RL Assignment 3

Stephan van der Putten
s1528459

Murad Bozik
s2619822

Yuchi Zhang
s2724952

April 23, 2020

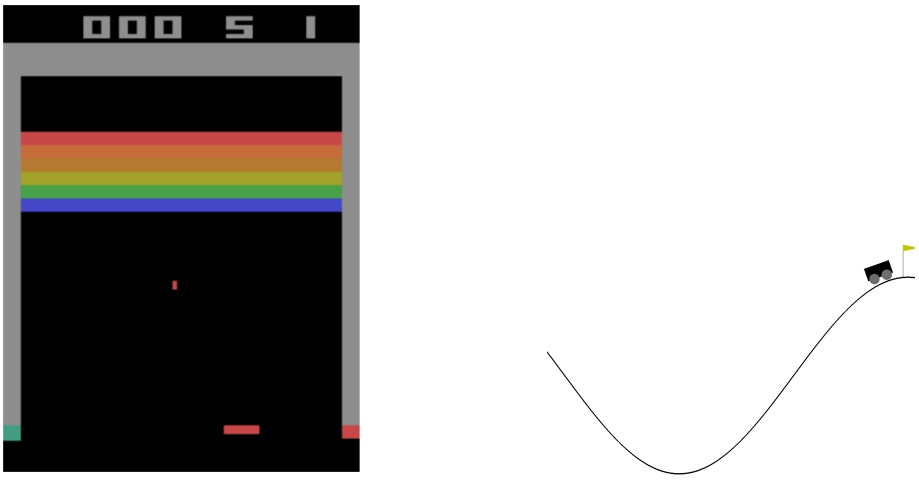


Figure 1: Breakout and Mountain Car

1 Introduction

In this assignment we build an intuition for deep reinforcement learning using neural networks. We will explore Mountain Car and Breakout environments. In the process of developing our models, we basically benefited from two books. We examined the developing process of deep reinforcement learning and deep Q learning from our lecture's main book[3]. And we improved our agents with the help of hints for breakout environment from Géron's practical text book[1].

2 Preliminaries

The Openai/Gym toolkit is used for game environments. We create models using the Keras library in combination with the Tensorflow backend. All code is written with Python 3.

2.1 General Approach

The basic difference between reinforcement learning and supervised learning is the type of learning. In supervised learning there are lots of labelled training examples. These examples can be used to train model and it can be really fast learning when compared with reinforcement learning. Reinforcement learning does not have labelled training examples. Training is based on rewards or losses which comes after an action. In this training process model/agent should wait for result of action and change his action according to result of previous actions. This is like a baby learning to walk. That's why this process can be slower than supervised learning. Also, since this learning process is an unstable process, many different learning methods have been developed. In order to solve we used Fixed Q-Value Targets model.

3 Mountain Car

Mountain Car is a simulation game in gym. A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; However, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. We train a DQN model to control the car to drive up the mountain on the right.

3.1 DQN Construction

We use Keras to build a DQN. Deep Q-learning Neural (DQN) Network is a neural network for Q-learning algorithm. In Q-learning algorithm, first of all, we create a reward table for each state and action. Secondly, we create a Q table, which represents the learned experience. It's at the same level as the reward table, and initialized to a 0 matrix, which represents the discount value of the total reward that can be obtained from one state to another. Then to compute the Q-value, we use the formula:

$$Q(s,a) = R(s,a) + \lambda * max(Q(s_{after}, a_{after})$$

As for DQN, we use a neural network to represent the Q-table since there are too many state-actions to store a Q-value.

However, we choose two tricks when we deal with DQN,

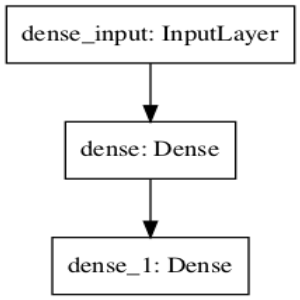
- Memory Sample
- Target Computing by Target_Network

For memory sampling, we record the history steps by a queue and put NN train on that queue. For Target Network, the Target Network is used to calculate the target. It has the same structure as the Q network, and the initial weights are the same, except that the Q network is updated each time the target network is updated every second. The goal of Target Network is to compute the loss by current experience, reward and Q network. This trick is done for convergence.

3.1.1 Implementation

We use the sequential Keras model to address the problem. We add two layers and we need to input the state (position, velocity) and output the (action0_value, action1_value, action2_value). Therefore, the input dim is 2. Also, we use 100 hidden layers empirically. We implement a same network as target network. Then we can train them. we use 'plot_model' to draw the model. Figure 2 shows the model. We choose 2000 episodes for the training and every 100 episodes we update the target network.

Figure 2: model



3.1.2 Results

Figure 3 shows the reward curve within a training process. We train our model twice, because there are random factors in the DQN training. Figure 3 shows that there is difference between the fluctuation of left curve and right curve. The reward of twice training process both converge on -100.

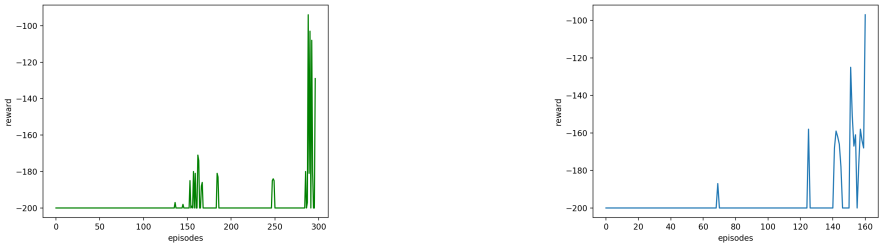


Figure 3: The rewards of the agent for two training processes

4 Breakout

4.1 Environment

As breakout environment we used *Breakout-v0* from OpenAI/Gym toolkit. In this environment the observation is an RGB image of the screen, which is an array of shape (210, 160, 3). Each action is repeatedly performed for a duration of k frames, where k is uniformly sampled from {2, 3, 4}. Since the observation is an image, this environment is called a discrete environment.

4.1.1 Implementation

The general structure of a DQN agent is similar to a Mountain Car agent we created before. We use a replay memory technique in order to reduce correlation between experiences in training. With this technique all experiences are kept in replay memory. Training examples are selected with a predefined batch size by sampling from this memory. This helps during the training process. Using just one model for predictions and training its weight again can cause a divergence, oscillations. This makes the Q-learning unstable. In order to eliminate this problem we implemented *Fixed Q-Value Targets* which is explained in Géron's textbook(page 639)[1]. In this method instead of one model two neural network model has been used. The second model, which is called target model, is used to compute Q-values for the next states. Target model is actually a clone of the main model. Target model's weights are updated at predefined intervals. To implement this method we counted each frame and passed it as a parameter to the train function.

Listing 1: Python example

```
def train(self, frame_num):
    ...
    if frame_num % self.update_interval == 0:
        self.target_model.set_weights(self.model.get_weights())
    ...
```

Also we used a variant of ϵ greedy method for balancing exploration/exploitation. We set epsilon to 1.0 and during training it gradually decreases until the minimum value 0.1. ϵ refers to the probability to select a random action. When ϵ equals to 0.1, agent will select an action randomly with 10% probability. ϵ greedy method for selection of actions brings the DQN between off-policy and on-policy learning.

4.2 Experiments

For breakout environment, we created two model and we tried them with different tuning parameters. Table 1 shows the structure of first model we created. Table 2 shows the model structure explained in the book[3].

Layer (type)	Filter size - Node	Kernel Size - Strides	Activation
conv1 (Conv2D)	32	8 - 4	Relu
conv2 (Conv2D)	64	4 - 2	Relu
conv2 (Conv2D)	64	3 - 1	Relu
flatten (Flatten)	-	-	-
fc1 (Dense)	512	-	Relu
fc2 (Dense)	4	-	None
Loss : Mean Squared Error Optimizer : Adam			

Table 1: Model-1 Structure

Layer (type)	Filter size - Node	Kernel Size - Strides	Activation
conv1 (Conv2D)	16	8 - 4	Relu
conv2 (Conv2D)	32	4 - 2	Relu
flatten (Flatten)	-	-	-
fc1 (Dense)	256	-	Relu
fc2 (Dense)	4	-	None
Loss : Mean Squared Error Optimizer : Adam			

Table 2: Model-2 Structure

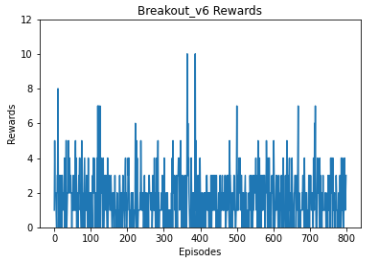
Experiments are named with their model and versions. In breakoutDQN_v5, we used Model-1 with following setup,

Parameter	Value	Parameter	Value
Episodes	500	Learning_Rate	1.0
Epsilon_start	1.0	Batch_size	64
Epsilon_min	0.1	Update_interval	50
Discount_rate	0.95	Memory_size	10000

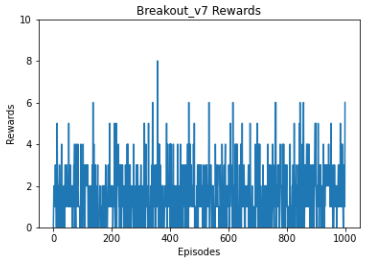
Table 3: Setup for breakoutDQN_v5

In the previous versions, we performed without frame skipping and with different learning rate, memory size and batch values (respectively 0.001, 2000, 32). In breakout_v6 and breakout_v7, we investigated the effect of epsilon greedy method. In breakout_v8, we applied the suggestions in the article[1]. Breakout_v8 includes the result of the DQN agent with suggested tuning. The experiments' results and trained models can be found in the models folder.

[1]https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756



(a) ϵ updated at the end of each training



(b) ϵ updated at the end of each episode

Figure 4: Effects of epsilon for different update times

4.3 Results

In breakout_v5, the cumulative reward could reach 6, while in the previous version it could reach 4. Frame skipping provides to add different frames to train the model. Otherwise the agent tries to learn from the images which are not very much different from each other. Later we used Model-2 with the same setups. Maximum cumulative reward reached only 8. This low degree of rewards pushed us to focus on epsilon method. We performed two different experiments. In breakout_v6 we updated epsilon value at the end of each training session. In breakout_v7 we updated epsilon value at the end of each episode. The purpose of these experiment was to identify if the agent explores only some part of the search space or not? Because, if the epsilon value drastically decreases in the first few episodes, later then it will 90% selects by predicting the Q-values. When we update at the end of each episode, epsilon value will decrease slowly and with higher probability the agent will select random action and it will force to explore much of search space. We performed 1000 episode for each agent. It did not provide reliable results, both agents stuck around 10 which is quite low for DQN agents. These results can be seen in Fig-4.

Since DQN agents perform unstable learning, it is quite hard to reach high rewards without proper parameter tuning. It is clearly seen in Fig-4. In breakout_v8, we applied and tried to see the difference of suggested tuning parameters.

5 Discussion

In this discussion we will talk about our report and what part could be improved. In this report we implemented deep Q neural network and tried to improve results by adding some variants of enhancements. Main goal for DQN learning is to reach a stable, smooth learning process. To achieve this goal many other enhancements has been made. Rainbow paper also showed that an agent which benefit from other enhanced agent can perform better than individual agents' performance[2]. In the rainbow paper, the authors combined 7 indepen-

dent enhancements to reach high performance score. These enhancements were independent and trained for maximizing general score individually. However, each agent has a different search state space. If the agent's search performance on state space can be monitored and tuned to work specifically on some part of search space; Then cumulative performance can be higher than an individually maximized agent's total performance.

6 Conclusion

DQN is implemented in Mountain Car. We can see the value of reward keep unchanging at the beginning when we train DQN model. And later on, with trying more times in the game and improvement of the Deep Q-learning table, reward converges on -100 sharply.

Breakout environment showed that tuning parameters are too important for Q learning algorithm. When we don't use a ϵ method to select a random action, the agent stuck and cannot explore search space. And if the epsilon is high and generally selects random action the agents may not reach high rewards because when the wrong action committed episode terminates and starts again. That makes Q learning unstable. Achieving stable learning also requires high amount of experimentation. That's why using GPU power is an essential part of Q learning.

6.1 Future Work

For future work, we can evaluate the arguments of DQN on fine-grained in these two games. We need to conduct a analysis on the impact of the different hyper-parameters. On the other hand, we could try other deep RL approaches. For example, Deep Deterministic Policy Gradient (DDPG) is based on the critic's method, which may get better performance in these two games, compared to DQN. In addition, DDPG is used to cooperate with the strategy function to directly output actions, which can respond to the output of continuous actions and large action space.

6.2 Acknowledgements

This report is based on the Reinforcement Learning graduate course at University Leiden. The program is created in combination with the knowledge gathered from the course book [3], the lectures, the help of the lecturer or teaching assistants.

References

[1] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, 2019.

[2] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] Aske Plaat. *Learning to play.* 2020.