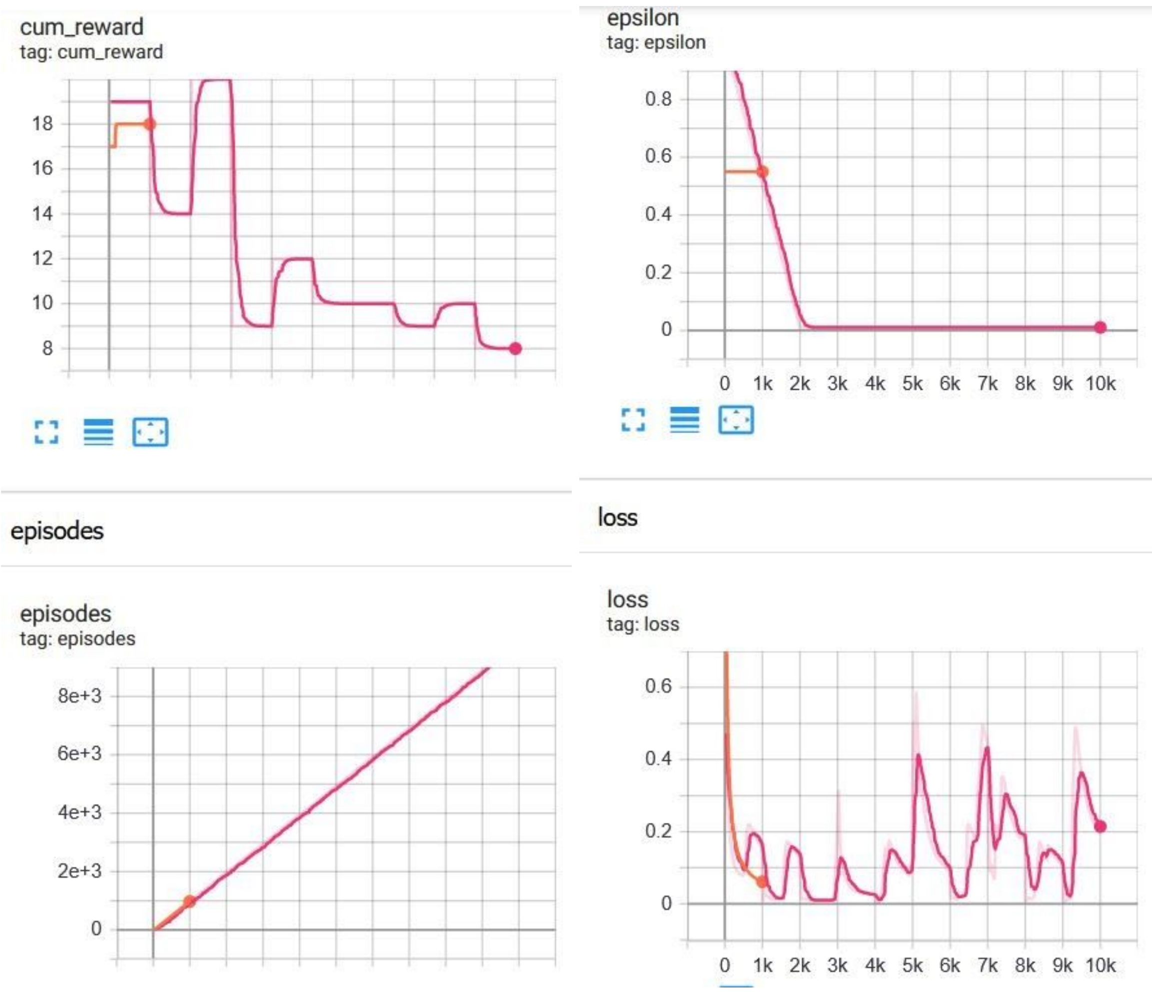
MT19213
Ghanendra Singh
DQN Simulation results.

1. **Case. When numSteps = 10,000 and decaystop = 2000, ep = 0.01**

```
trainer.train(numSteps=10000, decayStop=2000, endEpsilon=0.01)

Updated target network updates
```

Tensorboard smoothing 0.9

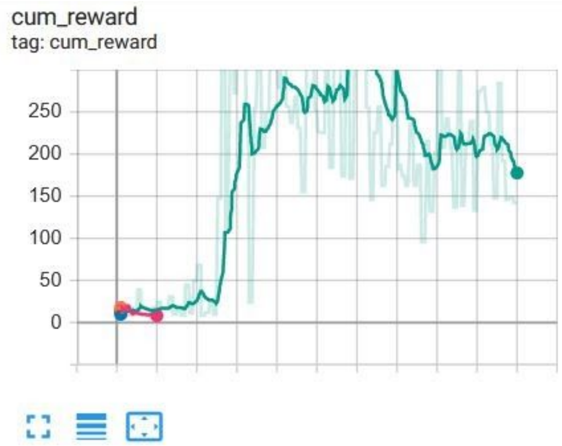


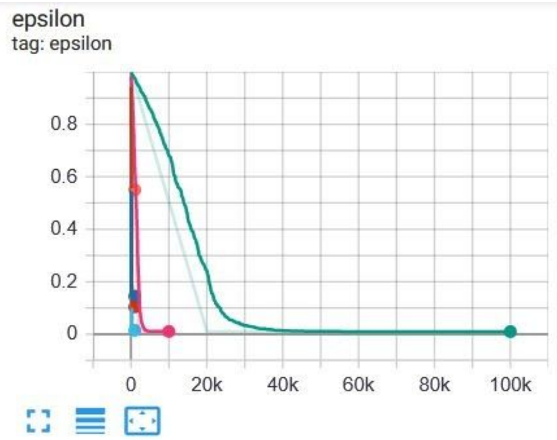
2. Case NumSteps = 10^5, decayStop = 20K

```
trainer.train(numSteps=100000, decayStop=20000, endEpsilon=0.01)
```

```
Updated target network updates
Updated target network updates
Updated target network updates
Updated target network updates
Updated target network updates
Updated target network updates
Updated target network updates
Updated target network updates
Updated target network updates
Updated target network updates
```

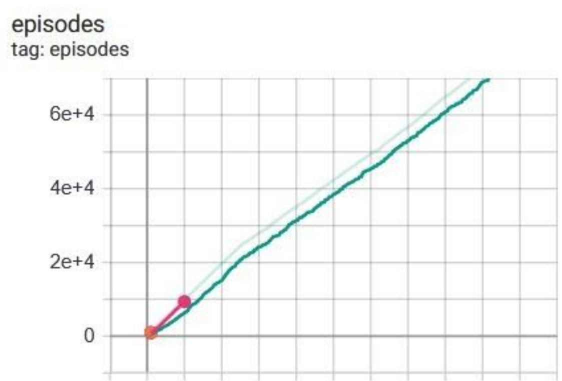
Tensorboard smoothing - 0.98

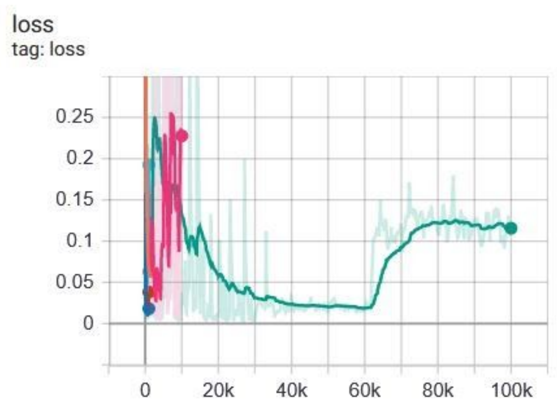




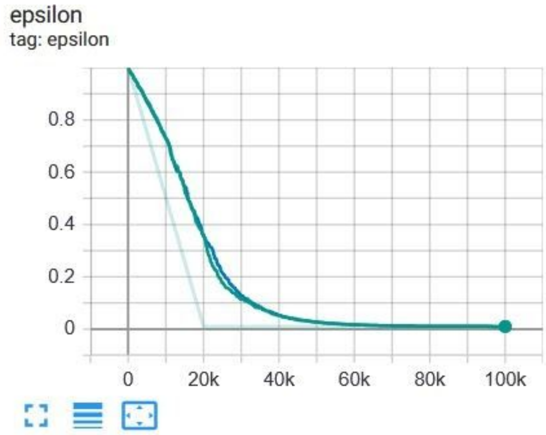
episodes

loss



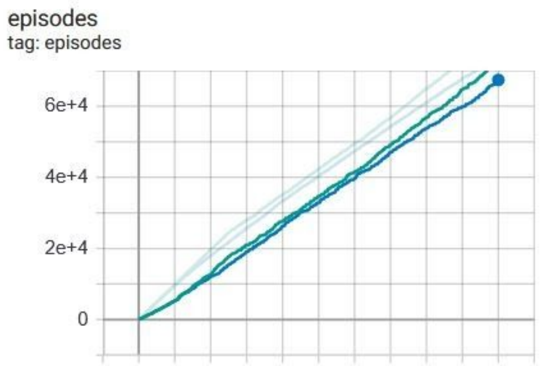


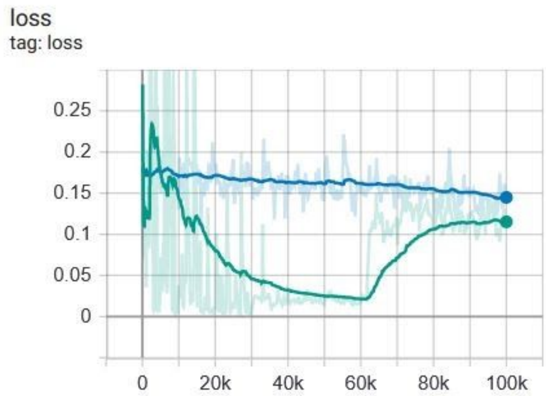
Q3. DDQN Simulation Results comparison with DQN.
When numSteps=100000, decayStop=20000
Tensorboard smoothing value 0.99 for removing the variations.

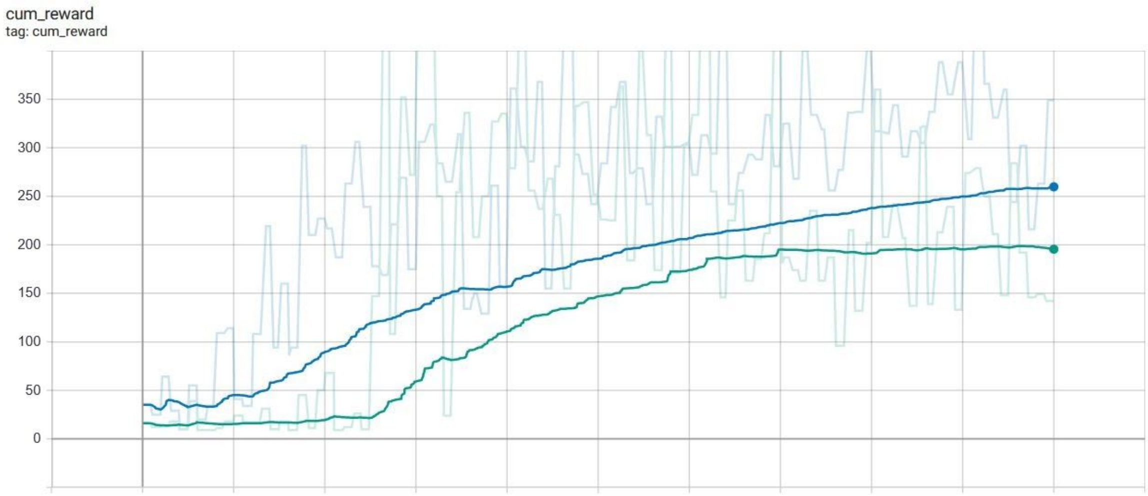


episodes

loss







Comparison.

1. Overestimates observed in the DQN case are reduced in DDQN case.
2. Here as the no. of steps are only 10^5, less difference is visible in the estimate values.
3. Action selection and evaluation is done based on the same values in DQN where as in DDQN two value functions are learned in which instead of Qa and Qb, the rules were modified.

OLD Implementation

Algorithm 1 Double Q-learning

Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

Pseudo-code Source: “Double Q-learning” (Hasselt, 2010)

NEW Implementation

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$