Lunar Lander Project

Chenxi Yu

Sunday, July 2, 2017

1. Introduction

DQL, as a deep reinforcement algorithm, has been tested successfully on playing Atari 2600 games, where the only input data are the pixels displayed on the screen and the score of the game. In particular, a deep convolutional network will be used to approximate the so-called optimal action-value function shown as below:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi\right]$$

During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s,a,r,s')\tilde{}U(D)$, drawn uniformly at random from the pool of stored samples.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2]$$

However, current methods used Deep networks to estimate Q suffer from instability or divergence for the following reason:

1. Correlation within sequence of observations
2. Small updates to Q can significantly change the policy, and thus the data distribution
3. The action values Q are correlated with the target values y = r_t + \gamma \max_{a'}Q(s', a')

Yet, the Lunar Lander problem is claimed to be solved by the DQL approach. Thus, I tried to replicate this approach to solve this problem. However, the challenge is to eliminate the correlation within sequence of observations and the possible corruption of the Q network due to small updates.

2. Experiment Overview

The problem is distributed from Openai Gym consisting of an 8 dimensional continuous state space and a discrete action space.
There are four discrete actions available: do nothing, fire the left orientation engine, fire the main engine,fire the right orientation engine.The total reward for moving from the t op of the screen to landing pad ranges from 100 - 140 points varying on lander placeme nt on the pad. If lander moves away from landing pad it is penalized the amount of rewa rd that would be gained by moving towards the pad. An episode finishes if the lander cr ashes or comes to rest, receiving additional 100 or +100 points respectively. Each leg gro und contact is worth +10 points. Firing main engine incurs a negative0.3 point penalty fo r each occurrence. Landing outside of the landing pad is possible. Fuel is infinite so that an agent could learn to fly (hover) before landing on the ground.

3. Hovering Check points

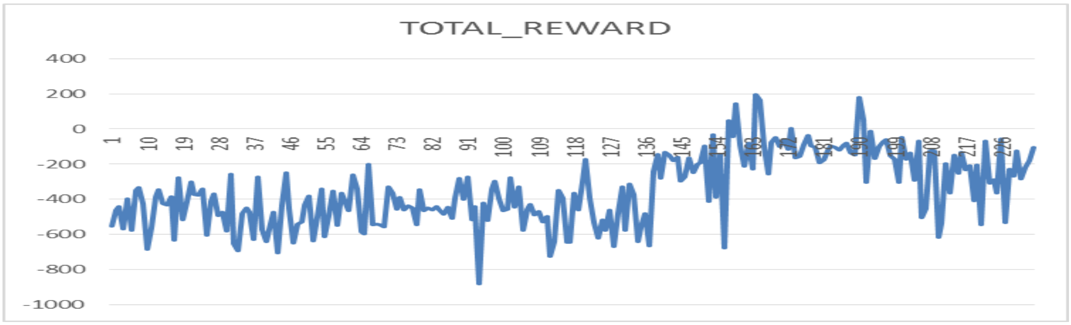
As mentioned, a general solution to the problem is to train it to hover before it lands. Practically speaking, this avoids the agent to crash in the ground since the main engine is expected not to fire. This calls for a non-linear function to land on the ground. The underlying model is relating actions to rewards, Q is estimated by a deep convolutional network and is updated by every step in time. My solution has achieved this bit as the reward stabilizes to -100 to -200 and video shows that the agent is hovering in the air all the time, while the loss function has stabilized in certain range until very occasional large value appears. And this is what I expect as a normal run where no small update upset the whole network presumably. Nevertheless, some combination of hyper parameters does break the network and I suspect the batch size and the weighing size has to do with this divergence. For example, if I have a batch size of 64 network with the 128 weights units size, I find the reward of the whole network increase dramatically after 700 episodes.

4. **Land on the pad**

This is considered as the most challenging task of this project. I could observe the reward during training has improved ever since. This improvement is dramatic when learning to hover yet not observed as a trivial task as it requires a convergence around 200 yet getting a positive reward has not been easy and I observe the curve for a long time. It is highly likely the agent's small learning rate (alpha =0.0005 or 0.001) causes such a problem. Nonetheless, a higher alpha will not help either since it may bring in a high mse in my experiments (some of them even reach 6 or 7 digits) thus requiring the agent to train for a longer time in each mini-batch. It is rather intuitive because if each time the network learn too much to have a consistency on gradual model improvement, it would just easily take in too much noise and diverge from the training target network. Thus I believe I should tune more on gamma and epsilon for a faster convergence while keeping a small learning rate i.e. alpha = 0.001. As the alpha is fixed, I would begin with discussing gamma, which should be carefully consider since it could easily hinder the agent's exploration based on the reward. As 0.95^100 has 0.006, this means that if I set my gamma way smaller than 0.9, after 100 steps, the Qmax is more trivial than the 0.95s effect. Thereby, I will keep a high gamma since it helps weighing in the effect of Qmax. Next, I would have a mild epsilon (=0.5) since my practice and peers tell that I should keep this level and then apply a decay function. I have tried the exponential decay, but I believe this approach would have the epsilon decay faster than it is expected. Since at the very beginning, if we expect a faster convergence, the model expects a higher chance to make a model prediction based on the network and thus enable the agent interact with the environment. After this step of exploitation, the agent could apply more random actions to explore the sample space. However, I expect more exploitation than exploration at the beginning, because this not only help a faster convergence but also help the network keep a lower loss rate. Basically if not enough exploitation is in place, more exploration mean nothing but noise to the agent. Thus, I replace my epsilon decay function by the uniform decay instead since it slows down the decaying of the epsilon and allows for a longer time in agent's exploitation.

Additionally, referring to the solution to Cartpole by Mr Morales in the video lecture, I have used Keras to build convolution nets and increase layers from 2 to 4. However, I could not observe any dramatic improvement from this process. But increasing the unit size from 32 to 128 does help the mse figure look normal as the model is trained. But I also adjust my batch size accordingly.

5. **Result**



I don't have enough time to get my project reach the target solution since it is not only tough to code things up but the parameters tuning is time consuming. However, I believe the overhead to proceed this project (with openai box2D and tensorflow) has been too high, especially for the window users like myself.
However, I will further discuss 2 of my observations: hovering and non-firing. I believe most of the time, hovering means the agent is still learning and possibly has us a convergence solution. However, this is essentially the struggle of this project, the training takes too long and I could only run everything based on my CPU. And I have the case that the agent was not converging or having the function loss explored. So I think if I could not overcome the shortcoming of DQN with other techniques like prioritized experience reply or tune perfectly on my parameters, this happens. And I also observed some occasions that my agent is not firing after some training time. I believe this is because some part of my network is biased toward that track and the exponential decay on the epsilon makes it worse.

6. Summary

In summary, DQN has helped extended my horizon thus fostering me to explore more in Deep Reinforcement learning. Yet, I think this leap to me has been too overwhelming, since I am not really too familiar with convolutional neural net, plus the challenge from the software configuration and my hardware issues. However, I really appreciate the chance to be introduced into the OpenAI and Keras. Yet, I would definitely need a lot more practice on Deep learning. Even though I was not able to make it through the experiment result, I find myself learn a lot in parameters tuning in DQN model construction and improvement.

Presentation link: https://www.dropbox.com/s/hhw1m497qa107a3/proj2.mp4?dl=0

Volodymyr, Mnih. "Human-level control through deep reinforcement learning." (2015): n. pag. Http://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf. Web.