Learning Algorithm

Two algorithms have been experimented in banana collector use cases , where firstly is the baseline model uses Deep Q-Network (DQN) algorithm and secondly is the enhanced version of DQN namely double DQN (DDQN).

Explanation of DQN algorithm.

Deep Q-Networks (DQN) algorithm proposed by Mnih et al. 2013 is a value based temporal different algorithm that approximate the Q-function. Approximated Q-function will be used by an agent to select the action.

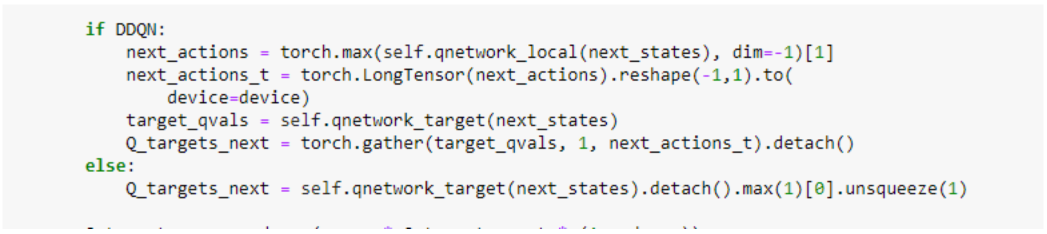
DQN algorithm is off-policy learning that used to improve its sample efficiency compared to algorithm like SARSA. During training for the agent learning, experiences gather by the older policies are stored in experience buffer.

Experience buffer is reuse for the agent learning which differentiated between how SARSA and DQN agent learnt. DQN is possible to decorrelate by reusing experiences through sampling random batches from a large experience replay memory.

Explanation of Double DQN (DDQN) algorithm.

Double DQN (DDQN) is having the same structure as DQN with the difference of having second Q-function approximator. Second Q-function approximator works as a reference to ensure noise coming from exploration stage of Q-value estimation can be reduced hence having better estimation of next action taken in the later stage. For second Q-function approximator instead of using independent DQN, second Q-function approximator use is from taking target network.

For DDQN, next action is first taking from current Q-function approximator and all the matrices of Q-values for next state is retrieve from target network. Using next action from current Q-function approximator and retrieve the value from target network (second Q-function approximator).



```
        if DDQN:
            next_actions = torch.max(self.qnetwork_local(next_states), dim=-1)[1]
            next_actions_t = torch.LongTensor(next_actions).reshape(-1,1).to(
                device=device)
            target_qvals = self.qnetwork_target(next_states)
            Q_targets_next = torch.gather(target_qvals, 1, next_actions_t).detach()
        else:
            Q_targets_next = self.qnetwork_target(next_states).detach().max(1)[0].unsqueeze(1)
```

Figure 1 Code screenshot on differences on DQN and DDQN

This step is to reduce the fluctuation of Q-value from the current Q-function approximator by having a "frozen" second Q-function approximator.

In Figure 1 code written in python if-else statement is the only differences between DQN and DDQN.

Implementation of Experiment and Hyper-parameter Setup

	Deep Q-Network	**Double Deep Q-Network**
Minibatch Size	64	64
Gamma	0.99	0.99
Tau	0.001	0.001
Learning Rate	0.0005	0.0005
Network update	Every 4 time step	Every 4 time step
GLIE - Epsilon Decay Rate	0.995	0.995
GLIE - Epsilon minimum	0.01	0.01
Training time to achieve +13 score over 100 episodes average	240 seconds	492 seconds
Episode needed to achieve +13 over 100 episodes average	350	590

Table 1 Hyper-parameter Setup for DQN and DDQN

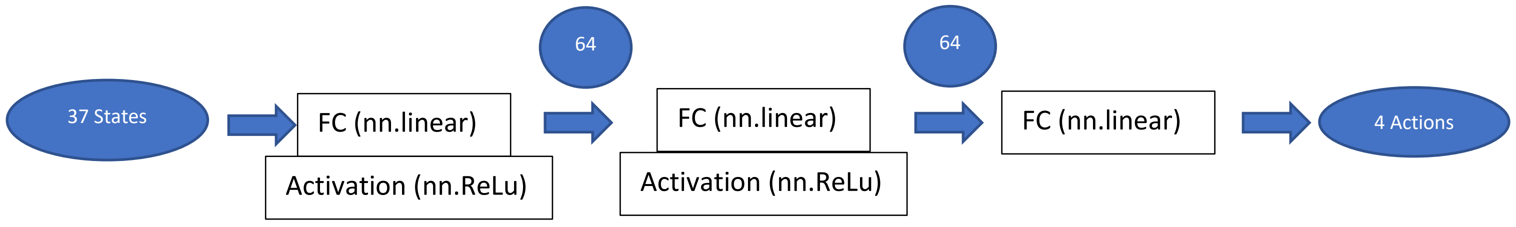


Figure 2 Neural-network connection for DQN and DDQN

Table 1 is the setup and result for achieved for both DQN and DDQN on banana collector use cases, setup in terms of batch size , gamma , tau , learning rate , network update frequency and GLIE are all the same. Figure 2 above is the Q-network that were implemented as experiment for both DQN and DDQN as well.

Results

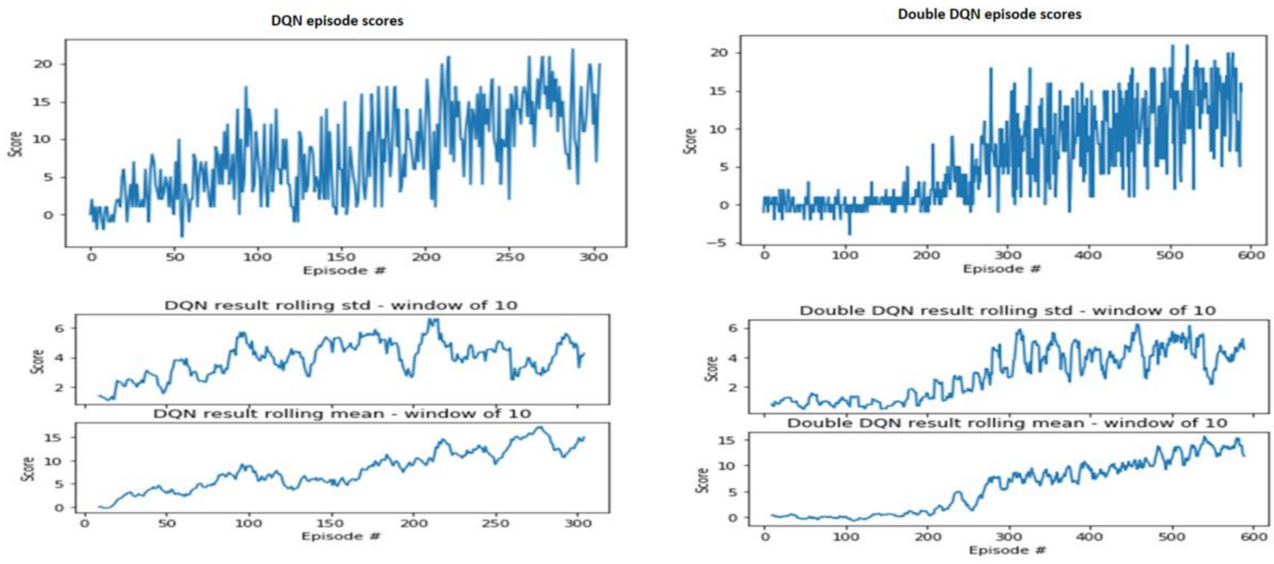


Figure 3 Reward graph in comparison (left : DQN reward graph with rolling std and mean below) and (right : DDQN reward graph with rolling mean and std below).

In this experiment, DQN and DDQN were carried out and result shown in Table 1 that DQN outperform DDQN in terms of time required to achieve score +13 over 100 episodes average. DQN algorithm able to achieve +13 score average within 240 seconds with 350 episodes agent training while DDQN required 492 seconds with 590 episodes agent training before achieving +13 score average.

In Minh et al. 2015 work, DQN proven to be overoptimistic on certain application author tested. Using DDQN method able to reduce overestimation significantly. In banana collector agent , rolling mean and rolling standard deviation over 10 steps were shown in in Figure 3 for DQN on the left and DDQN on the right.

Agent learning through DQN rolling mean shown the instability of the algorithm where significant drop happened twice at time step 250 and time step 290. In terms of stability, DQN rewards rolling standard deviation shown to have more frequent fluctuation compared to DDQN rewards.

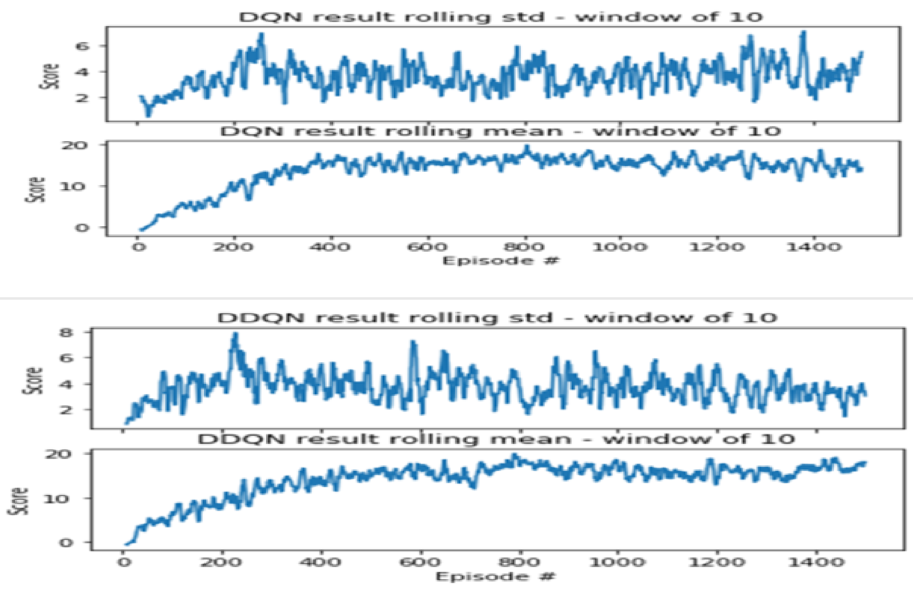


Figure 4 1500 episodes on DQN vs DDQN rolling mean and std for stability check for both algorithm.

Using1500 time steps as experiment for stability test, rolling standard deviation for DQN shown to have more frequent spike in deviation value compared to DDQN. For 1200 to 1500 episodes, differences in standard deviation value proved that DDQN is better at handling stability of agent learning.

Conclusion and Future Work

Both DQN and DDQN is able to achieve +13 average scores over 100 episodes below 1000 episodes, where DQN shown to have achieved +13 average score below 300 episodes while DDQN took 2 times of the episodes number to achieved +13 average score.

Overestimation is the common issue of DQN algorithm where using experiment of 1500 episodes for both networks shown to have surface out the instability of DQN at the later stage of the episode. Using DDQN shown to have better stability even with longer time to achieve +13 score average.

In future work, duelling DQN could be conducted on banana collector agent and compare the performance in terms of the episode needed to converge to +13 scores average and stability of agent rewards. Side by side comparison of duelling DQN versus baseline DQN and DDQN that have been experimented in this project to ultimately understand which network works the best for banana collector environment.

The hyperparameter for this experiment has been fixed to use the same Q-network, hidden layer changes could be another variable to adjust for performance measurement in the future work.