Reinforcement learning Pong by Deep Q-network Method

zhenduo Li, leyuan Yang, haotian Wu

January 24, 2022

1 Introduction

Reinforcement learning has gained most concentrations these days The theory of reinforcement learning provides a normative account, deeply rooted in psychological and neuroscientific perspectives on animal behaviour. And our project focused on the Pong in Atari games which are the classic games for testing reinforcement learning algorithms. We aim to maximize our score in the Atari 2600 game Pong. In the preperation, we found a effective algorithm with highly well performances: the DQN algorithm. We summarize it in the next section.

Our report are arranged in the following content. In section 2, we summarize the main method and ideas of the DQN algorithm and we pointed out the important ones. In section 3, we give the environment that we used in our project. In section 4, we will give the exact results of our replication. In section 5, we discuss and conclude what we have learnt through this project. Our contribution is distributed in name order by 30%, 30%, and 40%.

2 Model setting

We consider tasks in which the agent interacts with an environment through a sequence of observations, actions and rewards. Our goal is to select a series of actions that maximizes the objective $Q(s, a)$. The DQN method consider using a deep convolutional neural network to approximate the optimal value function

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + ...|s_t = s, a_t = a, \pi] \tag{2.1}$$

with γ the discount number.

Let us explain the main purpose of doing so. Normally, in Q-learning, we usually utilize the ϵ-greedy algorithm to explore the $Q(s, a)$ value function. This method will be computationally low efficient when the dimension is extremely high and have no memory to store such statistics. However, in fact, most of the normal problems will encounter this dimension problem by using the mentioned algorithm. We have to contract the dimension of the state. The author proposed to simulate a value function approximation, which can be done by the deep neural network, which will solve the problems. Then we can just output a vector $[Q(s, a_1), ..., Q(s, a_t)]$ in a lower dimension. And our training object function, i.e. loss function is defined as follows:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D}\left[\left(r + \gamma \max_{A'} Q(s', a'; \theta_i') - Q(s, a; \theta_i\right)^2\right] \tag{2.2}$$

We use the gradient descent method to realize this step.

In the origin paper, the author consider a four-floor neural network with two convolution floor and two fully connected floor. They got tremendously well result in the Atari games compared with the Best linear learner.

We firstly recall the DQN algorithm as follows:

Algorithm 1 deep Q-learning with experience replay

Input: Initialize replay memory D to capacity N, action-value function Q with random weights θ, target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

for *episode* $= 1$ to M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1$
  for $t = 1$ to T **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1}$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1}$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1}$ from D
    Set $\gamma = \begin{cases} r_t & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1,a';\theta^-}) & otherwise \end{cases}$
    Perform a gradient descent step on $(\gamma_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

This is the original DQN algorithm in the paper. In next section, we will state how we replicate this algorithm in code.

3 Environment

python	3.8.5
torch	1.10.1+cu113
gym	0.21.0
ale-py	0.7.3

The game environment is PongNoFrameskip-v4.
One who scores 21 first is the winner.

4 Methods

4.1 DQN

4.1.1 Model Introduction

For complex problems, we use the function approximation method to build a Deep Q-learning network to estimate the value of (state, action). The input is a frame of image in Atari game, and the output is the value of each action. We can easily choose the optimal action because the number of actions is very small. The network consists of three convolution layers and two full connection layers.

We use a trick called Frame stack which fits the four recent frames together as one state. A single frame is (84, 84, 1), the composition of four frames is (84, 84, 4). Frame stack can provide temporary information and accelerate the exploration.

We compute temporal difference loss as the loss function, which is from Bellman Equation:

$$TDLoss = R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

Because the convergence of deep neural network is very slow and requires a lot of samples, it will be very inefficient to train the network only according to the environmental interaction. Therefore, DQN uses a memory buffer for memory replay, which is to save the previous experience of interacting with the environment and reuse it during training. Memory buffer mainly implements two functions: push stores the experience and sample takes out the experience for training.

4.1.2 Results

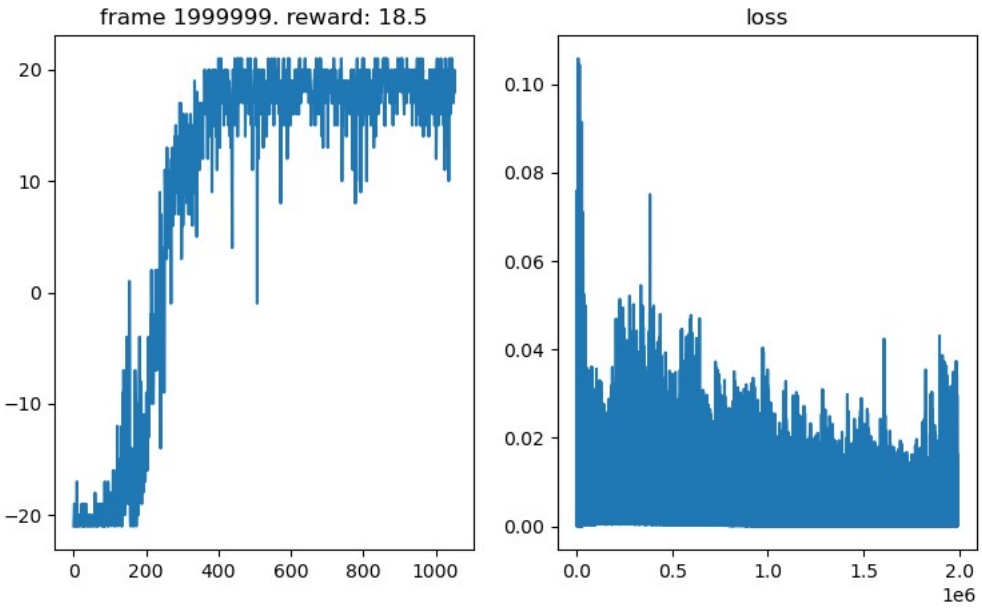

Figure 1: DQN 2000000frames

4.1.3 Comments

From Figure1 we can see the average reward is almost 20 after 400 episodes which is about 800000 frames. After 400 episodes, although there are some small fluctuations with the reward, the model performs well most of whose reward achieves 15. In terms of Loss, it keeps low which means our model performs well.

When we test the trained model, it scores 21:0 almost every time. We give the screen recording results of the game test in the attachment. For more game test results, you only need to run the corresponding test.py file to get them, which will not be repeated here.

4.2 DDQN

4.2.1 Model Introduction

Double-DQN is a further change based on nature DQN method.Let's revisit the Q-value update formula

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

For the general strategy, we have $v_\pi(s) \leq \max_a q_\pi(s, a)$, and for the optimal V-value function that follows the optimal policy, according to the Bellman equation, we have $v_*(s) \leq \max_a q_*(s, a)$. Since we are now approximating the Q-value function, the approximation error will keep accumulating during the max operation, eventually leading to the overestimation problem. To solve this problem, we we restore the target Q-value $R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$ to the action selection-action evaluation two processes, and let them use two different Q-networks $R_{t+1} + \gamma Q(S_{t+1}, arg\max_a Q(S_{t+1}, a))$. That is to say, we use the network to select the action, and then the network is used to calculate the target Q value. There is this structure called Double-DQN.

4.2.2 Results

Using the same parameters as the above DQN model algorithm, we train the neural grid to obtain the following reward and loss results.

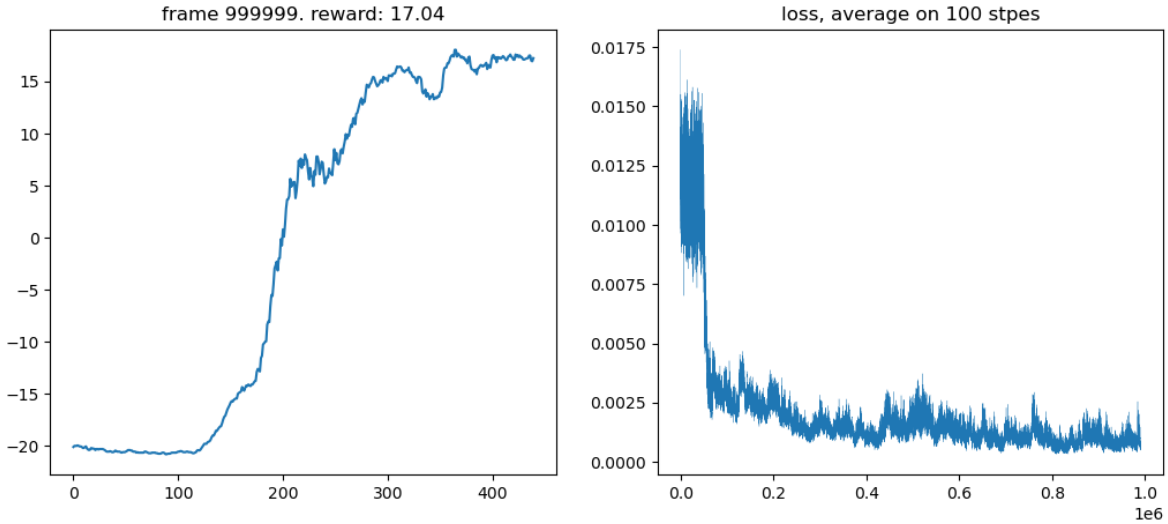

Figure 2: DDQN 1000000frames

4.2.3 Comments

From Figure2 we can see the average reward is almost 17 after 400 episodes which is about 800000 frames. After 400 episodes, although there are some small fluctuations with the reward, the model performs well most of whose reward achieves 15. In terms of Loss, it keeps low which means our model performs well, and compared with the DQN algorithm, its loss is basically an order of magnitude lower.

When we test the trained model, it scores 21:0 almost every time. We give the screen recording results of the game test in the attachment. For more game test results, you only need to run the corresponding test.py file to get them, which will not be repeated here.

5 Conclusion

In this section, we simply conclude our replication of DQN algorithm above. We can just come to the conclusion that we did a good job. The reward is extremely high after some training and the mean reward is up to 18.5. What's more, both methods' loss is very low and we can even say that they are just zero after several steps. All of us did learn a lot through this project.