally better to have a lot more units in each layer. Based on test 1-7, we found the best architecture of our two-layer neural network to be: 500 units in first layer, 250 in second layer.

Then, we use this structure to test epsilon decay rate.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Units in 1st layer | Units in 2nd layer | Epsilon decay rate | Result |
| prev | 500 | 250 | 0.996 | used 178episodes |
| 1 | 0.8 | used more than 200 episodes |
| 2 | 0.99 | used 155 episodes. |
| 3 | 0.993 | used **129** episodes.  (as figure indicated below) |



Thus, we found the best structure is: 500 units in first layer, 250 units in second layer, with epsilon decay rate of 0.993.

Conclusion

In this project, we implemented the lunar-lander game from OpenAI Gym. We designed and built an agent to play the lunar-lander game with DQN algorithm. We implemented our theory using keras, and we tuned the agent with the best parameter we tested. In the end, we successfully build a lunar-lander.

References

Deeplizard (2018). Training a Deep Q-Network - Reinforcement Learning. Retrieved from <https://www.youtube.com/watch?v=0bt0SjbS3xc>

Verma, S. (2019). Train Your Lunar-Lander | Reinforcement Learning | OpenAIGYM. Retrieved from <https://towardsdatascience.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197>

Verma, S. (2019). Retrieved from <https://github.com/shivaverma/OpenAIGym/blob/master/lunar-lander/discrete/lunar_lander.py>

Yu, X. (2019). Deep Q-Learning on Lunar Lander Game. Retrieved from <https://www.researchgate.net/publication/333145451_Deep_Q-Learning_on_Lunar_Lander_Game>