Replay Experience Prioritized DQN Model - A Performance Measurement in Atari Breakout Game

Bofu Dong, Yanzheng Wu, Luxi Liu

Abstract

We implemented Q-learning and Double Deep Q-network with prioritized experience replay. The DQN model is a convolutional neural network, whose input is raw pixels and whose output is a value function estimating future reward. We apply Q-learning to the Minigrid environment and DQN to a even more complicated Atari Breakout environment. We assess on the performance of Q-learning and DQN in terms of training time, training difficulty and the rewards from environment.

1 Introduction

Controlling agents directly from high-dimensional inputs like vision and text is one of the challenges of RL. Most successful RL relied on hand-crafted features with linear value functions or policies. However, this limit the types of input of agent, and make solving problem with vision or text inputs extremely difficult.

Thanks to the advances in deep learning, the extraction of high-level features was made possible. In both computer vision[4] and speech recognition[1], some breakthroughs were made. It seems natural to combine RL with deep learning method.

However, RL presents several challenges in a deep learning perspective. First, enormous training data is usually needed for a deep learning algorithm, but RL agents need to interact with environment to obtain data, and this is usually limited. On the other hand, RL agents must be able to learn from a scalar reward that is sparse, noisy and often delayed. Another issue is that most deep learning algorithm requires data samples to be independent, while in RL data is typically highly correlated. Furthermore, RL data distribution usally changes as the agent learns new behaviours, but deep learning methods assume fixed distributions.

The above challenges were first addressed by the invention of Deep Q-learning network (DQN)[6]. The model is a convolutional neural network that can overcome above mention challenges and learn a successful control policies from video data in complex RL environments. The network is trained with a variant of the Q-learning algorithm, with stochastic gradient descent to update

the weights. The experience replay[5] mechanism that can randomly sample previous transitions was applied to alleviated correlated data and non-stationary distributions problem.

2 Background Related Work

The combination of RL with Deep Neural Network was proved to be successful in many fields, including but not limited to robotics, video games, natural language processing, computer vision, education, transportation, finance and healthcare.[2]

In many practical decision making problems, the state space of Markov Decision Process (MDP) are high dimensional. This means, that with deep learning, Reinforcement Learning is able to solve more complicated tasks with lower prior knowledge because of its ability to learn different levels of abstractions from data.

In a complex problem, deep reinforcement learning replaces tabular methods of estimating state values with function approximation. Function approximation not only eliminates the need to store all state-value pairs, and also enables agent to generalize the state-value pairs that are unseen.

2.1 Experience replay

Experience replay[5] is a technique that allows agent to 'mentally experience' the effects of its actions without actually executing them, and is shown to be effective in reducing the number of action executions required.

Experience replay is used not only to improve the efficiency of network, but is also needed for the independent and identically distributed assumption of neural networks, because a naive Q-learning algorithm is often strongly affected by the correlation between the sequence of experiences during learning.

2.2 Target network

The fact that Q-Learning updates a guess with a guess could potentially cause wrong correlations, we want to make the training of our neural network more stable by using target network.

This idea comes from the fact that one can interleave acting and learning process in experience replay. Target network is a technique that we keep a copy of our neural network and use it for $Q(S', A')$ value - the predicted Qvalues of this second Q-network is called target network, and are used to back-propagate to the main Q-network.[7] The use or DQN and a target network is sometimes referred to as Double DQN.

3 Methodology

3.1 Q-Learning for Minigrid

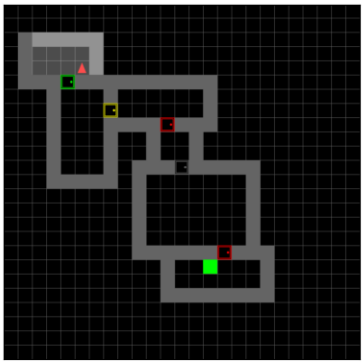

Figure 1: Minigrid Environment

To begin, a randomly generated map is provided with multiple rooms; two rooms can be connected by a closed door. The agent is randomly placed in the first room and the goal is at the last room. At each step, the agent tries to cross the rooms to reach the goal, selecting from the action set {TurnLeft, TurnRight, Forward, Toggle}. Note that at each door, the door is originally closed, and the toggle action is needed to open the door. Q-Learning will try to learn the state-action values for positions on the map. In each step, we reach either the goal (1 reward), or not the goal (0 reward). We choose to not discount rewards. At each state S, we take epsilon greedy action A (a move), collect reward, goes to S', and update the Q value for state S action A.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 2: Q-Learning algorithm

3.2 Deep Q-Network for Atari Breakout

Figure 3: Atari Breakout Environment Screen Interface

Action Space	Discrete(18)
Observation Space	(210, 160, 3)
Observation High	255
Observation Low	0
Import	`gym.make("ALE/Breakout-v5")`

Figure 4: Atari Breakout Environment Details

Since we did not manage to fix version issues when trying to connect a deep neural network to the Minigrid environment, we implemented the Deep Q-Network on a different environment - Atari Breakout. The action set here is {No_Op, Fire, Left, Right}, where fire i.e. firing the ball only happens at the beginning of each episode. Here the neural network takes in information from the observation space, which is a 210 by 160 rectangle, and each position contains a pixel value (r, g, b). The deep Q-network learns a function that allows agents to generalize on Q-value.

We use a method called Experience Replay. Agent's experience at each timestep is stored and pooled over multiple episodes into a replay memory. This approach increases efficiency by incorporating each step of experience into potentially multiple updates, and decreases errors caused by the strong correlation between consecutive samples.

3.3 Detailed network structure for DQN

:

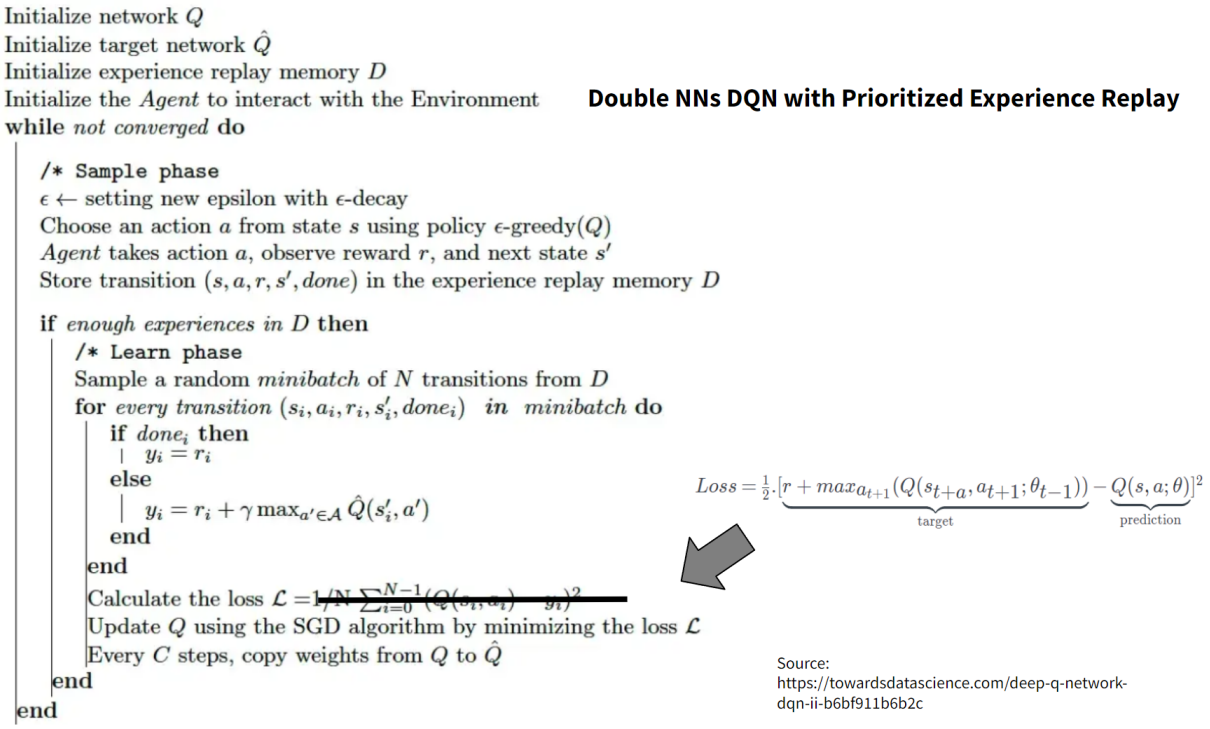

Figure 5: DQN algorithm with loss function modified

The original input, 210 x 160 x 3, is computationally demanding. We first turn the RGB image into gray-scale, reducing the number of channels from 3 to 1, then we sample down the rectangle into 84 x 84. Therefore the input after preprocessing becomes 84 x 84 x 1.

We set up four two-dimensional convoluted layers, the spec of which are as follows: 32 8x8 filters and stride 4, 64 4x4 filters with strides 2, 64 3x3 filters with strides 1, and 1024 7x7 filters with strides 1. ReLU is used as the activation function for all layers.

4 Experiments and Evaluation

4.1 Q-Learning for Minigrid

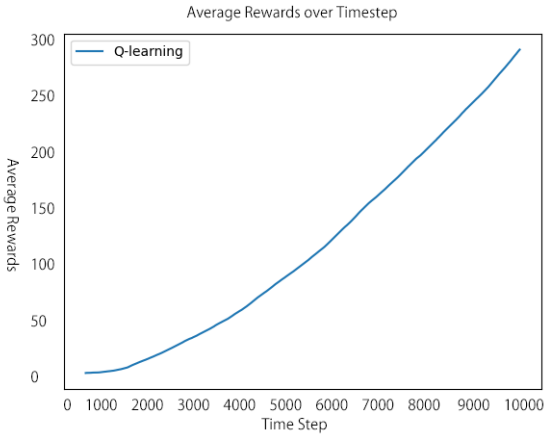

Figure 6: Q-Learning Reward

In our randomized experiments, the agent reaches the goal for the first time around 2000-3000 steps, after which it can better accumulate rewards. Since the set of state-action pairs is huge, this model is not very efficient: it needs to store a large table, and as mentioned, it doesn't reach the goal until 2000-3000 steps. The average number of finished episodes at 10,000 steps is 49.

Figure 7: Average reward on number of games in training

4.2 Deep Q-Network for Atari Breakout

In figure 7, it is obvious to see the overall increase in average reward during training. The sudden jump in the performance is due to the dynamic epsilon our model was using. In the beginning, the epsilon was big so the model was eagerly exploring, but as the network's parameters are learning, the epsilon decreases and relies more on the target NN's policy decision. At the end the agent reach a platform, and this is because the agent reach its limit in terms of deep learning theory. The gradient became so small that no more updates could be made to optimize next step. This could be improved by tuning hyperparameters, or use residual blocks in the network.

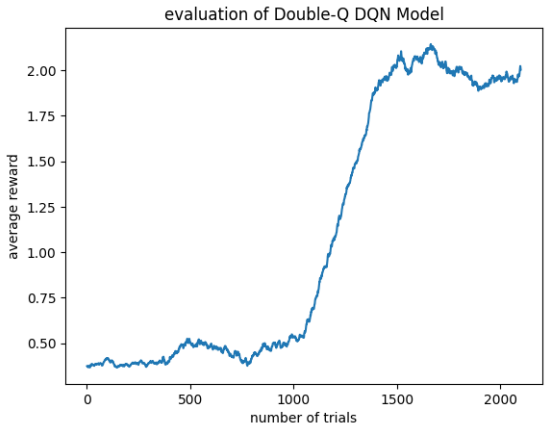

Figure 8: Average reward on number of trials for evaluation

The figure 8 is the average reward during evaluation of the model. The plot follows a very similar pattern with the figure 7, and this indicates that the overfitting problem is not likely to exist.

There are many possible ways to improve our current network on the problem. By training hyparparameter with more time, or use a dynamic learning rate, the current network could be further improve. Furthermore, using other extension of DQN network, the performance could be better. The Rainbow Q-network [3], for example, could obtain an even better result than our used Double DQN.

5 Conclusion and Future Work

Based on the experiments, we conclude the following:

- Double DQN more efficiently uses previous experience, by learning one experience multiple times.

- The performance also comes with a great cost: it is very hard to hyper tuning the DQN to see effective results.
- Overall, DQN is time costly to train. It usually takes hours and even days to see results. If one wants to further improve the network, decreasing learning rate is necessary, but this will increase the training time even more.
- It is difficult to find the appropriate time to stop training. The agent may experience 'catastrophic forgetting' when it is trained for too many epochs. It is possible that DQN remembers only success in the buffer, forgets what failure is, and then predicts everything with high values. Because of this, running the algorithm for more time, doesn't guarantee a better result.

The possible future explorations could be:

- Improving Double DQN with other deep learning techniques, such as noisy linear layers.
- Explore on the problem of catastrophic forgetting. One possible solution is adding memory, but this is not always possible for large DQN.
- Tackle multi-player environment. The solution is not as naive as giving the same algorithm to the second player, especially in an environment that players could interfere with each other. Introducing Shapley Q-value to DQN could be one solution to this

References

[1] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[2] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.*, 11:219–354, 2018.

[3] 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, 2017.

[4] Geoffrey E. Hinton and Volodymyr Mnih. Machine learning for aerial image labeling. 2013.

[5] Longxin Lin. Reinforcement learning for robots using neural networks. 1992.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.