Foundations of Reinforcement Learning
Project Report: Estimation Bias in Q Learning

Aditya Upadhyayula, Ketul Shah, Mai Rajborirug

December 10, 2020

Code : https://github.com/Adibuoy23/Deep-Reinforcement-Learning

1 Introduction

In Reinforcement Learning literature [4], the role of an agent to learn the state action space in order to implement optimal policies and make decisions that optimize its cumulative reward. Several algorithms have been proposed in the literature in order for the agent to learn the optimal policy. Q-learning [7] is one of the most popular model free reinforcement learning algorithms. In this algorithm, the agent iteratively learns the policy function by choosing the actions that have immediate high value, and in turns uses those actions to update the value functions. However, Q-learning has its downsides too. It overestimates the value associated with greedy actions. These over estimations result from a positive bias that is introduced in Q-learning. The bias is a result of the maximum action value as an approximation for the maximum expected state action value [1]. To overcome this limitation, double Q learning is introduced [4], [1]. At the core of this algorithm is using two Q estimator functions to stochastically approximate the action value space, instead of one Q estimator. In this report, we investigate the issues that result in estimation bias from both Q learning and Double Q learning perspective, and further extend it to Deep Q network architectures.

The report is organized as follows. In the second section, we go through the definition of Q-learning algorithm. In the third section, we demonstrate the estimation bias in Q learning use the simple Markov Decision Process (MDP). We will also provide mathematical proofs for the upper and lower bounds of the estimation bias in Q learning. Further, we will investigate factors that result in estimation bias. There are environmental factors such as the number of states, actions, and the stochasticity in rewards etc. And there are functional approximation factors that might contribute to the estimation bias (which we will demonstrate using Deep Q networks later in the report). In the forth section, we introduce the double Q-learning algorithm, which can mitigate the overestimation bias problem to an extent. In the fifth section, we empirically analyze the performance of Q-learning and double Q-learning algorithms in measuring the estimation bias as a function of both environmental and functional approximation factors. Our work expands into Deep Q networks (DQNs) and double Deep Q networks (Double DQNs) as well to demonstrate these estimation biases.

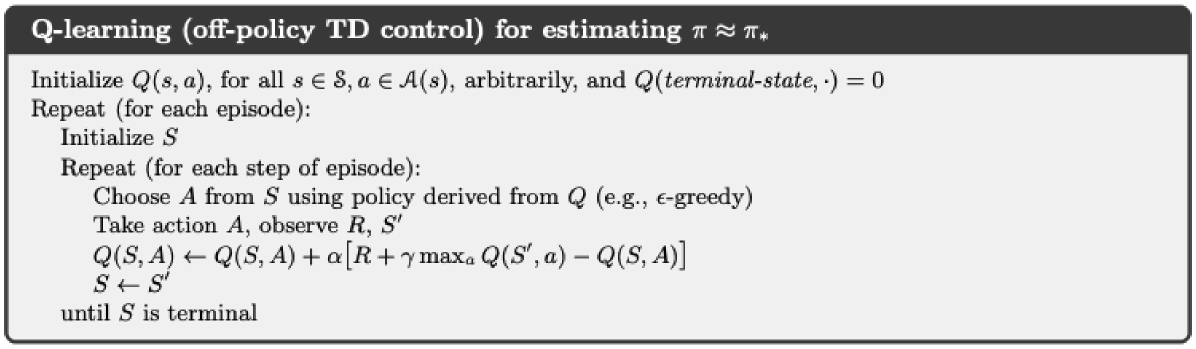
2 Q-Learning

Q-Learning is a popular Off-policy Temporal Difference (TD) learning method. Temporal difference methods combine ideas from Monte Carlo methods and Dynamic Programming (DP). Just like

Monte Carlo methods, experience is used for policy evaluation or the prediction problem. The update rule for state action value function for Q-Learning [7] is given as below:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

The learned action value function Q approximates the optimal action value function q^*. The Q-learning algorithm is formally described below (source: [4]):



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

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

The main reason for Q learning being an off-policy learning method is because of the max () operator. We need to store the values associated with all the actions for the next state in order to choose the best possible action. Other variants of Q learning such as expected Q learning can also be used. In these methods, we use the expected value associated with the next state, instead of choosing the maximum value. This however, might have exploration vs. exploitation trade offs, and this is out of the scope of our current project.

3 Overestimation in Q-Learning

Q-Learning algorithm is known to overestimates the values for greedy actions. These over estimations are caused by taking the maximum of action values as a substitute for maximum expected action value.

Consider the following example. Let $\{X_1, ..., X_M\}$ be a set of M random variables, and we want to estimate the maximum expected value of the variables in the set, given as $\max_i E\{X_i\}$, as calculating it exactly is very difficult without the knowledge of distributions of the random variables and their parameters. Let $S = \bigcup_{i=1}^{M} S_i$, where S_i are the samples from X_i. Let the PDF and CDF of X_i be given by $f_i(x)$ and $F_i(x)$. $E(X_i)$ can be approximated by the empirical mean $\mu_i(S) = \frac{1}{|S_i|}\Sigma_{s \in S_i} s$. And hence, the simplest way to estimate the maximum expected value is to approximate it as follows, using a maximal estimator:

$$\max_i E(X_i) = \max_i E(\mu_i) = \max_i \mu_i(S)$$

It is shown in [1] that using this single estimator causes positive bias, and using a double estimator mitigates it. Please refer to [1] for a more detailed proof.

We experimentally show overestimations of action values on using Q-Learning by reproducing the example from [4]. For this example, consider the simple Markov Decision Process (MDP) given in the Figure 1.

There are two non-terminal states A and B. The agent starts from state A, and can take either of the two actions, left or right. On taking a right, it ends up in a terminal state with a reward of zero. If it takes right, it ends up in B with a reward of zero, but can only go to the terminal state

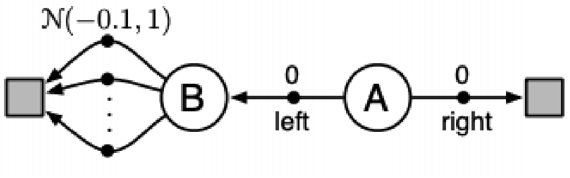


Figure 1: Simple example MDP to demonstrate overestimation of action values using Q-Learning.

from B via one of the M actions with reward distributed as $N(-0.1, 1)$. In this example, taking action left from state A is always sub-optimal. In the Figure 2, we can see that Q-learning with ϵ-greedy action selection initially learns to strongly favor the left action on this example. Q-learning takes the left action about 5% more often than is optimal, even asymptotically.

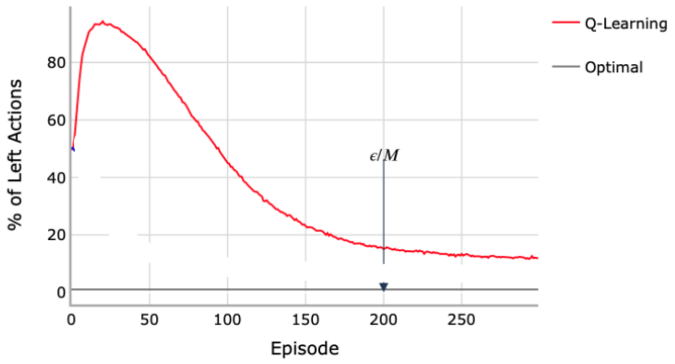


Figure 2: Q-Learning with ϵ-greedy action selection learns to favour left action strongly.

This is a direct measure of the estimation bias in Q learning. Under some conditions, we show the proofs of upper and lower bounds on the overestimation below. The following mathematical proofs show how the estimation biases of Q-learning are bound

3.1 Lower bounds for estimation biases of Q-learning

Estimation errors of any kind can cause a positive bias. These errors could be due to environmental noise, function approximation, non-stationarity or any other source. The following theorem from Hasselt 2015 [6] shows that even the action-value functions are on average correct, the estimation errors of any source can drive the estimation up. The lower bound of Q-learning estimation biases on Theorem 1 decreases as number of action increases.

Theorem 1. Consider a state s in which all the true optimal action values are equal at $Q_*(s, a) = V_*(s)$. Let Q_t be the arbitrary value estimates that are averagely unbiased $\sum_a (Q_t(s, a) - V_*(s)) = \sum_a (X_a) = 0$, but all individually unbiased, such that $\sum_a (Q_t(s, a) - V_*(s))^2 = \sum_a (X_a) = mC$ for some $C > 0$, where C represent some form of biases' variance and $m = |\mathcal{A}| \geq 2$ is the number of the actions in state s. Then, the lower bound of estimation bias is

$$Q_t(s, a) - V_*(s) = X_a \geq \sqrt{\frac{C}{m-1}}$$

Proof by contradiction. First let $\{X_i^+\}$ be the set of positive X of size n, and $\{X_j^-\}$ be the set of strictly negative X of size $m-n$. If $n=m$, then $\sum_a X_a = 0 \rightarrow X_a = 0\ \forall a$, which contradicts $\sum_a X_a^2 = 0 \neq mC$. Hence, it must be $n \leq m-1$. Next, we assume the set of biases $\{X_a\}$ has the maximum value of $\max_a X_a < \sqrt{\frac{C}{m-1}}$. Then, $\sum_{i=1}^{n} X_i^+ \leq n \max_i X_i^+ < n\sqrt{\frac{C}{m-1}}$. This implies $\max_j |X_j^-| \leq \sum_{j=1}^{m-n} |X_j^-| < n\sqrt{\frac{C}{m-1}}$. Applying the Hölder's inequality,

$$\begin{aligned}||(X_j^-)^2||_1 &\leq ||X_j^-||_1 \cdot ||X_j^-||_\infty \\ &= \sum_{j=1}^{m-n} |X_j^-| \cdot \max_j |X_j^-| \\ &< n^2 \frac{C}{m-1}\end{aligned}$$

Use these relation to compute the upper bound of the sum of square biases X_a:

$$\begin{aligned}\sum_a^m (X_a)^2 &= \sum_{i=1}^{n} (X_i^+)^2 + \sum_{j=1}^{m-n} (X_j^-)^2 \\ &< n\frac{C}{m-1} + n^2\frac{C}{m-1} \\ &= C\frac{n(n+1)}{m-1} \\ &\leq mC\end{aligned}$$

The result contradicts the assumption that $\sum_{a=1}^{m} X_a^2 = mC$. Thus, the estimation bias lower bound is $\max_a X_a \geq \sqrt{\frac{C}{m-1}}$

3.2 Upper bounds for estimation biases of Q-learning

Thrun and Schwartz 1993 [5] showed that if the action values contained random errors uniformly distributed in $[-\epsilon, \epsilon]$ then the overestimations of each target are upper bounded by $\epsilon\frac{m-1}{m+1}$ where m is the number of actions.

Theorem 2. Consider a state s in which all true optimal action value are equal at $Q_*(s,a) = V_*(s)$. If the estimation error $Q_t(s,a) - V(s) = X_a$ are independently distributed uniformly randomly in $[-epsilon, epsilon]$. then

$$E\left[\max_a Q_t(s,a) - V_*(s)\right] = E\left[X_a\right] = \epsilon\frac{m-1}{m+1} \tag{1}$$

Direct Proof. Because the estimation errors are independent, we get

$$\begin{aligned}P(\max_a X_a \leq x) &= P(X_1 \wedge X_2 \leq x \wedge ... \wedge X_m \leq x) \\ &= \prod_{a=1}^{m} P(X_a \leq x)\end{aligned}$$

The function $P(X_a \leq x)$ is the cumulative distribution function (CDF) of X_a. Thus,

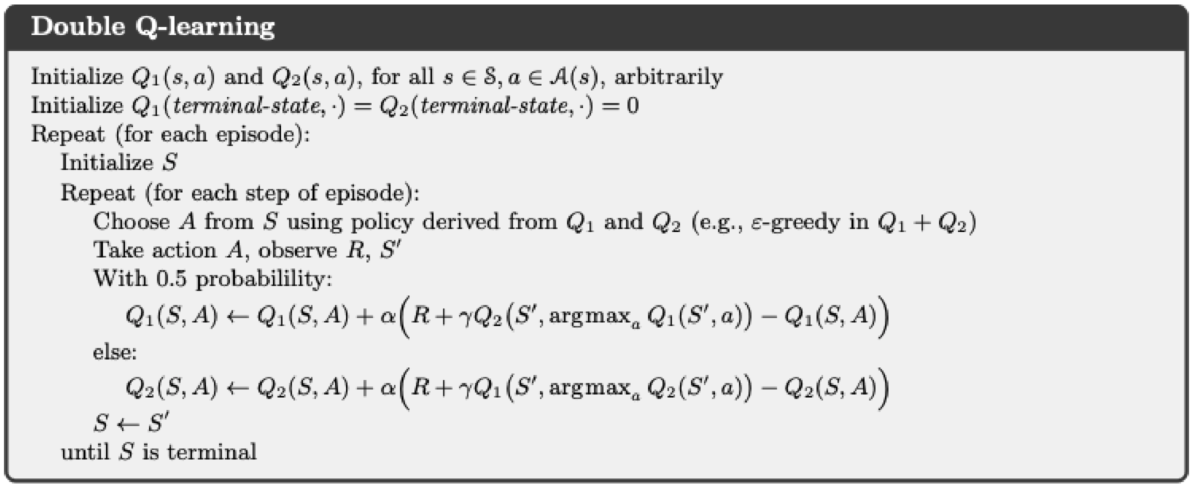
$$
\begin{aligned}
P(\max_a X_a \le x) &= \prod_{a=1}^{m} P(X_a \le x) \\
&= F_{\max}(x) \\
&= \begin{cases} 0, & \text{if } x \le -\epsilon \\ \left(\dfrac{\epsilon + x}{2\epsilon}\right)^m, & \text{if } x \in (-\epsilon, \epsilon) \\ 1, & \text{if } x \ge \epsilon \end{cases}
\end{aligned}
$$

Differentiate the previous CDF to get the probability density function (PDF) and use it to find the estimation of maximum bias, $f_{\max}(x) = \frac{d}{dx}F_{\max}(x) = \frac{m}{2}(\frac{\epsilon+x}{2\epsilon})^{m-1}$ when $x \in (-\epsilon, \epsilon)$ and $f_{\max}(x) = 0$ otherwise. The integration gives

$$
\begin{aligned}
E\left[\max_a X_a\right] &= \int_{-1}^{1} x f_{\max(x)dx} \\
&= \epsilon \frac{m-1}{m+1}
\end{aligned}
$$

4 Double Q-Learning

An alternative way to approximate the maximum expected value for any set of random variables is given by using double estimators, given by [1]. This method is shown to mitigate the estimation bias associated with Q learning 2. The central idea is to use two Q estimators - one to choose the value associated with the best action from the next state, and the other uses the action to update the state action value. Both of these update independently of one another, thus resulting in mitigation of the estimation bias. Formally, Double Q-Learning algorithm is given as below, taken from [4]:



Double Q-learning

```
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Repeat (for each step of episode):
      Choose $A$ from $S$ using policy derived from $Q_1$ and $Q_2$ (e.g., $\varepsilon$-greedy in $Q_1 + Q_2$)
      Take action $A$, observe $R$, $S'$
      With 0.5 probabilility:
         $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
      else:
         $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
      $S \leftarrow S'$
   until $S$ is terminal
```

In the simple MDP example shown in Figure 1, we perform experiments using Double Q-Learning and show that it improves the performance and reduces the overestimations of action values as shown in Figure 3 and Figure 4.

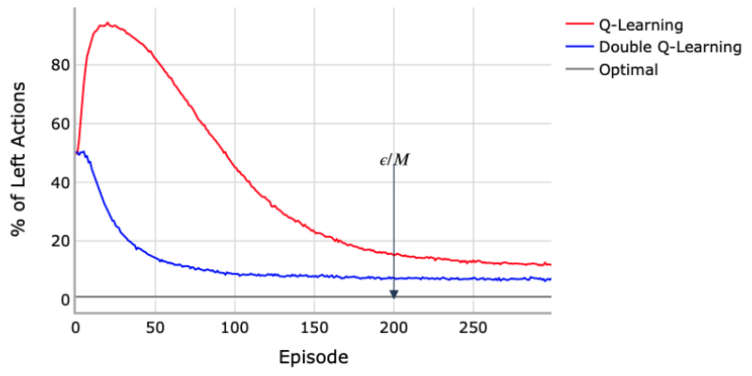


Figure 3: Double Q-Learning with ϵ-greedy policy reduces the % of time left (sub-optimal) action is selected.

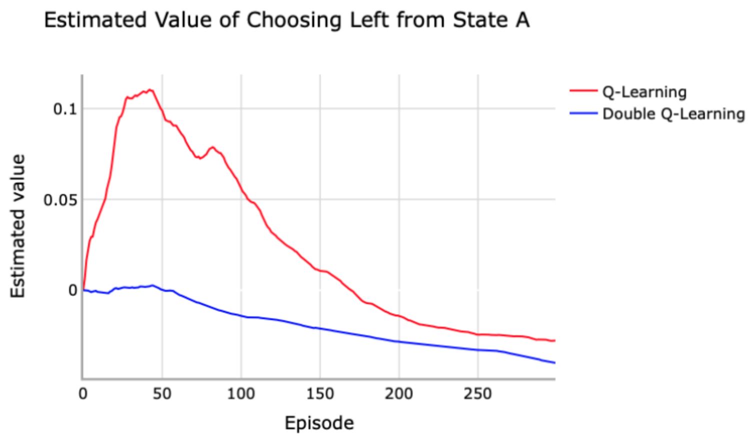


Figure 4: Double Q-Learning with ϵ-greedy policy reduces the overestimations of Q values.

5 Bias-inducing factors

Many factors can cause estimation errors which lead to a positive bias. We consider three such factors and experimentally analyse the performance of Double Q-Learning, and compare it to Q-Learning. For environmental factors, we evaluate the bias as a function of number of actions. We then experiment with Deep Q Networks (DQNs) to investigate the effect of functional approximation on estimation bias [2] and show that Double Deep Q Networks [6] (Double DQN) reduces the estimation bias compared to DQN, for Atari games Pong and Alien.

5.1 Bias due to environmental factors

Consider the above example MDP setup shown in Figure 1. Here, we evaluated the estimation error by varying the number of actions from B to the terminal state. We perform the same experiment by changing the reward distribution from normal to a uniform distribution. The estimation error in the case of Q-Learning and Double Q-Learning is given as follows:

$$\max_a Q(s,a) - V_*(s): \text{ for Q-Learning}$$

$$Q'(s, \arg\max_a Q(s,a)) - V_*(s)\text{: for Double Q-Learning}$$

The results of estimation errors from our experiments in both these cases are given in Figure 5 below. The plot on the left is for the case when reward function is a normal distribution with a mean of -0.1 and standard deviation of 1, whereas the one on right is for a uniform reward distribution. In this example, we do not have access to $V_*(s)$. However, we do know that the reward distribution is centered around -0.1. Therefore we assumed that the true $V_*(s) \approx -0.1$ (which is the overall expected value of taking a given action). These two figures illustrate that the estimation bias of Q learning is higher than that of Double Q learning in both the cases. Further, in the uniform reward distribution case, the estimation bias increases as a function of number of actions which can be verified from the upper bound proof introduced in the above section 3.2.

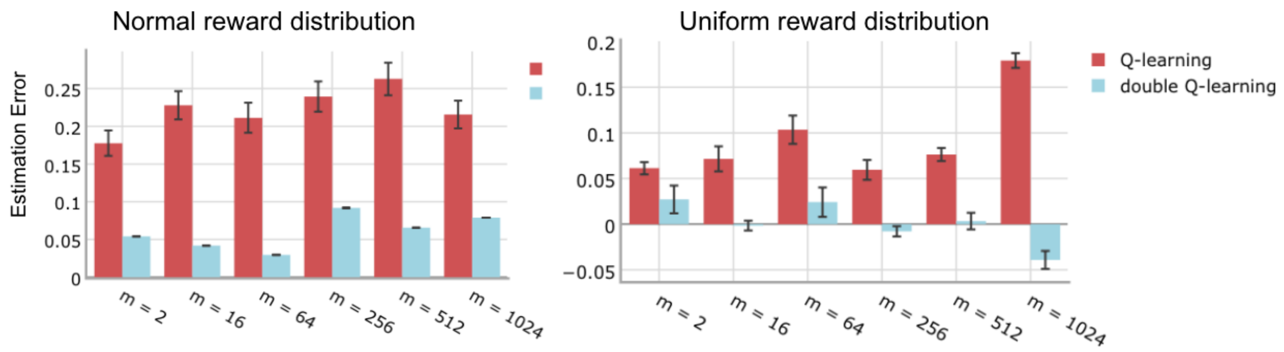


Figure 5: Comparison of evaluation errors for Q-Learning and Double Q-Learning in the MDP from Fig 1. Left: Reward distribution is same as in Fig 1, Right: Uniform reward distribution within $[-1, +1]$.

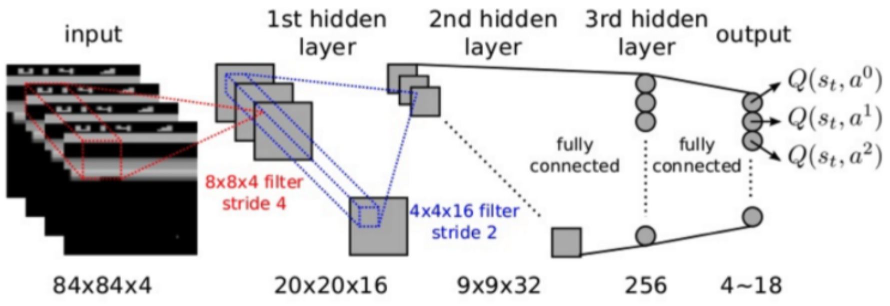
5.2 Bias due to function approximation

In this section, we investigated whether there are any estimation biases associated with functional approximation. Functional approximation is a very handy technique especially in environments that have large state and action spaces. Traditional methods use tabular methods to keep track of the values associated with a given state action pair. However, this because increasingly difficult because of the curse of dimensionality. Recent advances have proposed to use Neural Networks as functional approximators. Convolutional Neural networks (CNNs) have been shown to yield high performance in tasks such object classification, recognition, captioning etc. The central idea behind this proposal is that the CNNs can be used to approximate the state space by learning similarities among the states and thus help reduce the dimensionality of the state space in the environment. In this project, we worked with Deep Q Networks [2] which use convolutional neural networks as function approximators, and compare it's performance to Double Deep Q Networks [6] on Atari games Pool and Alien.

5.2.1 DQN

For our experiments, we use a convolutional neural network with three convolution layers (kernel size, stride: (8,4),(4,2),(3,1)) followed by a batch normalization layer. The output of the final convolution layer is then connected to a fully connected layer. The final layer outputs the state action values for the input state and all possible actions. Fig 6 shows the network architecture.

The inputs to the network are the images of the games sampled from the environments, and the outputs are the number of actions that are available for an agent in that particular environment. In short, the neural networks are playing the role of Q estimators mapping the policy from the state to the action space.



Source: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_folder_7/DQNBreakoutBlocks.png

Figure 6: Network architecture of the function appoximator used in DQN.

In the Deep Q Networks, two CNNs with the same architecture are used as Q network and Target network. The role of the target network is to store experiences in memory for the Q network to learn. This way the Q network learns from its own experiences and the associated rewards. The overall pipeline for training the DQN is as shown in Fig 7. The experience tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ sampled are pushed to the memory buffer and once it is full, experiences are sampled randomly from the buffer to train the Q network. The memory buffer is updated with new experiences constantly while flushing out the older experiences. Target network is updated sporadically (every 1000 steps), whereas Q network is updated each step.

$$\textbf{Target: } Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t)$$

$$\textbf{Update rule: } \theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t)$$

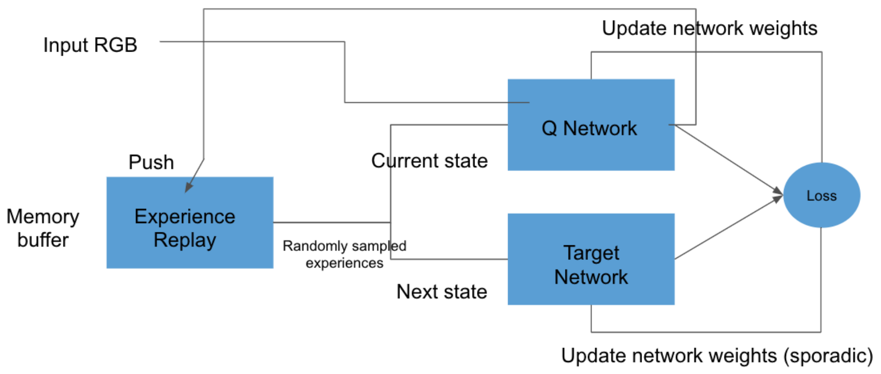


Figure 7: Overall pipeline for training DQN.

5.2.2 Double DQN

The network architecture is same as the DQN, however with one difference. In the DQNs, only the Q network alone was contributing to the learning of the state action space iteratively, while the target network was providing target state values from the past. The difference between the target and estimated values is backpropagated to update the weights of the Q network, and occasionally the target network. In double DQN, however, the state action values are obtained from both the Q and target networks. Just like in double Q learning, the two estimators used here are the Q and the target networks, while the target network continues to provide target experiences for the estimator to learn. The rest of the algorithm for updating weights is the same, the only difference being we use two different networks for calculating max and evaluating the Q value. For more details, see [6].

$$\textbf{Target: } Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t')$$

We trained the DQN and DDQNs on Atari game environments Pong and Alien (https://gym.openai.com/envs/#atari). To measure the estimation bias, we stored the q value associated with choosing the greedy action at every time step, and computed the average of such q values for 10,000 and 1,000 time steps for Pong and Alien respectively. The reasoning behind this analysis is as follows: in recursive estimation of the Q learning process, higher values at time 't' directly increase the value at 't+1' time point thus prompting the algorithm to choose the greedy action more frequently than is required. Ideally speaking the number of times the greedy actions are chosen is proportional to ϵ value in the ϵ-greedy algorithm. However, the results from figure 2 show that the number of times greedy action is chosen is far higher than the theoretical upper bound of choosing the greedy value (blackish gray line). Thus, even though we choose actions by exploiting and eploring as dictated by the probability ϵ, methods such as Q learning yield higher bias than desired. Therefore, quantifying the q value associated with greedy actions at every time step should provide us with a quantifiable metric for overestimation bias in DQN and DDQN learning algorithms.

We found that using the Double DQN formulation reduces the over estimatiion of greedy action values, as can be seen from the experiments we performed on Atari games Pong and Alien. This is true in spite of using the ϵ-greedy policy. The results are as shown in Fig 8 and Fig 9 below:

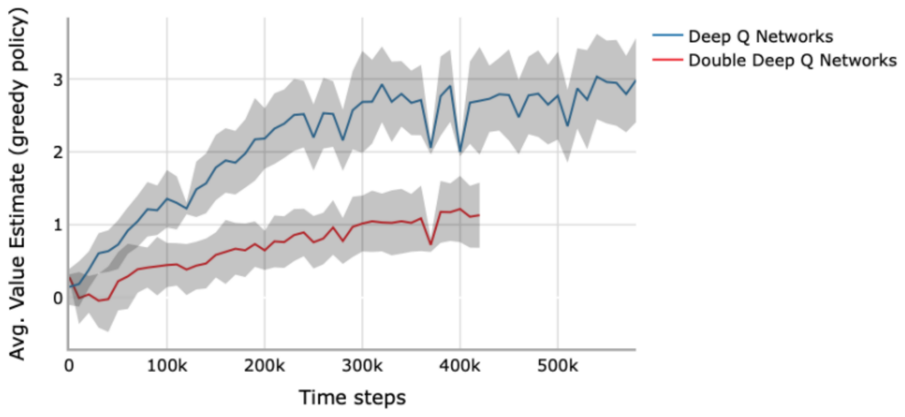


Figure 8: Estimation Bias for DQN vs Double DQN for Atari game Pool.

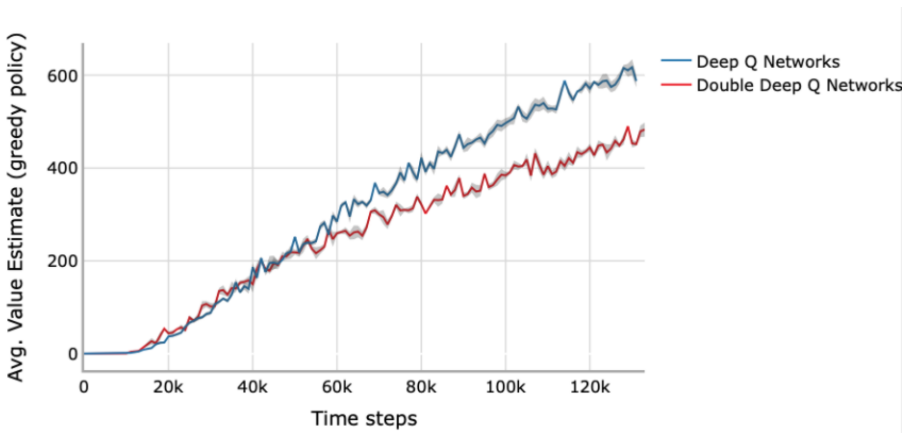
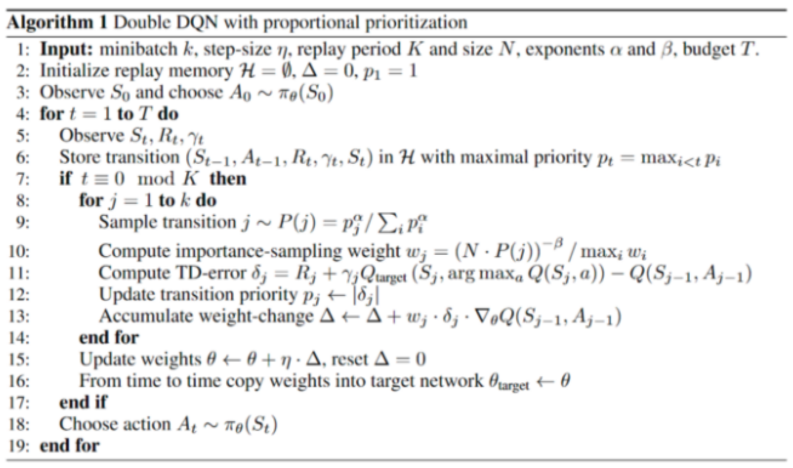


Figure 9: Estimation Bias for DQN vs Double DQN for Atari game Alien.

5.3 Estimation Bias due to experience sampling strategy

So far, in both DQN and DDQN, we have used the target networks to supply experiences to the policy (Q) network. These experiences are uniformly sampled from a memory buffer that stores all the past experiences upto a certain capacity (10000 experiences). Sampling randomly from these experiences can be a costly computation and also inefficient. Ideally, we would want to store the experiences that are more relevant (i.e. have a high / low reward for the same action performed for an approximately similar state). This can be implemented using priority experience replay sampling strategy [3]. Here each experience is prioritized by the associated loss function (i.e. the difference in the output estimate and the target). Each experience is then allocated a probability that is the function of the computed priority. This way, relevant experiences help the agent learn the environment faster. The algorithm is formally presented below. However, the estimation bias associated with the priority experience replay are not investigated, and to our knowledge are not explored. So we attempted to investigate the effects of the Priority Experience Replay (PER) on estimation bias. Our initial results suggest that in the Pong environment DDQNs with PER seem to have lower estimation bias compared to the uniform sampling strategy as can be seen from figure 10. We are still investigating this issue, and at the moment are handicapped by the limited run time on google colab. We intend to investigate this issue further in due time.



Algorithm 1 Double DQN with proportional prioritization

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

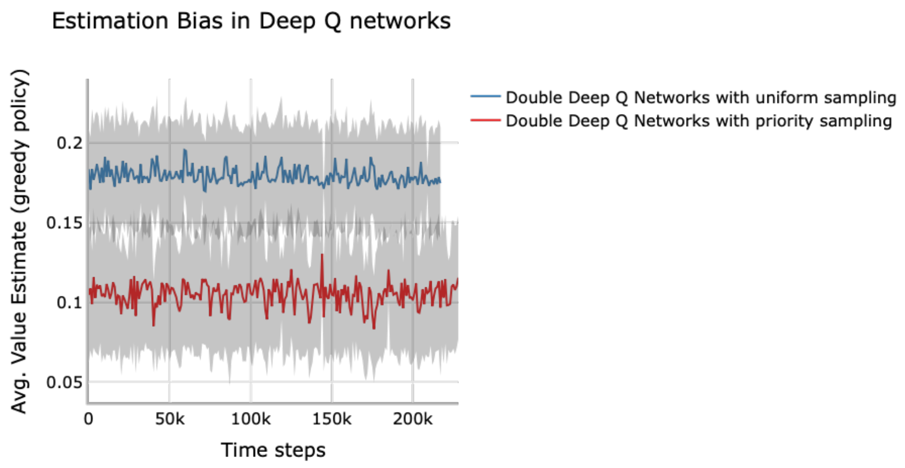


Figure 10: Estimation Bias for PER vs. Uniform Experience Replay for Atari game Pong using DDQN.

6 Conclusion

In this project, we demonstrated the overestimation bias associated with Q learning. We have listed out the potential factors that might contribute to this over estimation bias. These factors are both environmental (i.e. dependent on the number of states, stochasticity of the rewards obtained) and they also pertain during functional approximation of the state space. We have demonstrated this using DQN and DDQN architectures. Overall these results suggest that the overestimation bias in learning the state action space can be mitigated by using double Q learning (and double Deep Q learning) algorithms. In future, we would like to investigate the overestimation biases as a function of sampling strategy.

References

[1] H. Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010. 1, 2, 5

[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):529–533, 2015. 6, 7

[3] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 10

[4] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 1, 2, 5

[5] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993. 4

[6] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015. 3, 6, 7, 9

[7] C. J. C. H. Watkins. Learning from delayed rewards. 1989. 1, 2