Assignment 4

Name：Shiqu Wu

Student ID：518021910665

1 . DQN

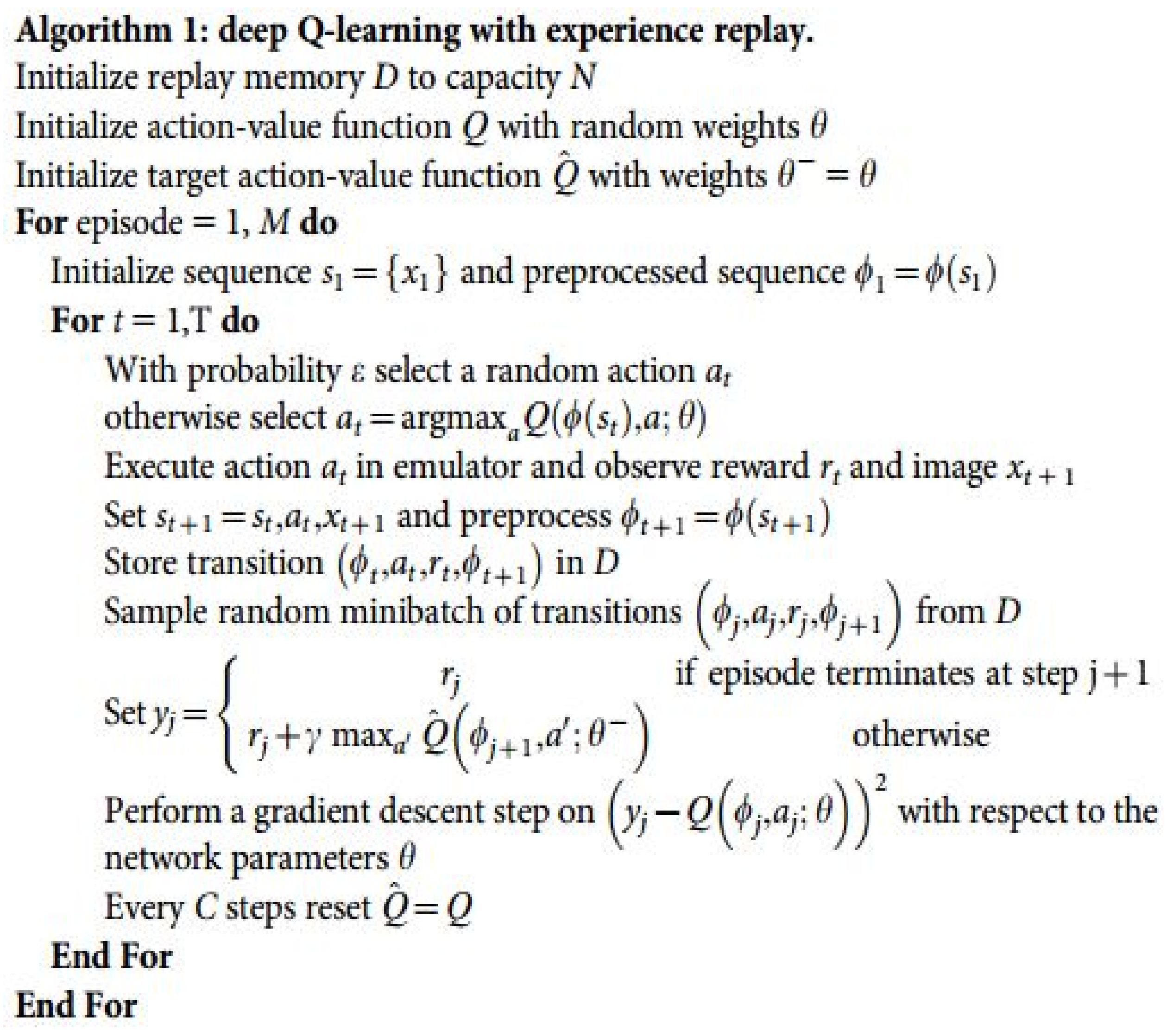
(1.1)

Why deep:

When action space, state space for a problem is big enough, it's impractical to use a explicit Q-table to store the policy or Q-value. For the fact that it needs tremendous memory to store a Q-table of such size. So scientists introduce Deep Neural Network (DNN) to solve this problem:

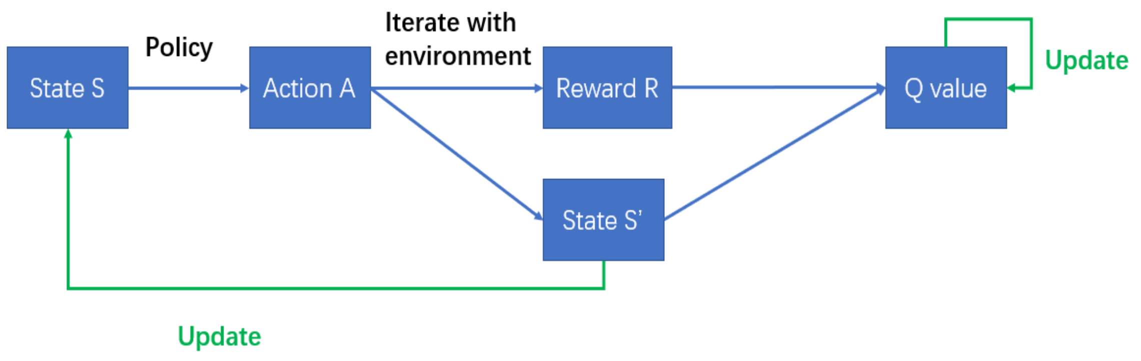
1. Construct a neural network, let its input be: S, output be: A, it can replace the $choose\_action()$ function in Q-learning.
2. Construct a neural network, let its input be: S, A, output be: Q, it can replace the $policy\_update()$ function in Q-learning.

DQN algorithm:



```
**Algorithm 1: deep Q-learning with experience replay.**
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**For** episode = 1, $M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    **For** $t = 1$,T **do**
        With probability $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
        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 $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    **End For**
**End For**
```

High level understanding of DQN process:



Core ideas of DQN

- **Experience Replay**

  Raw input data of reinforcement learning is high correlated and non-stationary, which cause the RL algorithm hard to converge. The essence of Experience Replay is to turn original **Online-Learning** into **Semi-Offline learning**. When we use memory to store the sample data and get a batch of data by random sampling, we can remove the high correlation among data. That's the key point of experience replay.

- **Fixed Q-target**

  As we've learned in the lecture, if the we use the same Q network during training, the possibility of divergence will increase. Because the target label is constantly changing (it's estimated by Q network), when we let the evaluated value to gradually approach the target value, **oscillation** is likely to occur. In machine learning training step, the label is known so we can assure that the training process let the estimated value updates and optimizes towards the fixed goal. So we benefit from this idea and put forward the Fixed Q-target structure that separates target network from Q-network.

(1.2)

Drawbacks of DQN:

1. **Upward bias problem:**

   We estimate q_target by:

$$q\_target_t = R_{t+1} + \gamma max_a\{\hat{Q}(S_{t+1}, a; \theta_t)\}$$

   For the fact that $\hat{Q}(S_{t+1}, a; \theta_t)$ is also a estimate value, so there is a certain error between it and the real value. However, $max\{\}$ policy improvement strategy will optimize neural network in the direction of maximum error. Therefore, DQN will suffer the problem of upward bias. That why we use **Double DQN** to alleviate this problem.

2. **Reward bias problem:**

Directly use the reward given by the environment to update the policy may have bias in some cases. For example, in self-driving game, there exists many **action-insensitive** states where the importance of Q value (calculated by action value function) is less than the V value (calculated by state value function). In these states, Q value is decided by state value and has little to do with action value. Thus, in these cases when we want to choose the best action through Q value, we'd better eliminate the effect of state value (only remain the effect of action value to choose the beset action). That the motivation of **Dueling Network**.

3. **Random sampling problem:**

   In many real work situations, there are very few positive samples and lots of negative samples, which is known as the unbalanced sample problem. Take Mountain Car setting as an example, in DQN memory, there is only one positive sample (received a positive reward when the car reached the flag) and the others are all negative sample (received a negative reward when the car didn't reach the flag) . So if we use the random sampling strategy, in most cases we will sample a batch of all negative samples which makes the model learn slowly. But if we sampling according to the **sample priority** p in memory, the model will learn more effectively in this situation.

   But how can we define the priority of the sample? It turns out that we can use **TD error**, to specify the degree of priority. If the TD error is large, it means that a lot of improvement is needed in correcting the prediction accuracy. That is to say, the more the sample needs to be learned. So we give the high TD error sample a high priority. This is the core idea of **Prioritized DQN**.

(1.3)

Advanced DQN:

Double DQN:

- Solve upward bias problem.
- Decrease the estimated target action value and make it more close to real target value.

Prioritized DQN:

- Solve random choose mini-batch problem.
- Increase the sample from another aspect and make the model independent of the influence of the state during the training process.

Dueling DQN:

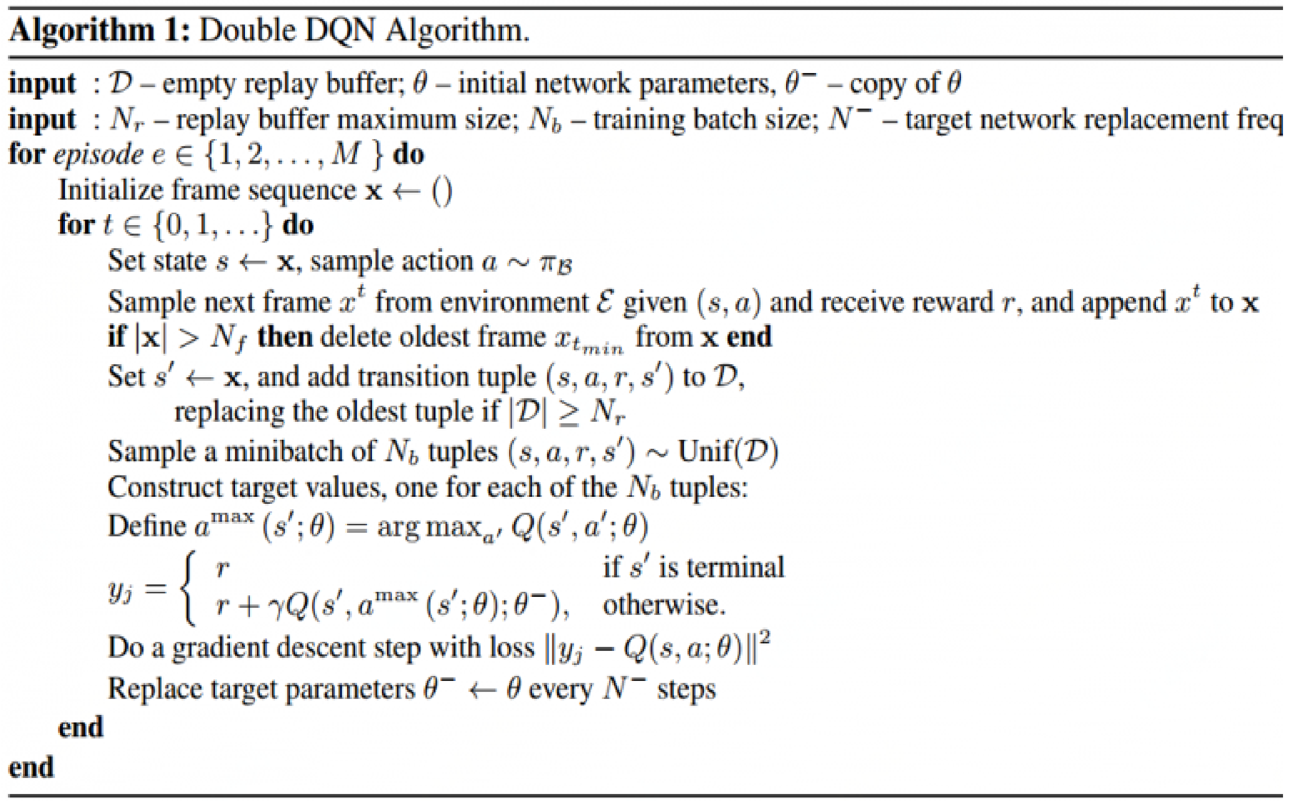
- Solve reward bias problem.
- Improve the efficiency of training by increasing the convergence speed.
- Decrease the model variance, thus increase the stability and generality of the model.

2 . Advanced DQN

(2.1)

Double DQN

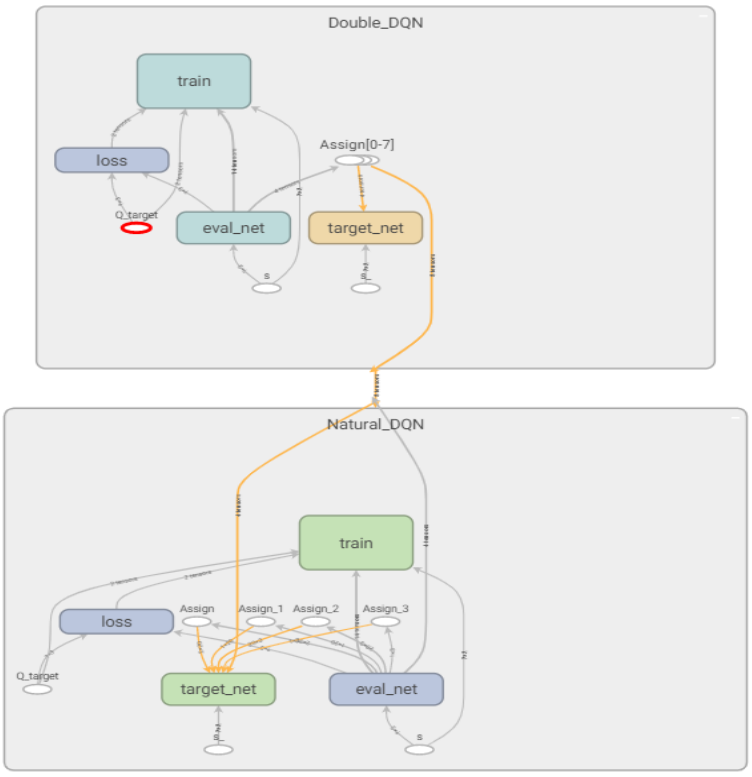
Algorithm:



Algorithm 1: Double DQN Algorithm.

```
input : $\mathcal{D}$ – empty replay buffer; $\theta$ – initial network parameters, $\theta^-$ – copy of $\theta$
input : $N_r$ – replay buffer maximum size; $N_b$ – training batch size; $N^-$ – target network replacement freq
for episode $e \in \{1, 2, \ldots, M\}$ do
    Initialize frame sequence $\mathbf{x} \leftarrow ()$
    for $t \in \{0, 1, \ldots\}$ do
        Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
        Sample next frame $x^t$ from environment $\mathcal{E}$ given $(s, a)$ and receive reward $r$, and append $x^t$ to $\mathbf{x}$
        if $|\mathbf{x}| > N_f$ then delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ end
        Set $s' \leftarrow \mathbf{x}$, and add transition tuple $(s, a, r, s')$ to $\mathcal{D}$,
            replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
        Sample a minibatch of $N_b$ tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
        Construct target values, one for each of the $N_b$ tuples:
        Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$
        $y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$
        Do a gradient descent step with loss $\| y_j - Q(s, a; \theta) \|^2$
        Replace target parameters $\theta^- \leftarrow \theta$ every $N^-$ steps
    end
end
```

Network Structure:



Improvements:

In natural DQN, we estimate q_target by:

$$q\_target_t = R_{t+1} + \gamma max_a\{\hat{Q}(S_{t+1}, a; \theta_t)\}$$

This will cause the upward bias problem. The main idea of Double DQN is to introduce another neural network to independently estimate the optimal action value then we can eliminate the $max\{\}$ operation. And we've known that in natural DQN we already have two separate network: $Q - eval$ network and $Q - target$ network. Thus, in Double DQN, we exploit this prior advantage and use $Q - eval$ network to calculate the optimal action value and directly use this action value as the chosen action in $Q - target$ network. In this way, we can avoid using $max\{\}$ operation, which will alleviate the upward bias. Here are the direct difference between natural DQN and double DQN:

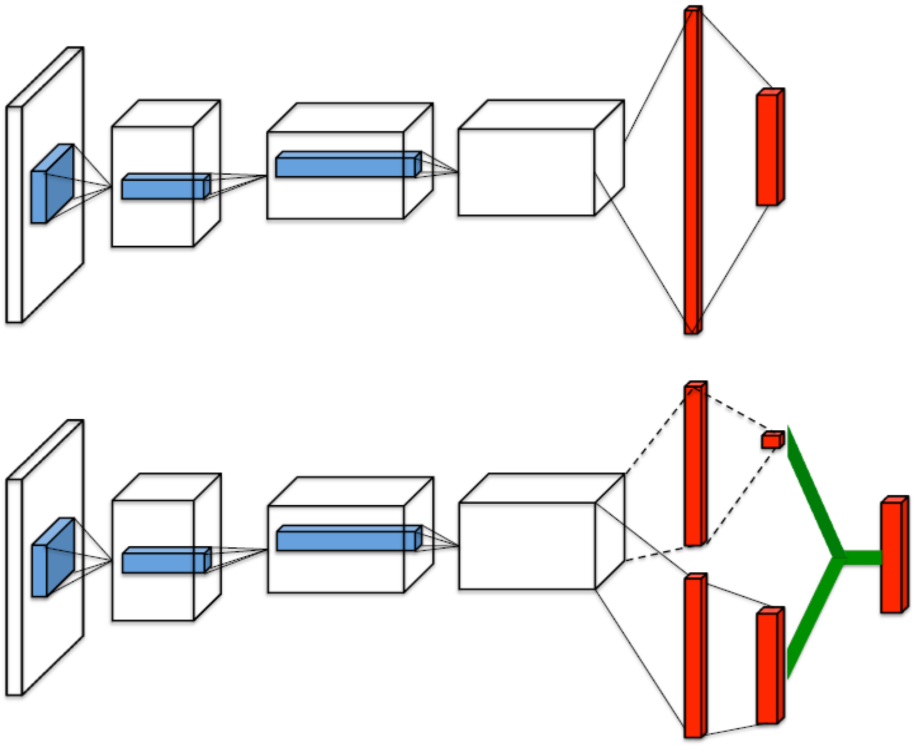
$$Y_t^{DQN} = R_{t+1} + \gamma max_a\{\hat{Q}(S_{t+1}, a; \theta_t)\}$$
$$Y_t^{DoubleDQN} = R_{t+1} + \gamma\hat{Q}(S_{t+1}, argmax_a\{Q(S_{t+1}, a; \theta_t)\}; \theta_t)$$

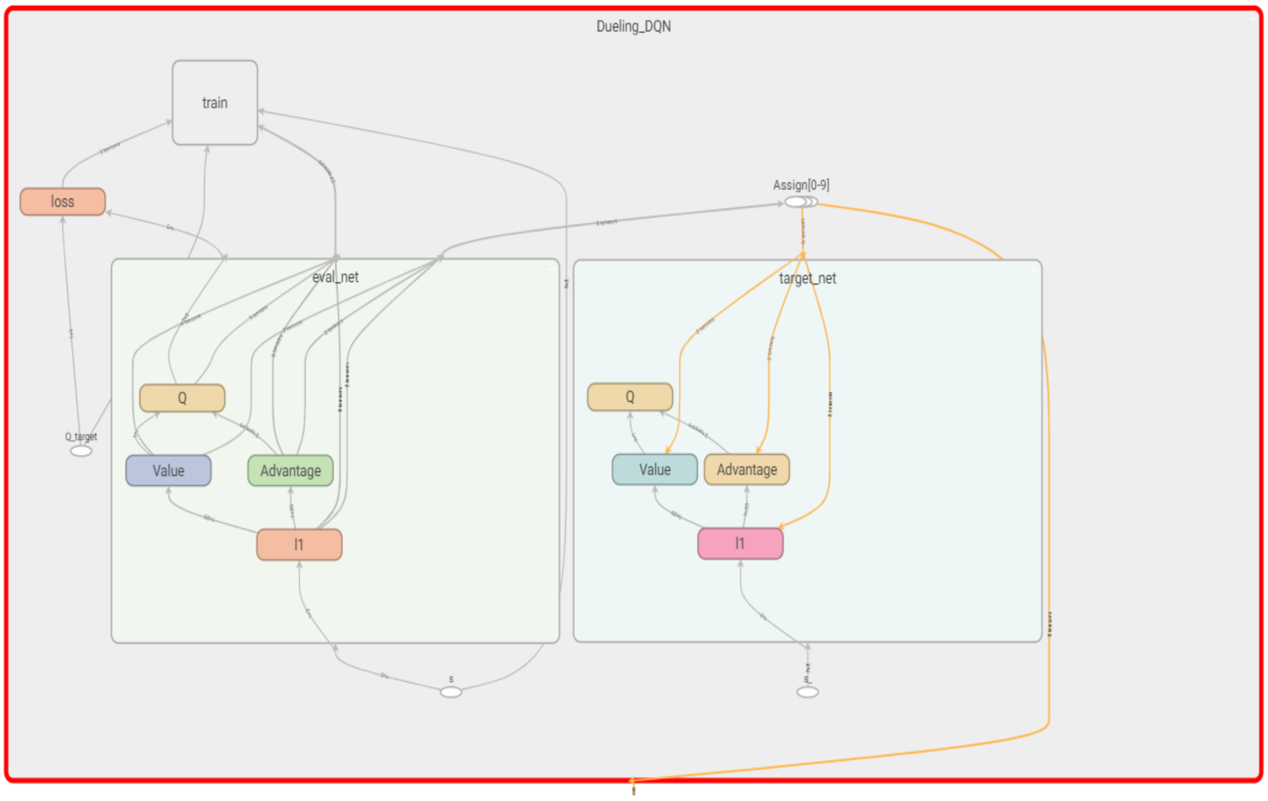
Note that $\hat{Q}(S_{t+1}, a; \theta_t)$ means $Q - target$ network and $Q(S_{t+1}, a; \theta_t)$ means $Q - eval$ network.

(2.2)

Dueling DQN

Network Structure:





Improvements:

Originally, DQN neural network directly output the Q-value for input actions, which makes the model suffer reward bias problem. For the fact that there exists many **action-insensitive** and when we want to choose the best action through Q value, we'd better eliminate the effect of state value. Therefore, in Dueling DQN, we divide Q-value into the following two parts:

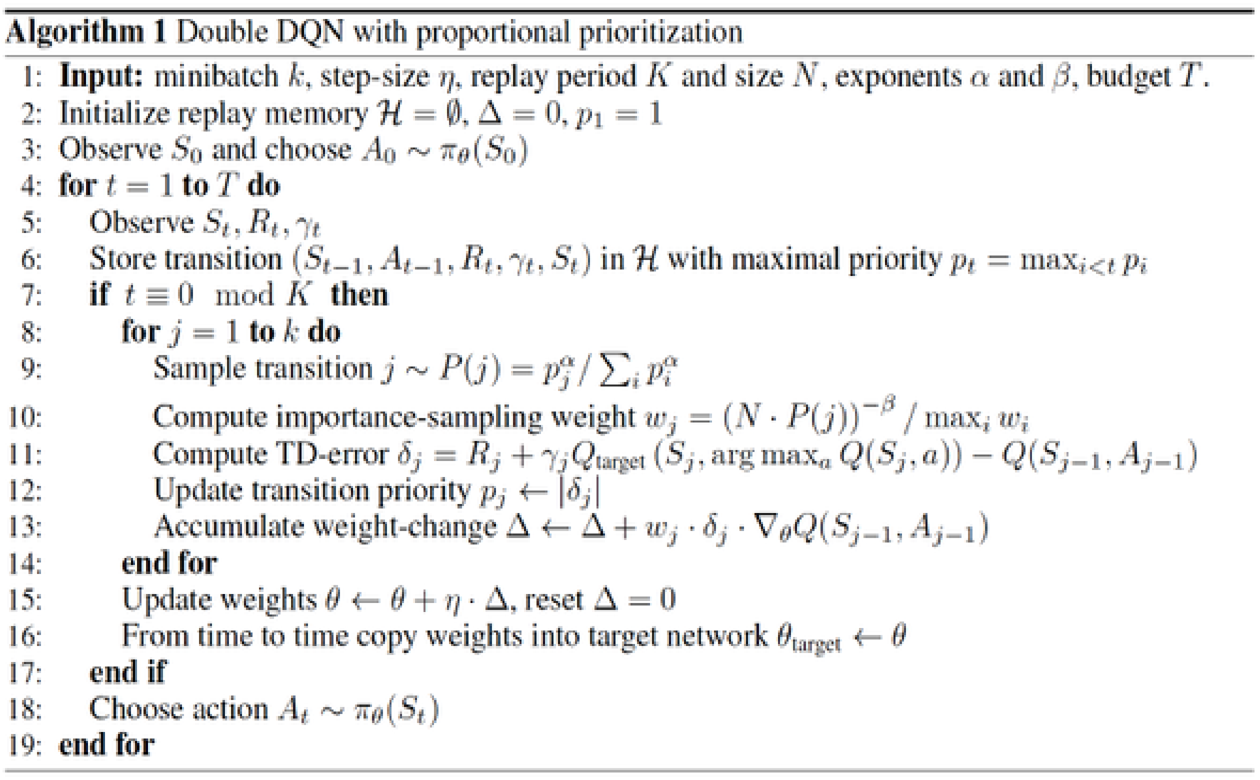
$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha), \qquad (7)$$

$V(s; \theta, \beta)$ is the state value, it's only affected by state s. While $A(s, a; \theta, \alpha)$ is the advantage and it represents the effect of action a. From the network structure we can see that $V(s; \theta, \beta)$ is a scalar value and $A(s, a; \theta, \alpha)$ is the vector that has the same shape with $Q(S_{t+1}, a; \theta_t)$.

(2.3)

Prioritized DQN

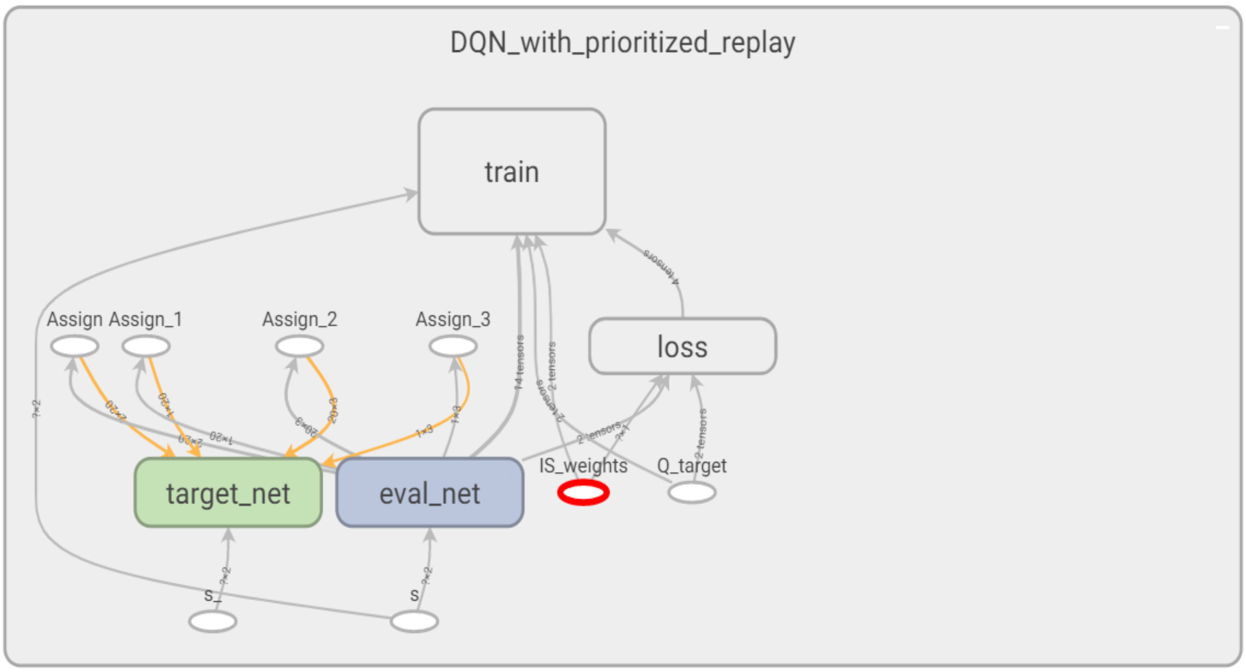
Algorithm:



Algorithm 1 Double DQN with proportional prioritization

```
Input: minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and $\beta$, budget $T$.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ to $T$ do
    Observe $S_t, R_t, \gamma_t$
    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ then
        for $j = 1$ to $k$ do
            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
            Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
            Update transition priority $p_j \leftarrow |\delta_j|$
            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
        end for
        Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
        From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
    end if
    Choose action $A_t \sim \pi_\theta(S_t)$
end for
```

Network Structure:



Improvements:

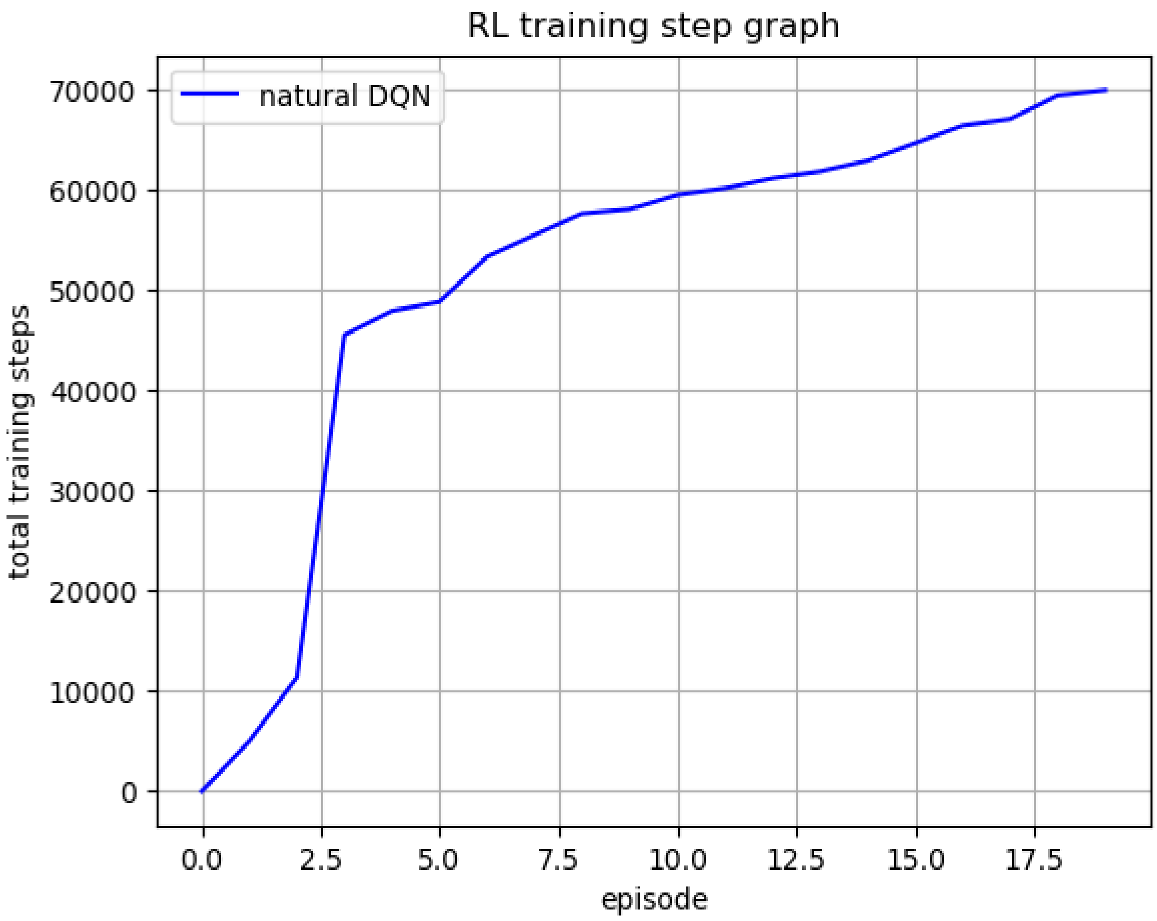
From the comparison of network structure graph we can easily see the difference between Prioritized DQN and natural DQN is the "**IS_weights**", which is highlighted by red circle. IS_weights measures the priority of samples in DQN memory and it's decided by TD error. Larger the TD error, larger the IS_weights. For the fact that when the TD error is large, it means that a lot of improvement is needed in correcting the prediction accuracy. That is to say, the more the sample needs to be learned. So we give the high TD error sample a high priority (i.e. large IS_weights).

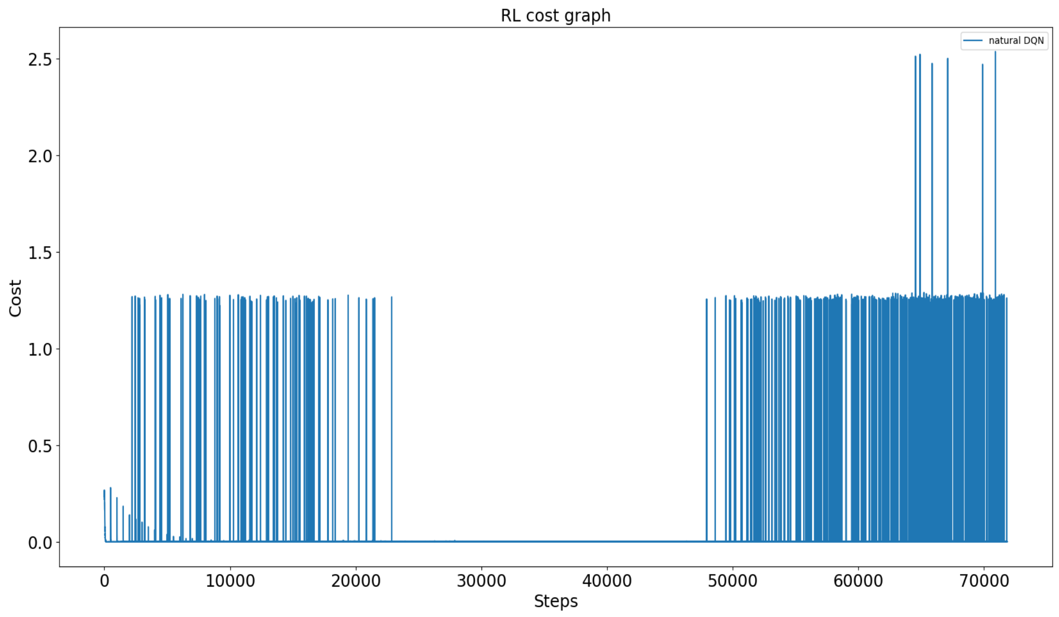
3 . Experiment

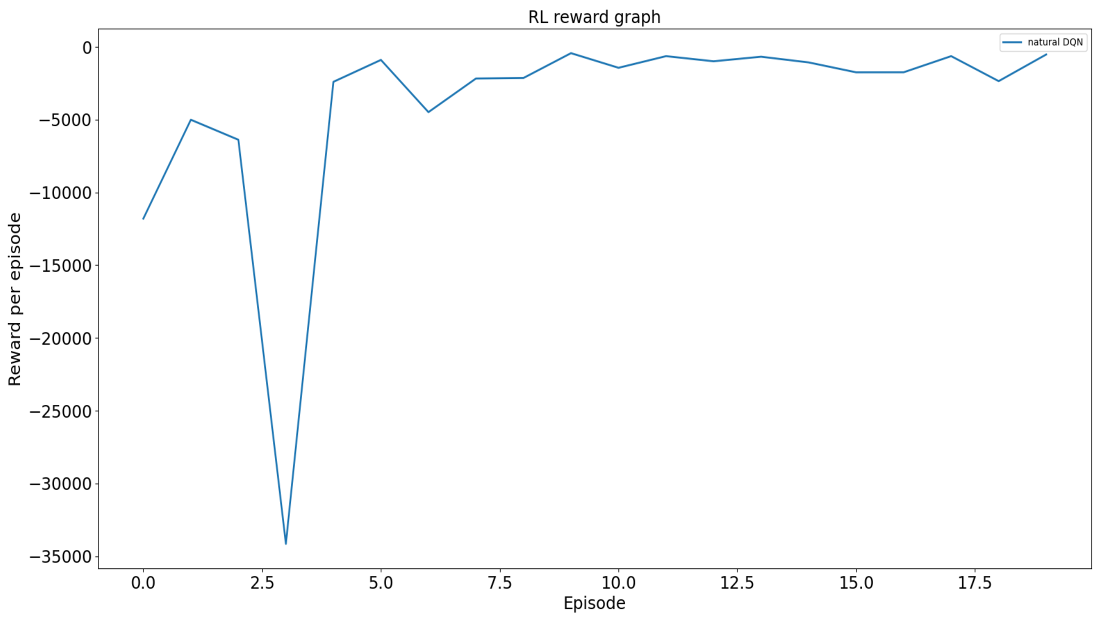
(3.1)

DQN:

Experiment on Mountain-Car


RL training step graph
natural DQN
total training steps
episode
0
10000
20000
30000
40000
50000
60000
70000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


RL cost graph
natural DQN
Cost
Steps
0.0
0.5
1.0
1.5
2.0
2.5
0
10000
20000
30000
40000
50000
60000
70000



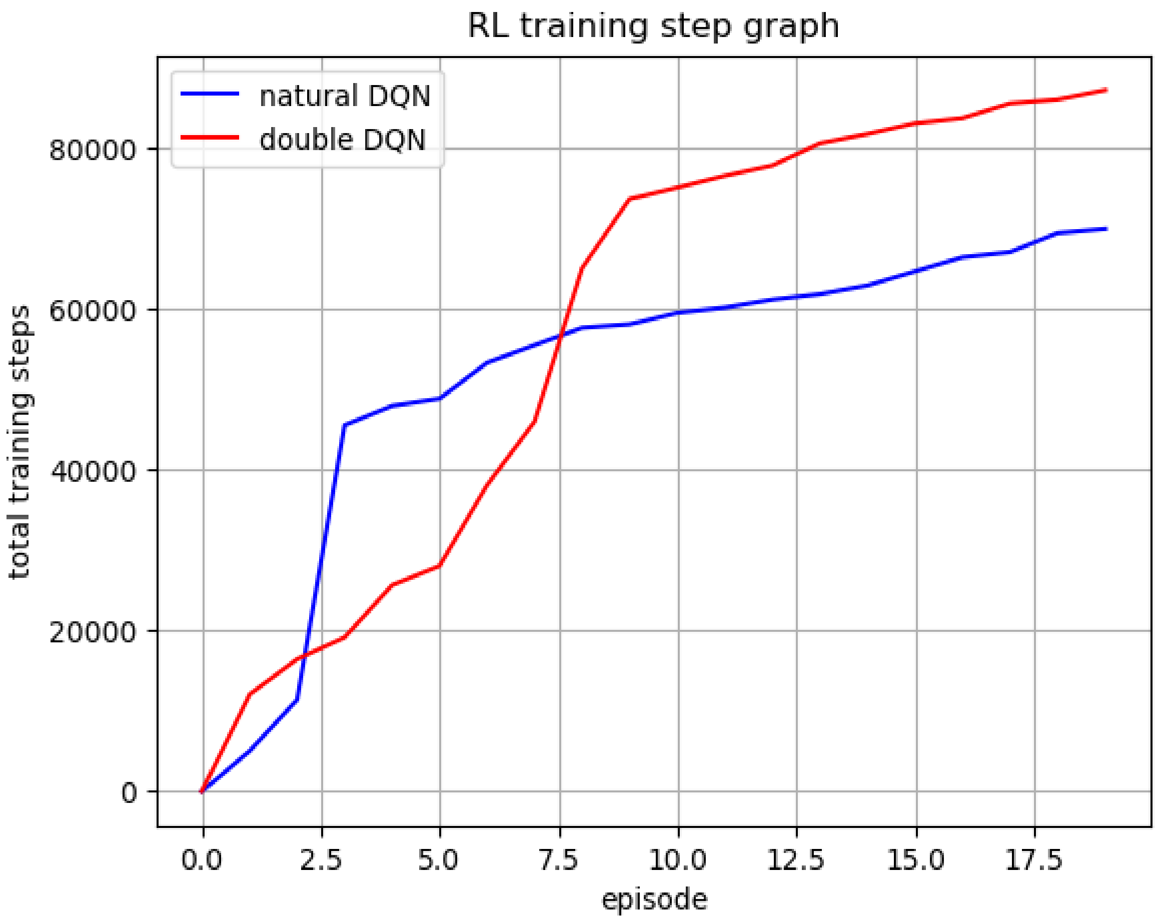
Analysis:

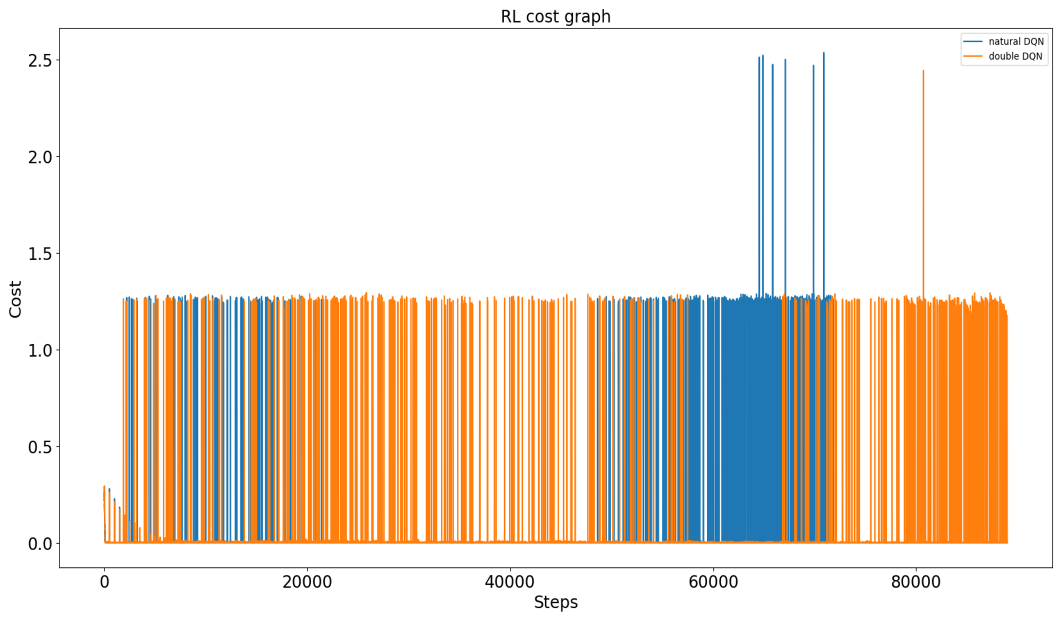
From the RL reward graph we can see that the model **converged** after episode `5`, the reward value per episode remains at a stable level. But we also noticed that in episode `3` the model is stuck in a bad area which results in relatively low reward and high training steps. According to the RL training step graph, we can see that total training steps increase evenly after episode `5` and arrived at `70000` after `20` episodes. The analysis will focus mainly on comparison between natural DQN results and advanced DQN results in the following experiments. The difference shows the effect of improvements.

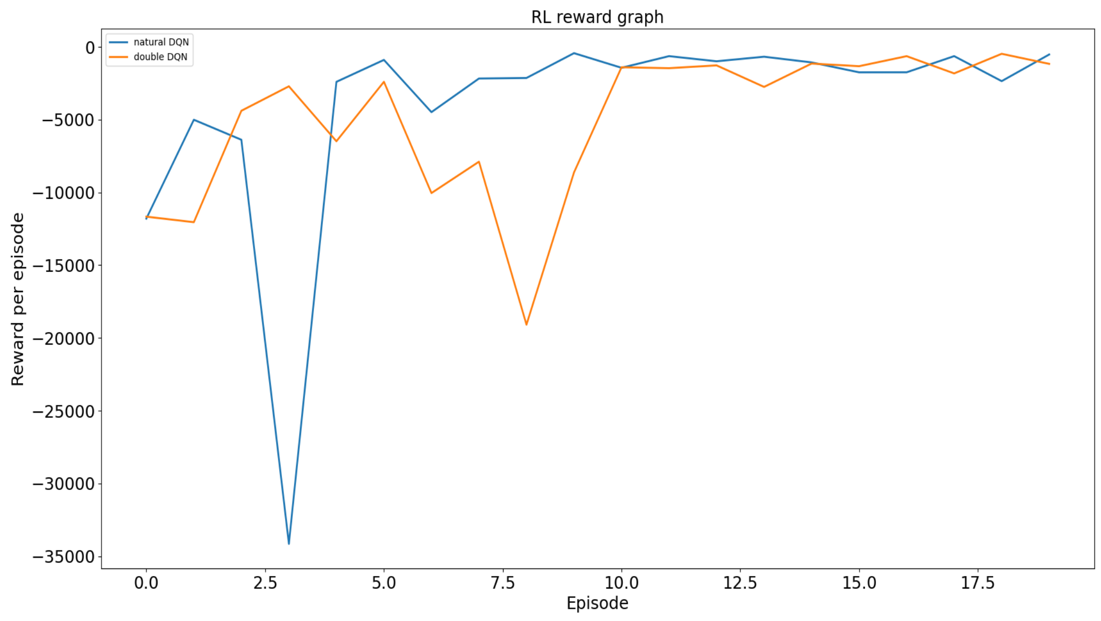
(3.2)

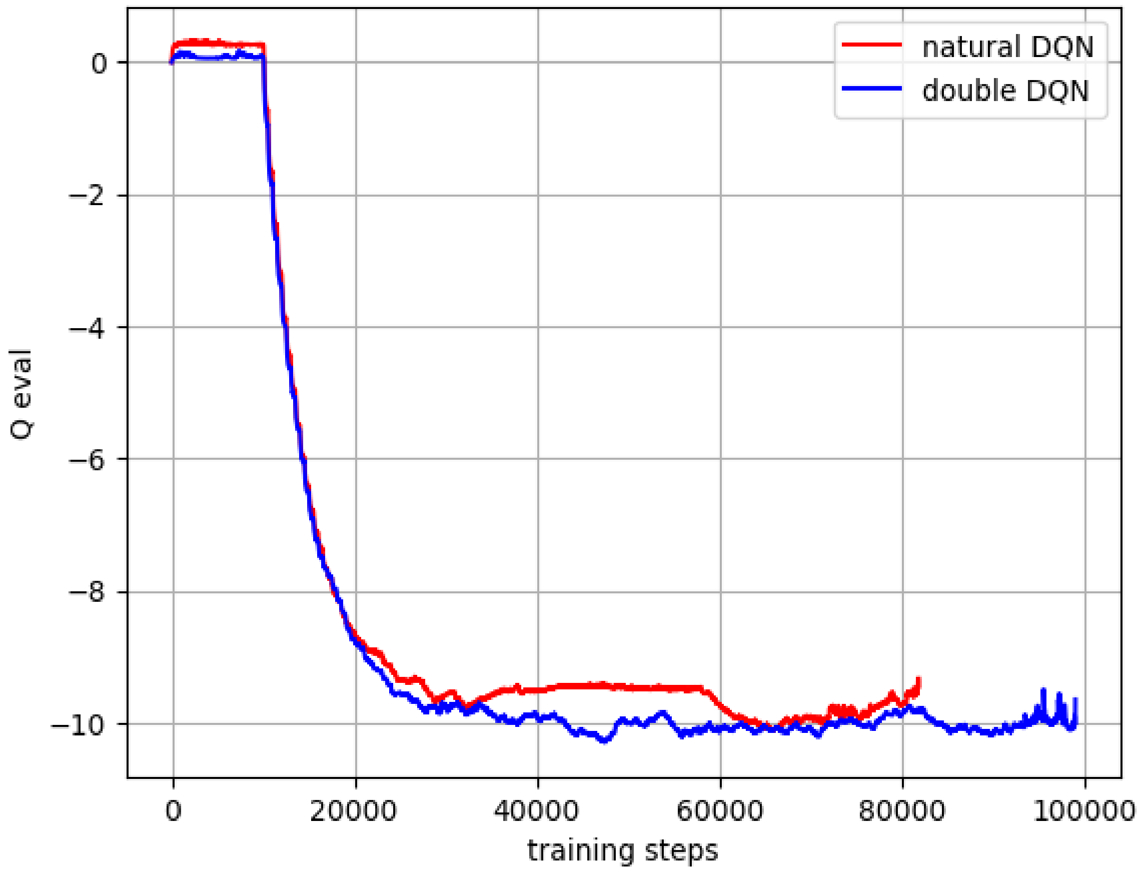
Double DQN:

Experiment on Mountain-Car:

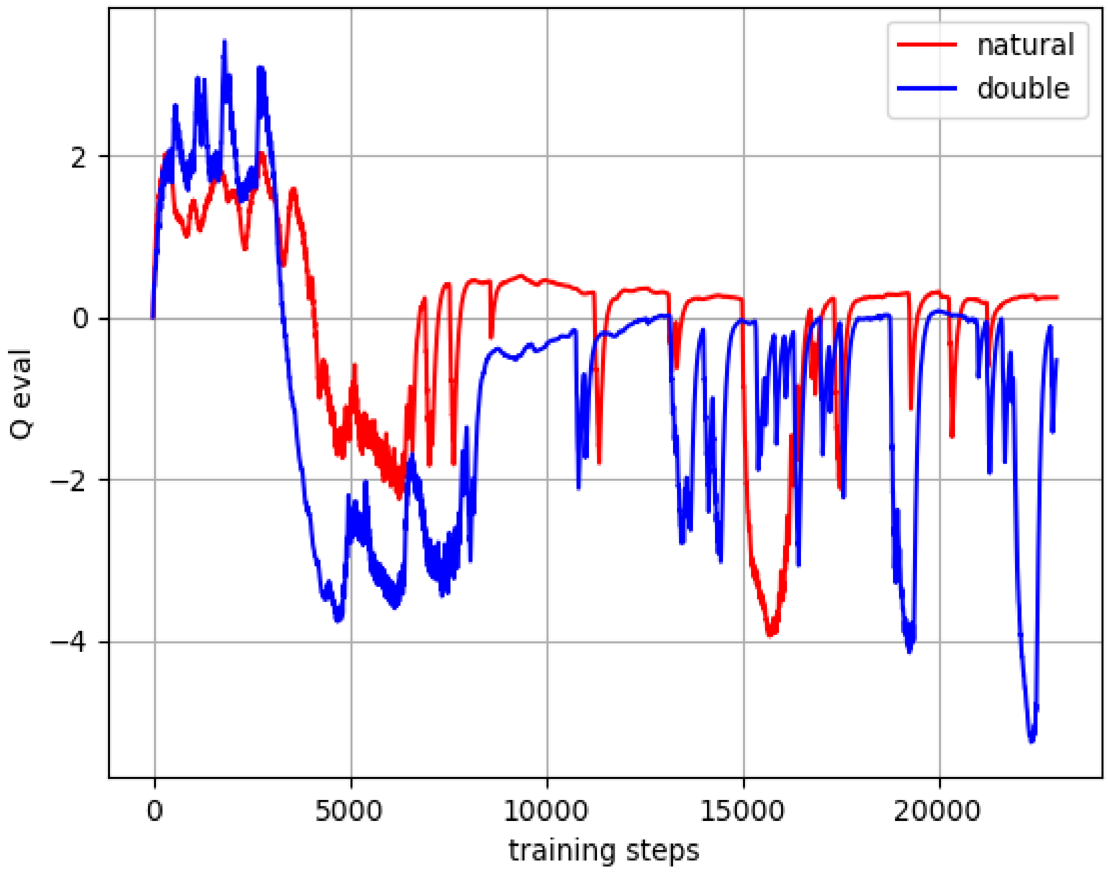

RL training step graph
natural DQN
double DQN
total training steps
episode
0
20000
40000
60000
80000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


RL cost graph
natural DQN
double DQN
Cost
Steps
0.0
0.5
1.0
1.5
2.0
2.5
0
20000
40000
60000
80000





Experiment on Pendulum:



Analysis:

From the RL reward graph we can see that the model **converged** after episode `10`, the reward value per episode remains at a stable level. According to the RL training step graph, we can see that total training steps increase evenly after episode `10` and arrived at `85000` after `20` episodes. In current environment setting, the change in estimating Q-eval value makes the model converge more slowly, which results in more training time and higher calculation cost.

The main difference between double DQN and natural DQN is the way to estimate Q eval value. In order to show that double DQN can generally alleviate the problem of upward bias, we do the experiment on both **Mountain-Car** and **Pendulum** environment settings. From the Mountain-Car Q-eval graph we can see that during training process, double DQN predicts Q-eval more close to the ground truth Q value `-10`. And natural DQN suffers the upward bias problem that it's curve is generally larger than the ground truth value. From the Pendulum Q-eval graph we can see the difference more clearly:

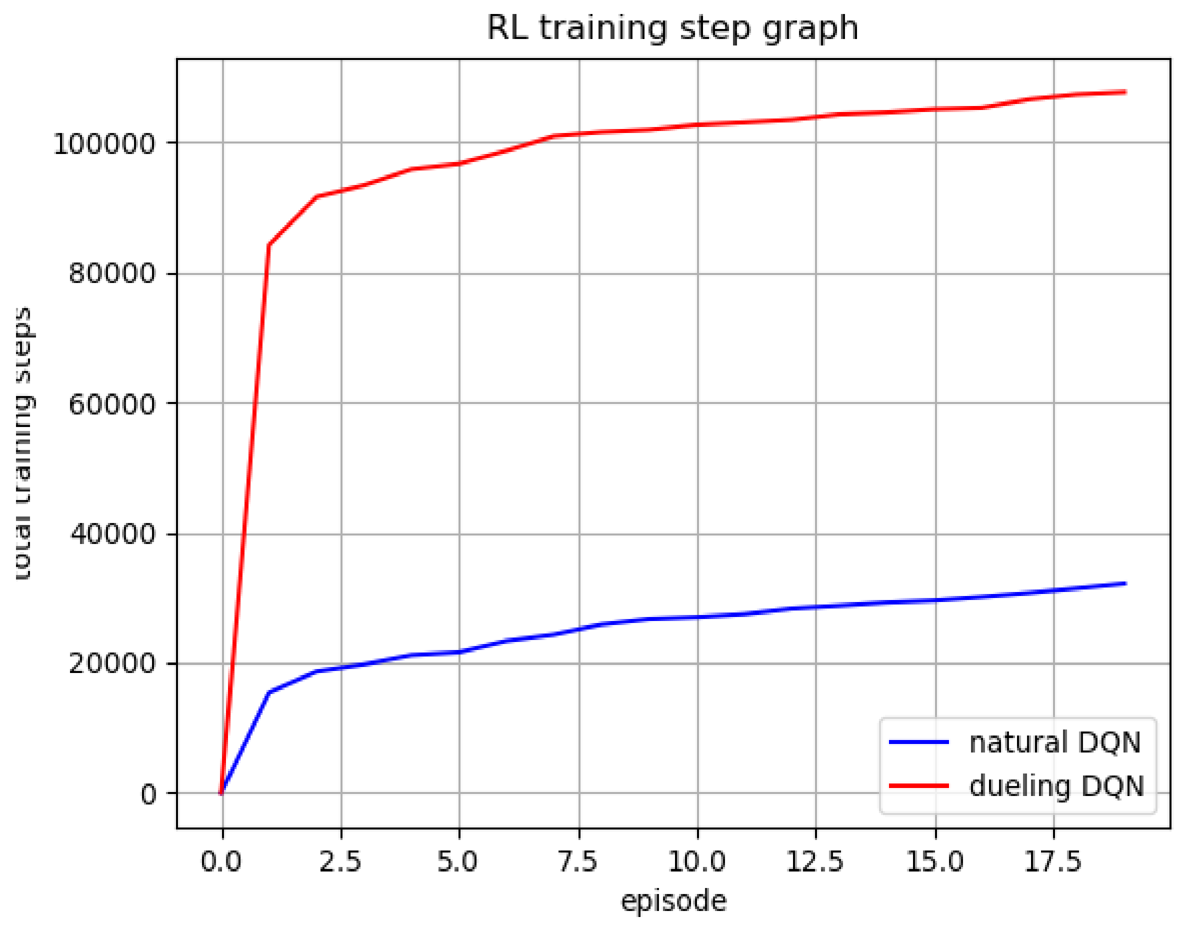
- Ground truth Q-eval value: `0` (pendulum in up-right position)
- Natural DQN estimated Q-eval value is larger than `0` as a whole
- Double DQN estimated Q-eval value is closed to 0 after the algorithm converged.

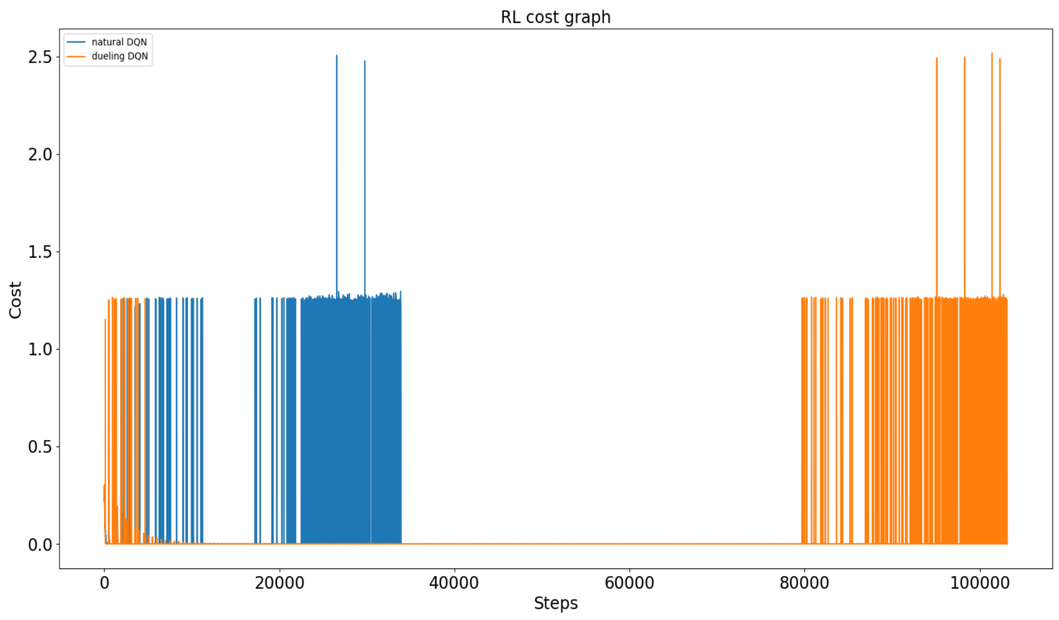
The difference indicates that double DQN can alleviate the upward bias problem and make the prediction of Q-eval more precise.

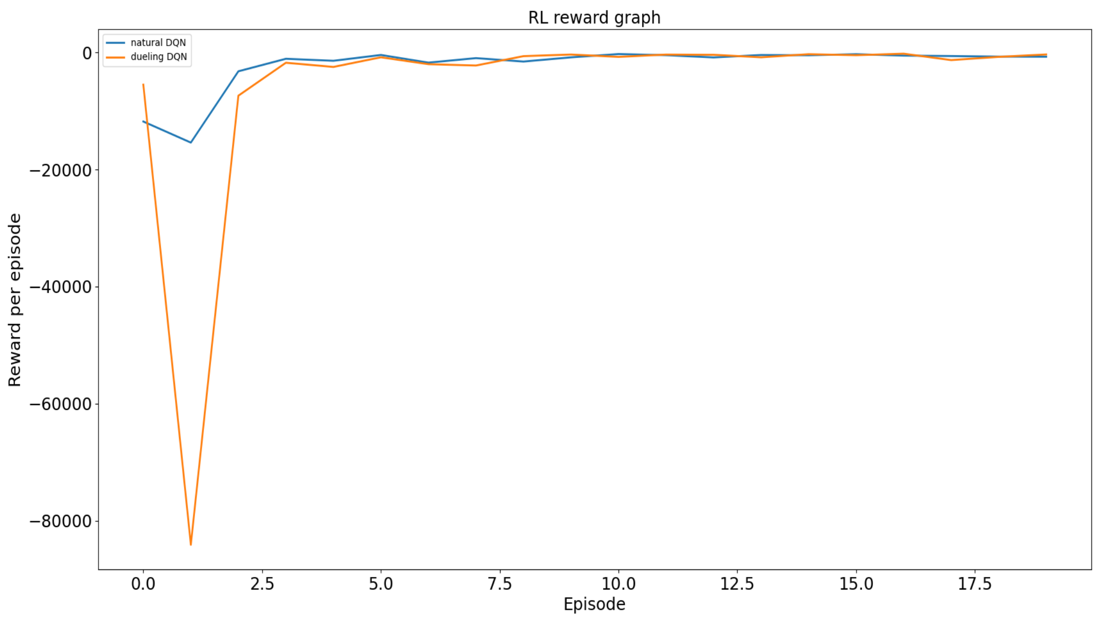
(3.3)

Dueling DQN

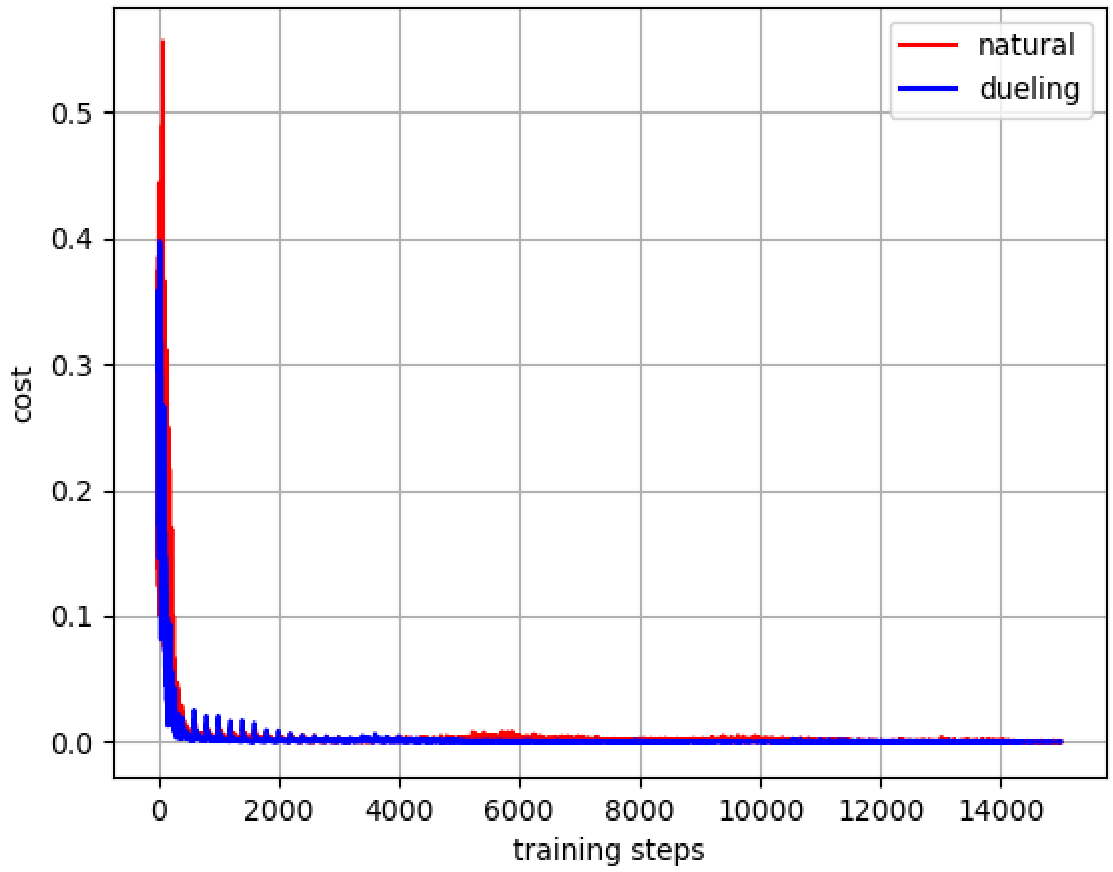
Experiment on Mountain-Car (*n_action*=3):

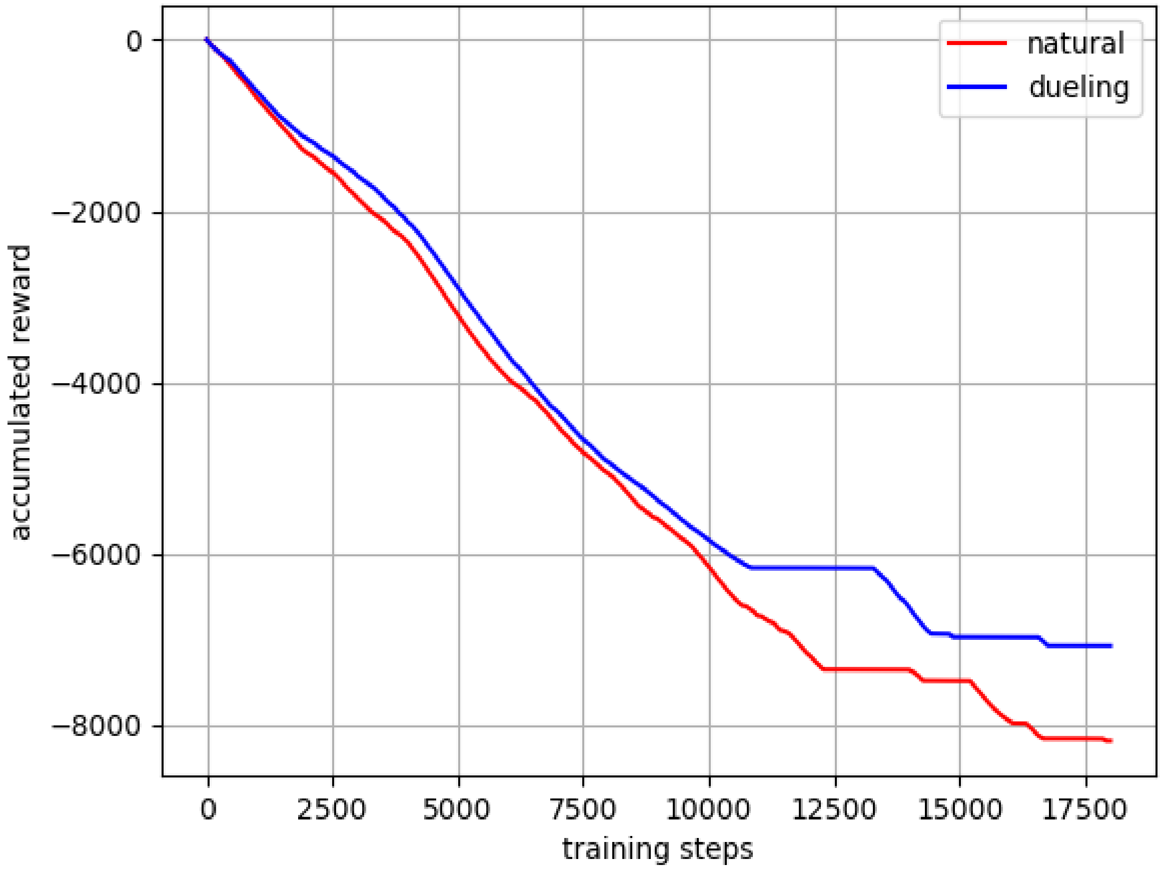




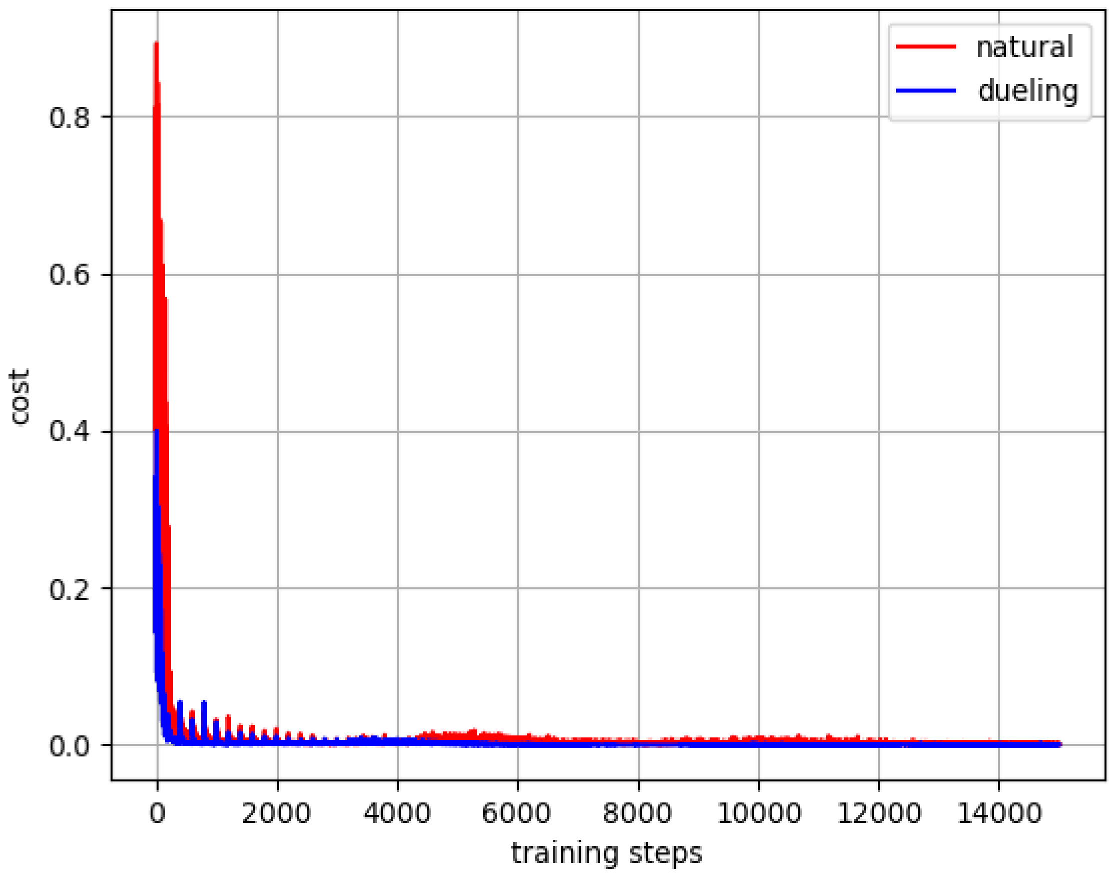


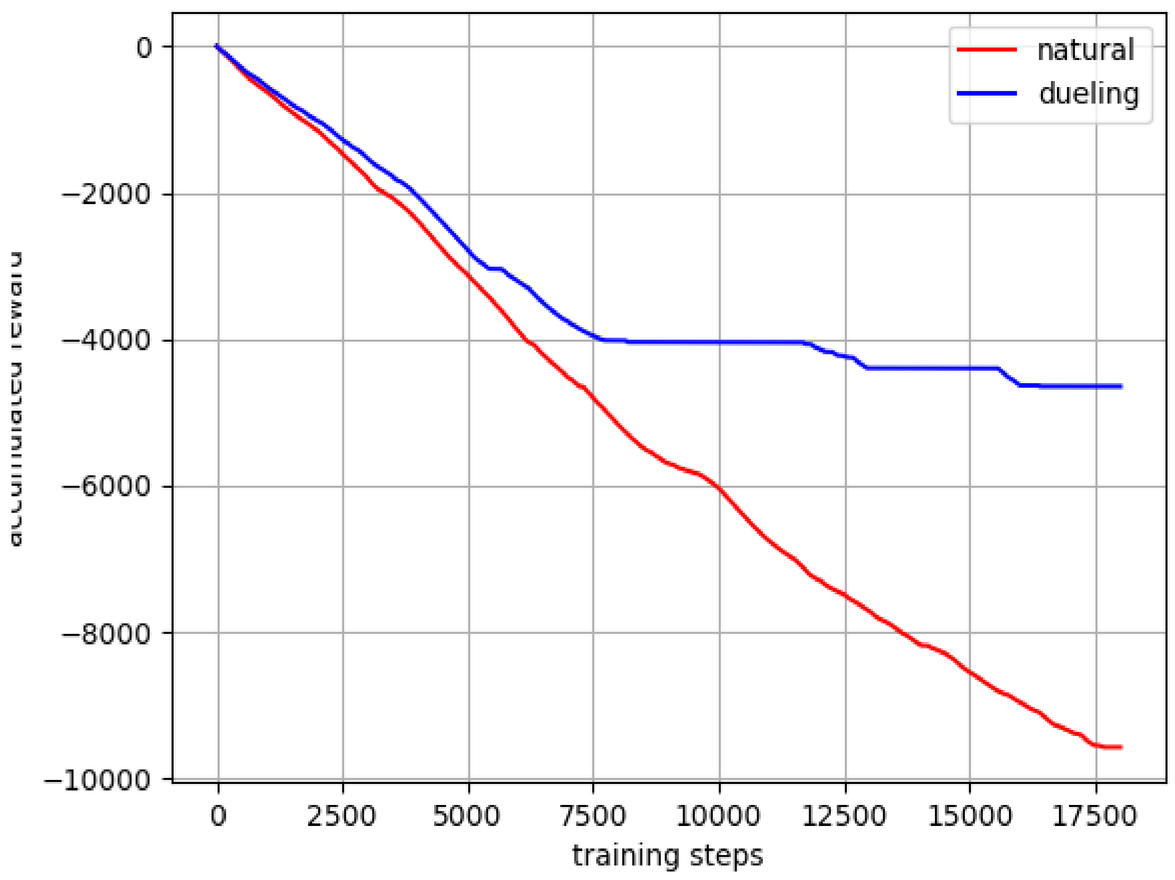
Experiment on Pendulum (n_action=5):


natural
dueling
cost
0.0
0.1
0.2
0.3
0.4
0.5
0
2000
4000
6000
8000
10000
12000
14000
training steps


natural
dueling
accumulated reward
0
−2000
−4000
−6000
−8000
0
2500
5000
7500
10000
12500
15000
17500
training steps

Experiment on Pendulum (n_action=10):





Analysis:

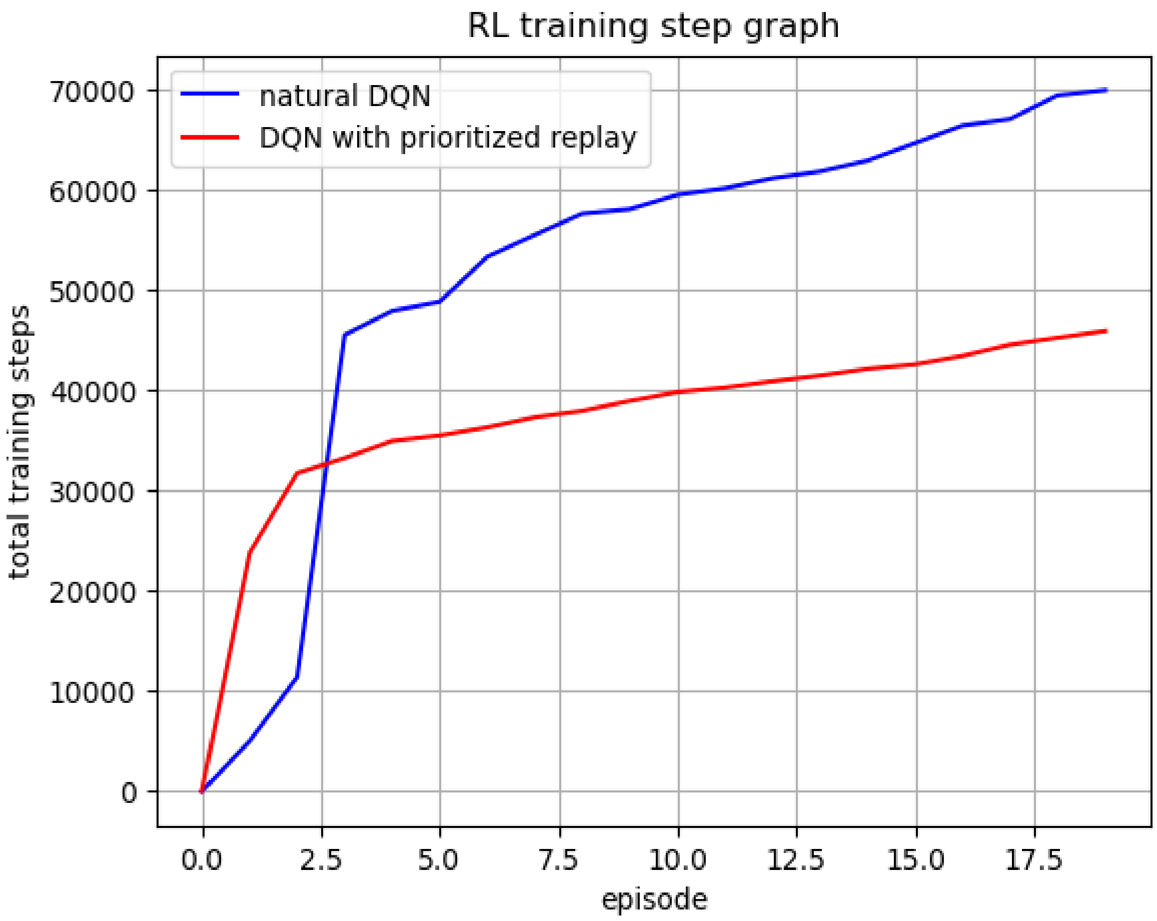
From the RL reward graph we can see that the model **converged** after episode `1`, the reward value per episode remains at a stable level. According to the RL training step graph, we can see that total training steps increase evenly after episode `2` and arrived at `110000` after `20`episodes. The increment in total training steps comes from tremendous training steps on episode 1. In current environment setting, the division in calculating Q value makes the model hard to find the solution at the beginning, which results in extremely high training time calculation cost on the first episode.

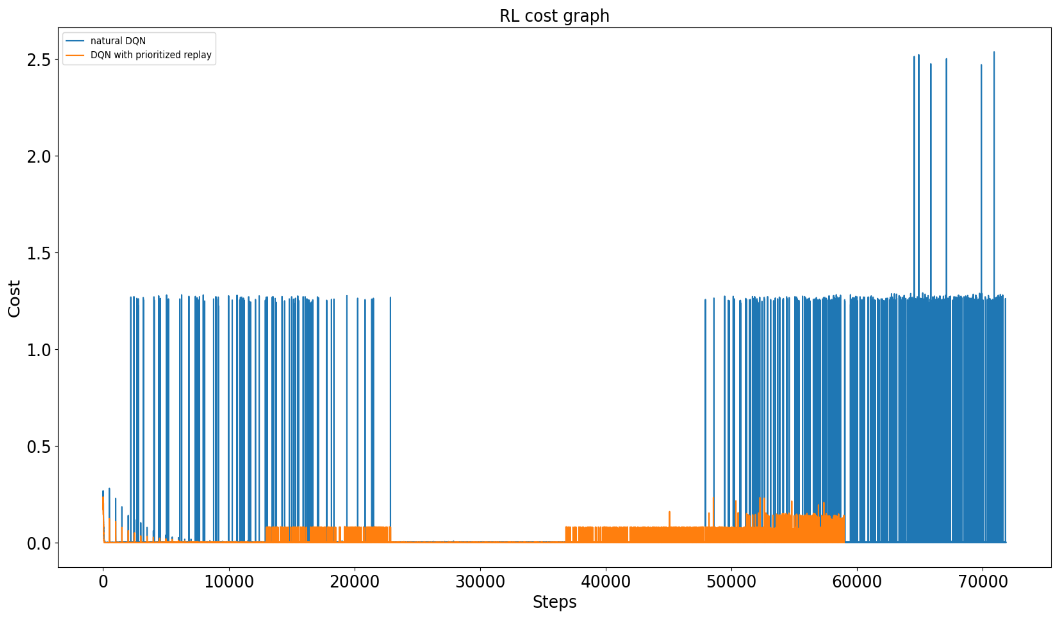
The main difference between Dueling DQN and natural DQN is the way they calculate Q-function. In both **Mountain-Car** and **Pendulum** environment settings, cost per step is similar. But the convergence speed of dueling DQN is faster. In Pendulum *n_action*=5 setting, the Dueling DQN model converged after `10500` while the natural DQN model converged after `12500`. And through the experiment result of different *n_action* setting, we found that the **larger** the action space, the more **obvious** the improvement effect. In Pendulum *n_action*=10 setting, the Dueling DQN model converged after `7500` while the natural DQN model converged after `17500`. Probably because when the action space is large, the model are more likely to meet **action-insensitive** states where Q value is decided by state value and has little to do with action value.

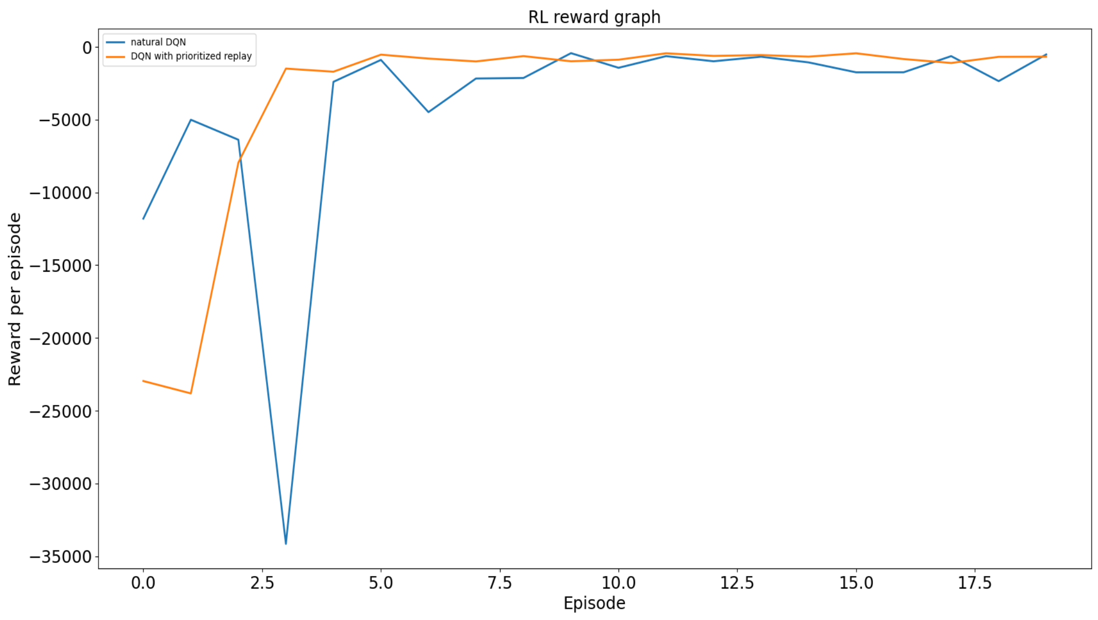
(3.4)

Prioritized DQN:

Experiment on Mountain-Car:







Analysis:

From the experiment results above we can see that prioritized DQN beat natural DQN in every way: total training steps, , convergence speed, cost per step and reward per episode. Thanks to the priority, the model knows how to do the sampling efficiently and effectively. This significant change makes improvements in every way. I think it's the best model for the current environment setting (Mountain-Car V0).