EECE5870 Project: Immediate Deliverable One

Chi Nok Enoch Kan, Zachary Nordgren, Xiang Jin

October 24, 2019

Instructor: Prof. Richard Povinelli

1 Introduction

1.1 OpenAI Gym Environment

OpenAI gym is a package that can be used to develop and evaluate custom reinforcement learning agents. It comes with built-in control environments such as CartPole, MountainCar and many Atari games. Each gym environment contains an action space which represents all the possible actions that the agent can take. Environments also contain observation spaces, which contains frames that are of size $height \times width \times channels$. Every time an agent inputs an action an observation of the next game state and a corresponding reward is returned from the environment. The agent also receives additional information such as info and done, which contains diagnostic information and a boolean value denoting if the game is completed.

1.2 Breakout-v0

Breakout is a classic 2D game that runs on the Atari 2600 console and was released in 1976. The game requires players to bounce a ball of a movable paddle in order to destroy colored bricks. Players lose if the ball reaches the bottom of the screen and win if all bricks are destroyed. According to OpenAI's documentation, the Atari Breakout-v0 game has 4 possible actions from its action space: **noop**, **fire**, **right** and **left**. In this game **noop** results in no player action. **fire** initiates the ball into play either at the beginning of the game or after a life is lost. **right** and **left** are used to move the paddle right or left one step. Only one action can be performed at each time step.

1.3 Q Learning and Deep Q Networks

Before discussing what deep Q learning is, it is important to understand the concept of Q learning. In short, Q learning is a model-free reinforcement learning algorithm that approximates Q values given states S and actions A. Where the Q value is a variable which represents an estimate of the reward of an action or actions for a given state. Ideally we want to iteratively update the Q values given the weighted average of the old Q values, plus the current learned values.

Given the basic concepts of Q learning, deep Q learning is using deep neural networks to approximate the Q function. One additional procedure that deep Q learning incorporates is experience

replay. Experience replay is a powerful technique which works by randomly sampling the data distribution of past memories to train the network. This random sampling procedure has been found to speed up the agent's learning speed and reduce undesirable correlations. It is worth mentioning that the agent's memory e_t contains the following information $e_t = (s_t, a_t, r_t, s_{t+1})$, where s, a and r stand for state, action and reward at times t and $t+1$ respectively.

In this study we employ a recently developed Deep Q Network architecture called the dueling-Deep Q Network (dDQN). This architecture was first proposed by Wang et al. (2015), and involves a simple modification to the last layer of the deep Q network. Before the last aggregation layer the convoluted Q value is divided into value of a state ($V(s)$) and advantage of each action (A). Without going into the mathematical rationale behind this split, it has been found by the authors of the proposed network to stabilize the Q values and thereby speeding up training.

2 Methodology

We constructed an agent trained on dueling Deep Q Network using keras, an open-source Python library for neural network construction. We also used Huber Loss, a loss function used by Mnih et al. (2013), which is less sensitive to outliers than mean squared error. Parameters such as learning rate, minibatch size and image size were consistent with the parameters that Mnih. et. al. had previously used. We modify the number of episodes (denoted as epoch in our code) to 90000 in order to reduce training time. We also kept the same ϵ annealing schedule as the original paper, where they reduced ϵ gradually from 1 to 0.1 using a decay rate of $\frac{epsilon_{final} - epsilon_{init}}{step_{\epsilon}}$ during a phase we denote as the annealing phase.

In Mnih et al. (2015)'s other paper "Human-level control through deep reinforcement learning", they also suggested using a $no\ op_{max}$ hyperparameter. This hyperparameter controls how many times the agent can play the game without performing any actual actions. We simply play the game $no\ op_{max}$ times with integer 1, which denotes **fire** from the action space. This allows our agent to observe by doing nothing and simply restarting the game. The actual game begins after this observation phase and the aforementioned annealing phase.

3 Results

Due to time constraints, we were only able to run our agent for a total of 15000 epochs. The original Atari Breakout experiment done by Mnih et. al. ran a total of 100 epochs for training, where each training epoch contains 50000 minibatch updates. This adds up to around 5 million minibatch updates, or 50 hours of clock time. We were only able to perform around 60000 minibatch updates under 24 hours, assuming an average of 4 minibatch updates per epoch. Training is done on a single RTX 2080 Ti GPU without parallelization. Testing was done in a linux environment and was run on the same GPU. At each testing iteration, we record the score achieved by the bot as well as the high score from all the iterations. Our bot was able to achieve a high score of 24 over the course of 80 epochs. While this record is far from the current best record of 700+, we considered our training to be a success since there was clear evidence that the bot was able to learn. Given more time to run our dDQN bot, we expect the testing performance of our bot to continually increase.

One major takeaway from this study is that deep reinforcement learning is extremely slow and resource intensive. Training deep neural networks can be a difficult task even with advanced hardware and powerful GPUs. Training a dDQN agent can take days and is computationally

exhaustive. If we were to use evolutionary algorithms (EAs) to perform optimization of the dDQN hyperparameters- it would have taken us days, if not months, to complete training. Therefore, we suggest using a much simpler approach for the next deliverable of this project. We can also compare the performance of our EA-optimized agent to the dDQN agent we trained in this deliverable. We also learned that downsizing the outputted frames and converting them to grayscale had a tremendous effect on the training speed. In the future, we may be able to further downsize the images to $44 \times 44 \times 1$ to cut down training time. Of course, this is done at the expense of reducing the amount of information the dDQN can learn. If we were given a more powerful GPU, or even the possibility of parallelizing training on multiple GPUs, we could have experimented with increasing the number of frames in experience replay.

Finally, we learned at the completion of the experiment that Breakout-v0 might not be the best environment for training a dDQN. As suggested by Mnih et. al., Breakout-v0 ocassionally skips a couple frames, whereas Breakout-Deterministic-v4 does not or the RAM version. We wonder if skipping frames had a significant effect on the training and convergence speed of our agent, which could explain why the bot was not learning as fast as we expected.

4 Conclusion

In this study, we constructed a dueling DQN agent that was able to play the OpenAI Atari Breakout-v0. We trained the agent for a total of 15000 episodes over 24 hours, and our trained agent was able to achieve a high score of 24 during testing. Our work could be further extended to other deep Q networks such as the double DQN proposed by van Hasselt et al. (2015) and further optimized using optimality tightening proposed by He et al. (2016). Overall, we found dDQN to be a relatively slow reinforcement learning algorithm and was not suitable for optimization via evolutionary algorithm. We intend to use a simpler agent architecture for our next deliverable such as NEAT.

References

He, F. S., Liu, Y., Schwing, A. G., and Peng, J. 2016. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *CoRR*, abs/1611.01606.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. 2013. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

van Hasselt, H., Guez, A., and Silver, D. 2015. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461.

Wang, Z., de Freitas, N., and Lanctot, M. 2015. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581.