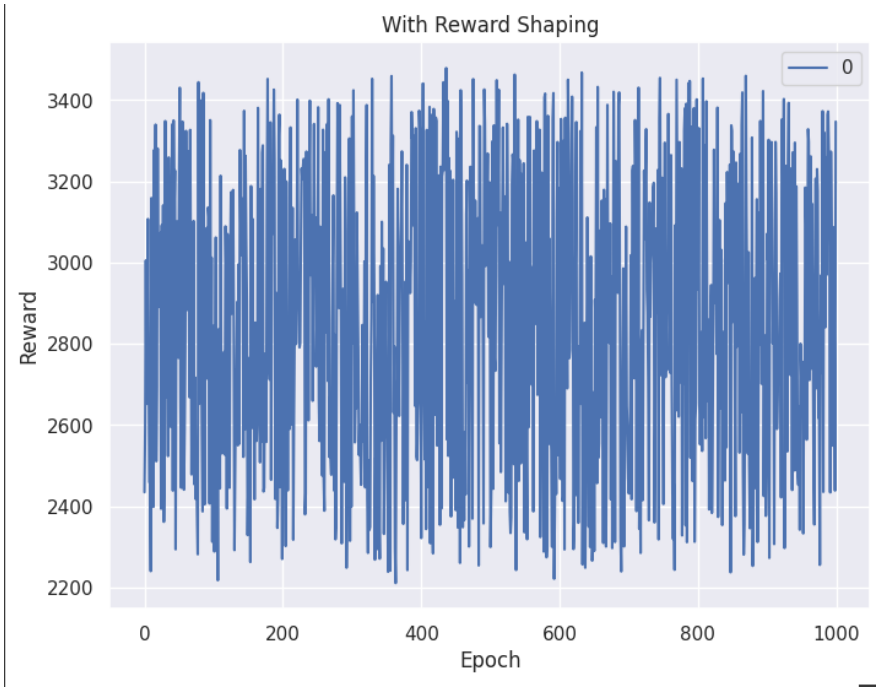
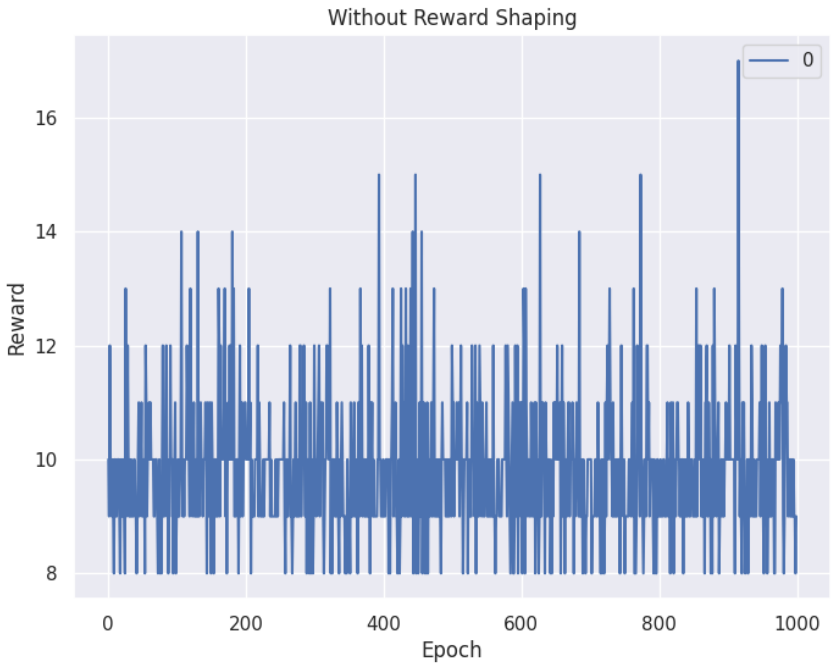
1. Plot the true reward (the reward returned by env.step()) collected against episode during training for both with and without reward shaping case. Note that you still need to plot the true reward in the reward shaping case too.
   For the DQN we have the following:





2. Does the performance improve from episode to episode? Did it overfit?

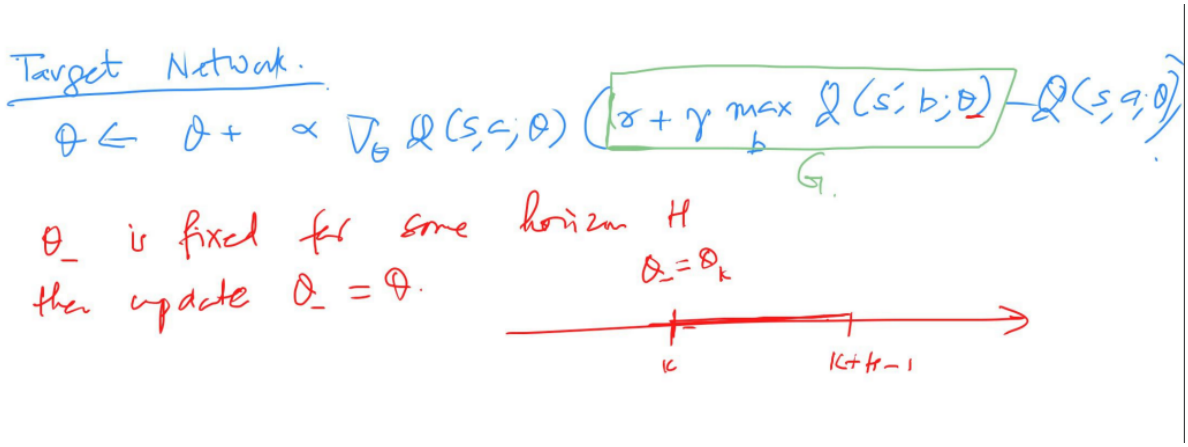
The performance is fluctuating and the Value Function is not chattering around the optimal Value function. This is due the phenomenon of deadly Triad, where we use

1. Linear Function Approximator Relu
2. Bootstrapping with Target Value Function
3. Off Policy Learning

The combination of these three is generally observed to not converge. The Graph is shown for 1000 epochs but a similar trend is seen when I worked with 10k epochs.

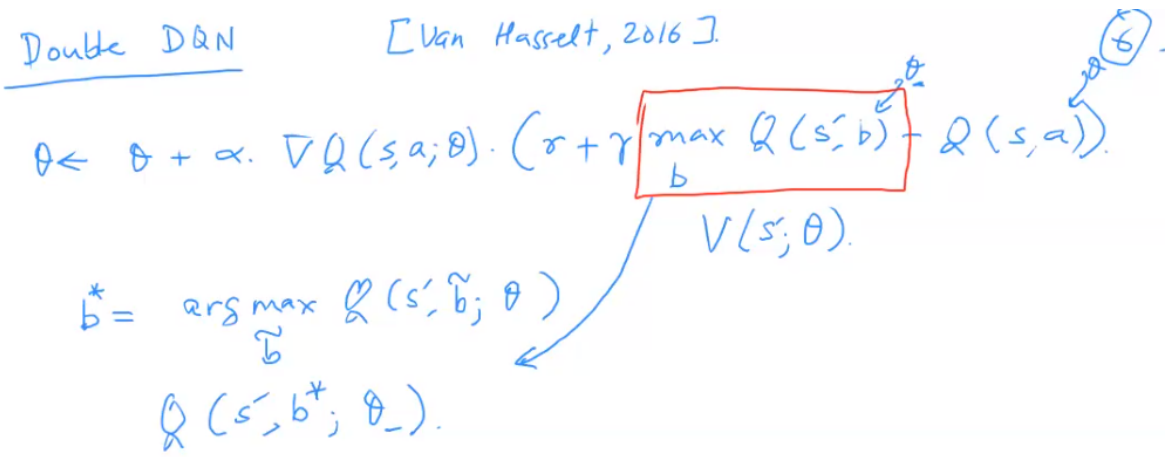
3. Why do we need two neural networks? Why do we need the fixed target network(model_ft)?

We generally assume iid data but that is generally not true therefore we use the Model_Ft as a baseline for calculating a loss function as we do not have a model or dynamics of the system and therefore it is tough to evaluate our performance. So by taking a Bootstrapping approach of introducing a Target Network and not changing it for the current epoch we suppose that we will reach the optimal value function, we try to implement a kind of supervised learning. By not keeping the model_ft fixed we might diverge from the convergence as the target will continuously change.



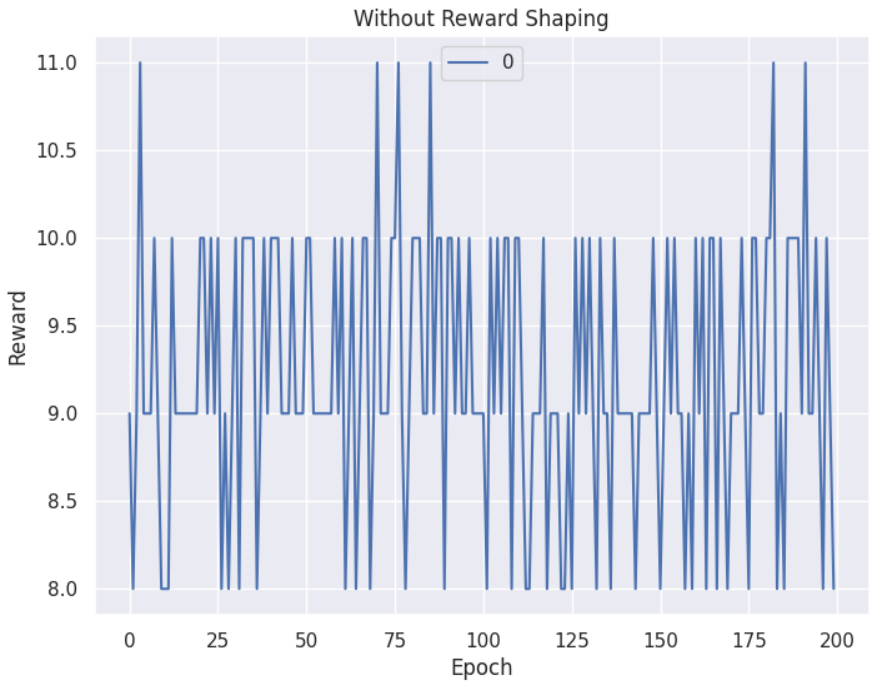
4. How would you modify the above DQN method to double DQN method?

The Algorithm is as follows:



We can implement the Double DQN by keeping two networks, one for action and one for evaluation. This is used to remove the bias that we might have been having previously by using the same network.

The doubleDQN seems to perform much better than DQN and it has better convergence as shown in the diagram after the Reward shaping. It seems to oscillate around the Optimal Value Function. It is therefore not overfitting. The performance also seems to improve. Also I have shown 200 epochs, but in my previous trials where I used 10k epochs the performance seemed to really improve with time.
These are the plots for DoubleDQN.



5. The above Code did not use experience replay. What will be the benefit of using experience reply?

An experience replay buffer is just an array of transitions (st,at,rt,st+1,dt) that we store while traversing the MDP. Then at each timestep, instead of updating θ according to the most recent transition, we instead sample a batch of training data from the buffer and use that instead. Replay buffers can be used to mitigate catastrophic forgetting. This is a phenomenon where updating θ to minimize the loss L(θ) decreases the loss on the current training examples, but actually increases the loss on older training examples, effectively unlearning what the network learned in the past. (Same answer for DQN and DoubleDQN).