## Algorithm and Implementation

The Prioritized Experience Replay algorithm with Double DQN (priority DDQN) was implemented for this project [1][2]. The Prioritized Experience Replay algorithm samples experiences from the Replay Memory not uniformly but based on a prioritized probability distribution. The priority probability is given based on the "importance" of the experience so that important experiences are more often sampled. The "importance" of the experience is decided based on the magnitude of the temporal-difference (TD) error for each experience. More specifically, when the training process runs for a sampled experience, the TD error ($\delta$) is computed and stored in the Replay Memory along with the experience. Then the Replay Memory is sorted (in descending order) based on $\delta$. The priority probability for the experience, $P(i)$, is then computed as follows.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

where $p_i$ is the priority of the $i$-th experience and given as follows.

$$p_i = \frac{1}{rank(i)}$$

where $rank(i)$ is the rank of $i$-th experience in the sorted Reply Memory. The exponent, $\alpha$, determines the "level" of prioritization, and $\alpha = 0$ corresponds to the uniform sampling.

There is however a theoretical drawback in sampling the experience based on the priority probability. When estimating the expected action value with the stochastic updates, it is assumed that the stochastic updates correspond to the same distribution as its expectation. Sampling based on the priority probability breaks this assumption. This can be corrected by using importance-sampling (IS) weights ($\omega_i$) for the sampled experience in each stochastic update.

$$\omega_i = \left(\frac{1}{NP(i)}\right)^\beta$$

where $N$ is the number of experiences in the Replay Memory. In addition, $\omega_i$ is normalized with its maximum value ($\omega_i = \omega_i/\omega_{i-max}$). When $\beta = 1$, the priority probability $P(i)$ is fully compensated. With the IS weight, the Double DQN model ($\theta$) is updated as follows.

$$\delta_i = R_i + \gamma Q_{target}\left(S_i', argmax_a Q_{local}(S_i', a)\right) - Q_{local}(S_i, A_i)$$

$$\theta \leftarrow \theta + \mu \omega_i \delta_i \nabla_\theta Q_{local}(S_i, A_i)$$

where $\mu$ is the learning rate.

Table 1 shows the pseudo code of the implementation. The following describes several key aspects of the implementation.

(1) The action is selected based on $\varepsilon$-greedy policy, and $\varepsilon$ is linearly annealed from 1.0 to 0.0001 in every episode.
(2) A training series is conducted every new 1000 experiences added to the Replay Memory.

(3) In a training series, 1000 trainings are repeated. For each training, a batch of 64 experiences is sampled based on the priority probability.
(4) The sort of the Replay Memory is not done every training but done once in every 100 trainings because it is computationally expensive.
(5) At the same timing as the Replay Memory sort, the Qtarget for the Double DQN is updated with Qlocal.

Table 1: Pseudo Code of Implementation

| Configuration |
|---|
| Replay Memory (H) size: 1e5<br>Batch Size (K): 64<br>Start $\varepsilon$: 1.0<br>Minimum $\varepsilon$ ($\varepsilon$_min): 0.0001<br>$\varepsilon$ decay (d$\varepsilon$): 0.9<br>Replay Frequency (RF): 1000<br>Number of Training (T): 1000<br>Sort Frequency (SF): 100 |
| **Pseudo Code** |

```
Initialize H, Qlocal, Qtarget, $\varepsilon$, $\alpha$ and b
for episode = 1 to last episode do
  Observe S1
  for $t$ = 1 to end of the episode do
    Choose At corresponding to St from $\varepsilon$-greedy policy with Qlocal model
    Compute $\delta$t
    Store transition (St, At, Rt, St') and $\delta$t in H
    if $t$ % RF == 0 then /* Go to training set every RF updates */
      for $i$ = 0 to T-1 do
        if $i$ % SF == 0 then /* Sort Replay Memory every SF trainings */
          Sort H based on $\delta$
          Update Qtarget with Qlocal for Double DQN ($Q_{target} \leftarrow Q_{local}$)
        end if
        Sample K experiences and corresponding IS weights based on priority probability
        Train Qlocal and update $\delta$ for each experience
      end for
    end if
    St = St'
  end for
  $\varepsilon$ = max($\varepsilon$*d$\varepsilon$, $\varepsilon$_min)
end for
```

## Hyperparameters

Table 2 shows hyperparameters used for the training. Parameter search was conducted for several hyperparameters. For those parameters, the examined values are shown in the "parameter search" column in Table 2. Each final value was selected based on the training quality, which was measured by three metrics: (1) achieving the 100-episode average reward of 13 with fewer episodes, (2) the maximum average reward in the training with 1000 episodes, and (3) small variances in the per-episode reward plots (in other words, if the per-episode rewards jumps around more, the hyperparameter is deemed nonoptimal).

Table 2: Hyperparamters

| | Hyperparameter | Used Value | Parameter Search |
|---|---|---|---|
| DQN Model | Hidden Layer Size | [64, 32, 16, 8] | [64, 64], [64, 32, 16], [64, 32, 16, 8] |
| Training | Batch Size | 64 | No search |
| | Learning Rate | 2.5e-4 | 0.001, 0.0005, 2.5e-4, 6.25e-5 |
| | Discount ($\gamma$) | 0.99 | 0.9, 0.99, 0.999 |
| | $\varepsilon$-greedy ($\varepsilon$) | 1.0 → 0.0001 (linear annealing) | No search |
| Priority Probability | $\alpha$ | 0.5 | 0.5, 0.7 |
| | $\beta$ | 0.4 | 0.4, 0.5 |

## Training Results

The plots of the reward history in the priority DDQN training are shown in Figure 1. The blue plot shows the reward per episode, and the orange plot is the average reward over 100 episode. The average reward reaches its minimum criteria of 13 around 400 episodes. The quick learning seems to be settled down around 700 episode, and the reward improvement becomes very slow after that. The highest average score in the training is 17.32.

The trained model was validated by running the model for 100 episodes. The average reward for those 100 runs was 17.14. Although it is necessary to run more validations to accurately evaluate the trained model, the very close average rewards between the training and the validation (the discrepancy is just 0.18) is a good indication that the Double-DQN estimates the action value with a small (or no) bias.

Out of curiosity, the "vanilla" DQN (uniform sampling and no Double DQN) was also examined for comparing with the priority DDQN. All the hyperparameters and the model architecture were kept same, differences are the sampling method and the way to compute the target Q-value in the TD update. Figure 2 shows the result. The average reward reaches 13 before 300 episodes. The highest average reward is 16.54. While priority DDQN obtains a slightly higher maximum average reward in training than vanilla DQN, it takes more episodes to get the minimum criteria of the average reward (13+). This is contrary to the expectation to the priority sampling for which we expect faster learning. The slow learning might be a side effect of the Double DQN which tries to avoid "overestimation" of the Q function [2] (I am not sure…).

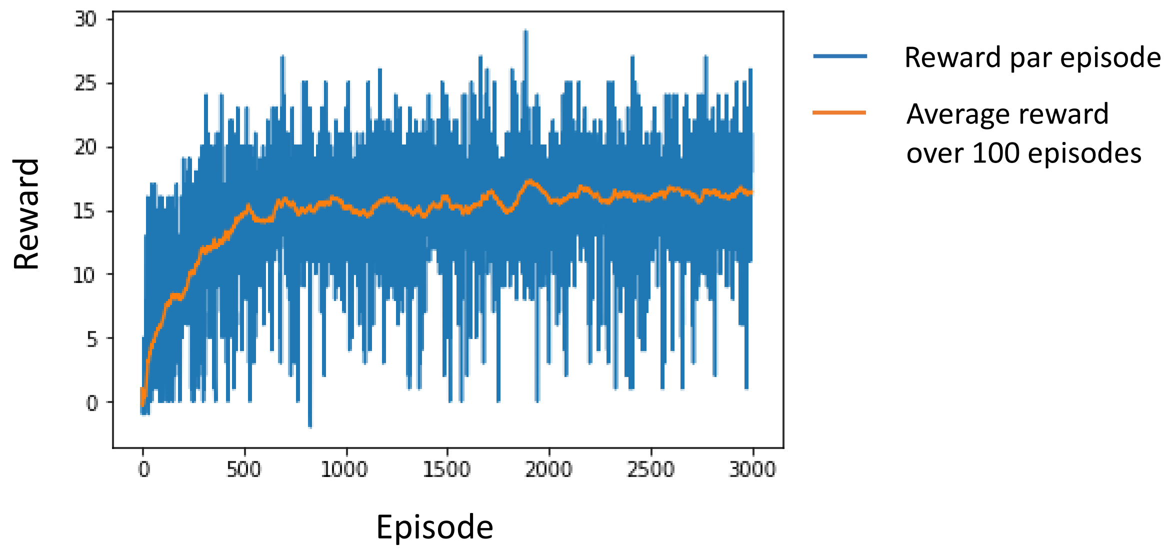

Figure 1:  Reward history in the training of Priority-DDQN

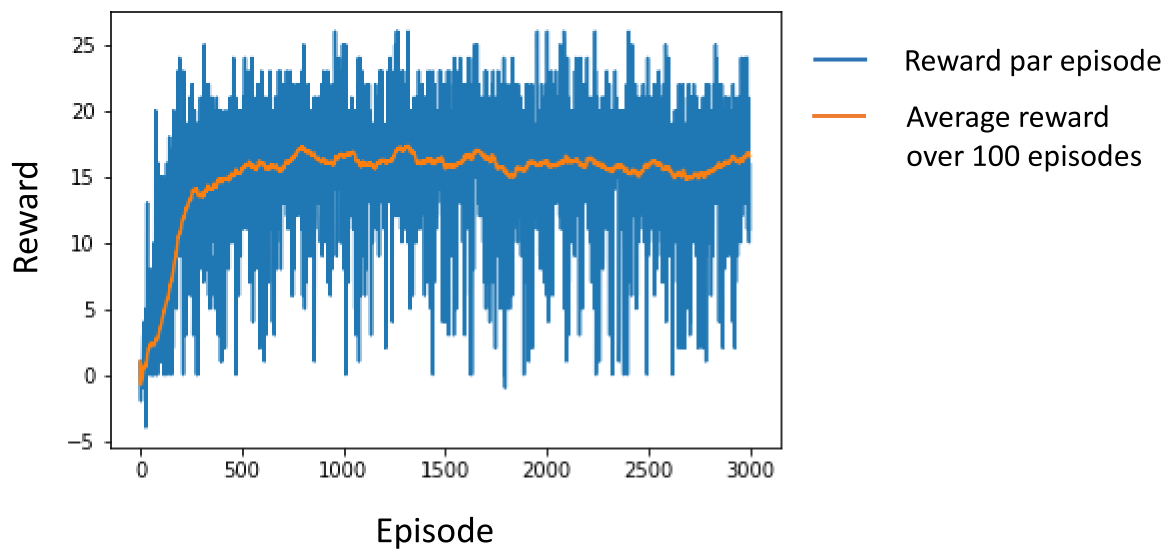

Figure 2: Reward history in the training of “Vanilla” DQN

The “vanilla” model was also validated with 100 runs. The average reward for those runs was 16.15 which is slightly smaller than the validation result of priority DDQN. Figure 3 shows the boxplots for the 100 validation runs for both cases. The distribution of the validation rewards and the median value also show that the priority DDQN model works slightly better than vanilla DQN.

## Future Work

In this project, priority DDQN and vanilla DQN were examined, and both training methods satisfied the minimum criteria for the average reward. Contrary to the expectation, however, the superiority of the priority DDQN against vanilla DQN was not clearly show. In particular, I am disappointed that priority DDQN showed a slower learning curve than vanilla DQN. This might be due to nonoptimal selections of hyperparameters. Continuing on the hyperparameter search would be one of future works. In addition, 100 runs for the model validation are far from enough. As the future work, it is necessary to conduct much more validation runs to accurately evaluate the trained models.

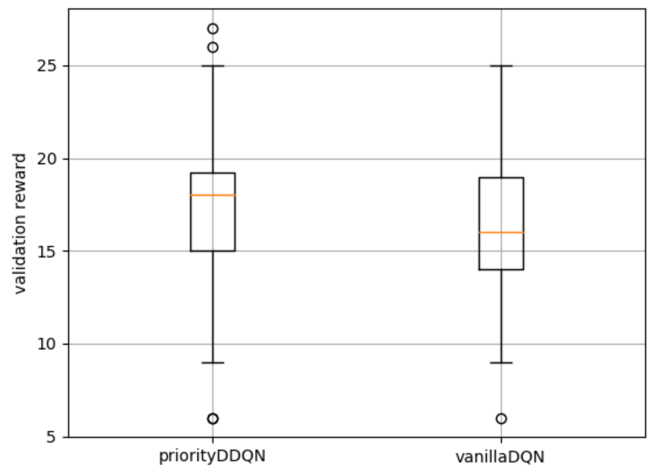

Figure 3: Boxplots for the validation results for priority DDQN and vanilla DQN

## References

[1] Tom Schaul, *et al*. “Prioritized Experience Replay”, conference paper at ICLR 2016.

[2] Hado van Hasselt, *et al*. “Deep Reinforcement Learning with Double Q-learning”, Proceedings of 13th AAAI Conference on Artificial Intelligence, 2016.