Deep Q Learning on Atari Game - Breakout

Kung Hung Lu, Po Yen Hsu
Department of Electrical Engineering
University of Washington

Abstract

Reinforcement learning is critical for problems in which corrected data/label pairs are never presented instead the focus is on performance. It exploits trial and error strategy to interact with environment to learn the knowledge or skills. In this project, we show the deep reinforcement learning has a good ability to learn how to play the game Breakout. Without knowing the knowledge or rules of the game, the agent can develop an optimal strategy purely by the high-dimensional sensory data. We uses a convolutional neural network to approximate a reward function, Q-function to predict the longterm reward to guide the decision making process and demonstrate that it is able to achieve human-level performance. Furthermore, We also exploit the target network and double Q network to alleviate the slow convergence problem. The experiment results show that the training process indeed speed up with these improvements.

1. Introduction

Reinforcement learning is helpful when we try to make machine perform a task but there is no explicit correct label to guide it. For example, how could program a robot to grab a glass without spilling water. It is impossible to program each move as the situation would be different every time. On the contrary, it should learn to make a sequence decisions under uncertainty in order to reach the goal. According to that, Reinforcement is inspired by animal behavior, of how agents may optimize their control to an environment to maximize some notions of cumulative reward. In this project, we tried to study the power of reinforcement learning by letting the software agent successfully learn to play the Atari game - breakout. Using Machine Learning algorithms on actual game playing has been broadly researched and discussed due to the important and valuable information we obtain from such studies. Observations on how Artificial Intelligence compared to that of a human player provides patterns and knowledge on how to revise and modify such ML models toward future applications.

Figure 1: Screenshots of the game - Breakout

Breakout is a game in which the player tries to control a paddle to bounce the ball upward to keep it alive as long as possible. In the game, many layers of bricks lines the top third of the screen. A ball travels across the screen and bounces off the top and side walls on the screen. But the ball falls out from the bottom of screen and it causes the player losses a turn. When a brick is hit, the ball bounces back and the brick is destroyed. Therefore, the player must follow the movement of the ball to control the paddle to move left or right to prevent the ball touches the bottom of the screen as well as make ball hit the bricks. The score is measured by how many bricks the ball hits and once all the bricks are destroyed, another set of bricks will appear again. So in order to get a high score, the player must keep the ball alive for as long as possible. Figure 1 shows an example screenshot of breakout.

The main challenge for this project is that we only provide pixel information and the score to the agent. The agent does not understand the rule of the game, the appearance of the bricks even the location of the paddle and the ball. That is, it needs to learn these representation and concepts from scratch and be able to generalize as the state space is very huge. We exploit a convolutional neural network to approximate a total reward function. The total reward function

contains two parts: immediate reward which comes from the current environment and the future reward which is expected to get in the near future based on the current state. In each iteration, agent gets an observation(game screen)and a reward from game environment and then feed them to the Q-network model to choose a optimal action. Then, the agent apply this action to the environment to get an new observation and reward. The whole process is executed recursively. In training, once getting new observation, the agent would store this transition information which includes current state, action, reward and next state into the experience reply memory. Next, we randomly sample previous transitions in a small batch from experience reply memory. The network is trained with the deep Q-learning algorithm, with stochastic gradient descent to update the weight. Furthermore, to overcome the slow convergence problem, we also add two modules: target network and double Q-network to the basic deep Q-Learning method. The experiment result demonstrated these strategies indeed accelerate the training process and reach the human-level performance on Breakout game.

2. Related Work

Many studies have applied reinforcement learning to train an agent in learn control policies, since reinforcement learning theory normatively provides a rooted psychological and neuroscientific aspect similar to human behavior [4]. A well-known study that using reinforcement learning successfully is TD-gammon [5], which predicts the state-value function instead of the action-value function. Most reinforcement learning studies have focused on linear function approximations as in [6]. Subsequently, others have also made studies in combining deep learning with reinforcement learning [3], which uses restricted Boltzmann machines for value function estimations.

3. Approach

3.1. MDP Formulation

The **actions** in breakout includes doing nothing (a=0), firing ball (a=1), moving left(a=2) and moving right(a=3). The **state** is represented by a sequence of frames from the game environment. In our implementation, we stack continuous 4 frames as the states. The reason for stacking multiple frames rather than storing a single frame is because the agents needs temporal information to get the idea about movement of ball. For instance, the agent cannot deduce the direction of the ball from a single static frame, but movement is critical for making a decision. The **rewards** from GYM environment is limited to be 1 for all positive rewards and 1 for all negative rewards, leaving 0 rewards unchanged. Since the scale of scores variesfrom game to game, we follows the original DeepMind paper for this step to makes it easier to use the same learning rate across multiple games.

$$\{h_t, y_t\} = f(h_{t-1}, x_t). \tag{1}$$

3.2. Q-learning

The goal for the agent is to select an action that maximizes the upcoming future rewards. The optimal action-value function Q(s, a) defined by the Bellman function is,

$$Q(s,a) = r + \gamma \max_a(Q(s', a')). \tag{2}$$

where we try to take an action from the current state that optimizes the reward function, r refers to the immediate reward where is a discount factor that we assume the future rewards are discounted by. The property behind this equation is that if the optimal value Q(s, a) of a certain state s was known for all possible actions a, then the agent achieves optimal policies by selecting the maximum action at each state s as

$$\pi(s) = \arg\max_a(Q(s,a)). \tag{3}$$

In practice, this approach is unachievable due to the fact that the action-value function is separately estimated for each sequence, so in actual implementation, we use a function approximator to estimate the action-value function $Q(s, a; \theta) = Q(s, a)$. In contrast to studies we mention that uses a linear function approximator, our reference study uses the neural network as our non-linear function approximator. The here refers to each weights within the Q-network. By applying stochastic gradient descent to minimize the loss function

$$L(\theta) = E[(Y - Q(s, a; \theta))^2]. \tag{4}$$

To avoid issues caused by correlated data and non-stationary distributions, we also use the experience replay mechanism that smoothens the training of past behaviors through randomly selecting samples, the approach also uses a - greedy policy which is an off-policy that selects a random action with probability The Deep Q-learning with experience replay algorithm is shown as the following:

The agents experience e_t by each time-step is stored in a replay memory data-set D, Within the inner loop of the algorithm, we draw mini-batch updates randomly. After this experience replay the agent executes an action based on the ϵ - greedy policy.

3.3. Pre-processing

To reduce the computational complexity, a basic image preprocessing step is applied, we first convert the direct raw Atari image frame to gray scale image. Second, we downsample it from 210 to 160 pixels to a 84 x 84 pixel image. It is then normalized from [0, 255] to [0, 1]. Finally, we stack four consecutive frames as our input as shown in Figure 1.

Deep Q-learning with experience replay ;
Initialize D (size N), Q(random weights);
for *episode 1 to M* **do**
  Initialize $s_1 = \{x_1, x_1, x_1, x_1\}$;
  for *t = 1 to T* **do**
    random generate p from (0,1) ;
    if $p < \epsilon$ **then**
      select random action a_t from action space;
    else
      select action a_t through
      $\pi(s) = \arg\max_a(Q(s, a))$;
    end
    execute action a_t and observe reward r_t and frame x_{t+1} ;
    set $s_{t+1} = \{s_t[1:3], x_{t+1}\}$;
    store $e_t = (s_t, a_t, r_t, s_{t+1})$ in D ;
    sample random mini-batches of e_i from D ;
    if *game is over* **then**
      $y_i = r_i$;
    else
      $y_i = r_i + \gamma Q(s_{i+1}, a_{i+1})$;
    end
    Perform SGD on $(y_i - Q(s_i, a_i))^2$
  end
end

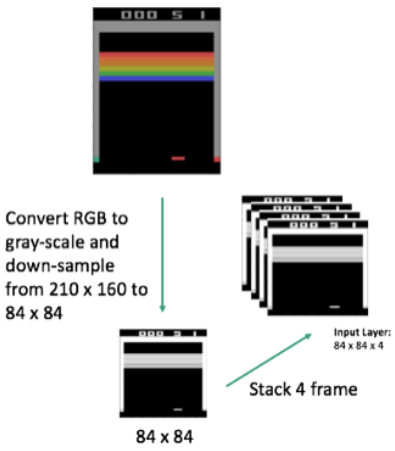

Figure 2: Preprocessing

3.4. Deep Q-Network

Our Q-function is approximated by a convolutional neural network(CNN). The structure of our CNN has four layers and is illustrated in 3. The first layer is a convolution layer with 16 filters of size 8×8 with stride 4, followed by a RELU layer. The second convolution layer has 32 filters of size 4×4 with stride 2 follwed by another RELU layer. Then, it is followed by a fully connected layer with 256 neurons and the output layer with outputs of each valid action spaces.

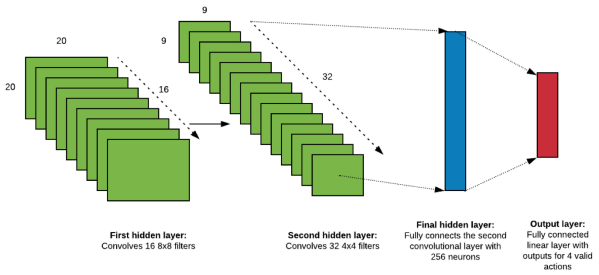

Figure 3: The architecture of Q-netwrok

3.5. Improvement

In our pilot study and training result, we noticed a few improvements to make, firstly, the convergence rate is relatively slow, this issue is causes by the fact that as we train the optimal action-value function, it is shown to be a recursive function, so both sides of the equation with oscillate causing a slow convergence rate. Secondly, we noticed that the action values are tended to be overestimated due to the upward bias each time we choose a maximum action value from the optimal function. To alleviate the first issue, we use a target network proposed in [2]. Within a given iteration, we fix the network Q as the target network and use to generate updates, with this modification, it adds a delay between the time updates to Q, which fastens the convergence rate and the oscillations are less likely to occur. For the second issue, [7] attempts to use a Double DQN to revise the model, and the results are shown to decouple the upward bias be using this Double DQN. In this approach, we use two Q-functions Q1 and Q2, one function is used to determine the maximum action, and we use the second function as the optimal action value function to estimate its value. In the papers experimental results, it shows that the Q-values are decorrelation in a sense and cancels out the overestimations.

4. Results

We provide a demo video at the following link: https://youtu.be/N1YoBr4PRVA. The evaluation metrics for the perfomance of the DQN is the reward score and the duration of the game which means how many turns does the agents can survive with the limited 5 life.

4.1. Setting

There are several hyper-parameters need to set in advance. The discounted factor γ is set to 0.99. The exploration probability ϵ decreased from 1 to 0.1 over 1 million updates to the DQN. The size of experience reply memory is 10000 instance.

Table 1: Statistics of our photo critique captioning dataset.

Aspect	# photos	# sentences	# words
General Impression	4123	12908	237337
Composition & perspective	4000	12848	262194
Color & Lighting	3769	9384	168028
Subject of photo	3812	8022	129179
Depth of field	3017	4864	86391
Focus	2994	5421	89626
Use of camera, exposure	3396	8255	156721
Total	4235(union)	61702	1129476

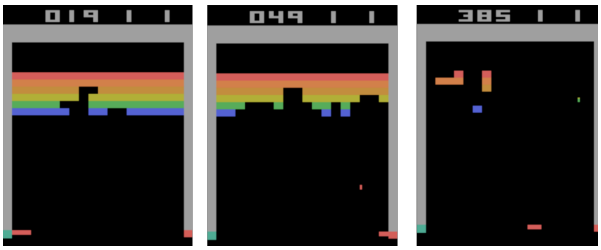

Figure 4: The screenshots of the performance from different models. From left to right, the agents is trained under 30000, 90000 and 150 million iterations.

We implement this project using Keras with tensorflow as backend. For training, we use RMSProp with a learning rate of 0.0001 and momentum as 0.99. These setting is chosen according to original DeepMind papers [1] and [2]. We tuned some of these parameter in the experiments to get reasonable result because the different deep learning framework and weight initialization method we used. Before the training, we let the agent to choose random action to fill the replay memory first. After that, the model is updated in mini-batches of size 32. We update the target network once for every 10000 updates by copy the current Q-network to it. The whole training process is run on a desktop with 2 Nvidia Titan XP GPU.

4.2. Qualitative Analysis

The trained Deep Q-learning agents plays quite well and even perform better than humans in a limited training time. We compare the final result which is trained about 1.75 million episode to the baseline model. The baseline model is a purely randomly control strategy agent which will randomly select to move left or right in each iteration. This is also the policy for our model in the very beginning before training state in which the goal is to explore the possible states and to store them into experience replay memory. In Figure 4, we can find that the score showed on the top of screen is increasing through training process. With about 150 million updates, the model can almost clear all the bricks, which has a huge different from the random model. The more detailed example can be found through the video link provided earlier.

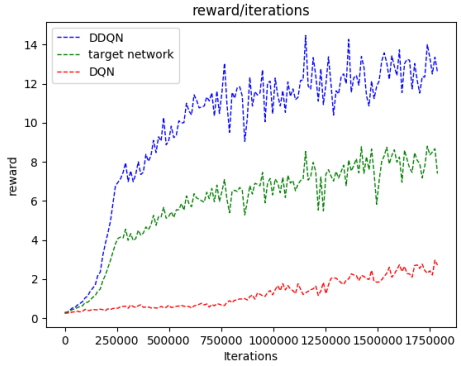

Figure 5: Reward comparison

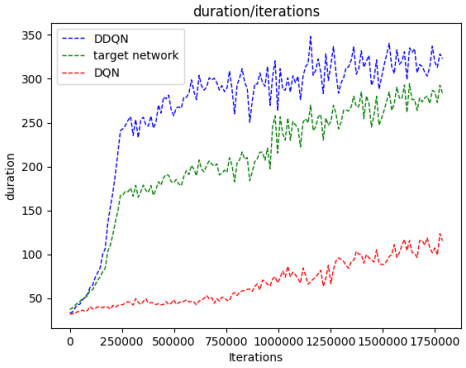

Figure 6: Duration comparison

4.3. Quantitative Analysis

In this study, we started with a basic DQN model and subsequently improved the model by adding a Target Network and Double Q-Network. The basic DQN model has issues referring to slow convergence rate and overestimation on q-values. By the revised improvements, we can find that the speed of convergences has a lot of improvement according to the following figures while we fixed all the other hyper-parameters including learning rate, gamma etc. As shown in the Figure 5 and 6, we can found that the original deep Q-learning approach exists the slow convergence problem. The reward indeed increases but in a very slow pace. However, after we add target network, it becomes much stable while training so that the speed of convergence is improved. The results is quite similar to the original paper [2] which said that because we use the same network to do both predicting future reward and optimize the current q-function, the training is too unstable to reach the convergence. Furthermore, we also try to add the double Q network from [7] and it even reach the better training results.

5. Conclusion

We successfully trained a agents to plat Breakout game with deep Q learning algorithm. Without explicit program the step by steps for the agent, we can make the agent reach almost the same human-level control by only providing pixels and score information. The agent has the capability to learn straight from the low level information to get the high level concepts about how to play the game. Moreover, we overcome the slow convergence problem in our earlier study through adding target network and double Q network tectonics. Both the qualitative and quantitative results demonstrated the effectiveness of trained agent on Breakout game.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 4

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 3, 4

[3] B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5(Aug):1063–1088, 2004. 2

[4] H. Shteingart and Y. Loewenstein. Reinforcement learning and human behavior. *Current Opinion in Neurobiology*, 25:93–98, 2014. 2

[5] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995. 2

[6] J. N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997. 2

[7] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016. 3, 4