**Exercise 3 – Deep Q Learning**

Code completion:

DQN.py:

1) *TODO: build the target network and set its weights to policy\_net's wights (use state\_dict from pytorch)*

target\_net = Network(network\_params, device).to(device)  
target\_net.load\_state\_dict(policy\_net.state\_dict())  
target\_net.eval()

2) *TODO: implement action selection.*

if random.random() < epsilon:  
 return torch.tensor([[random.randrange(network\_params['action\_dim'])]], device=device, dtype=torch.long)  
else:  
 with torch.no\_grad():  
 return torch.tensor([[policy\_net(s).max(1)[1].view(1,1)]], device=device, dtype=torch.long)

3) *TODO: fill curr\_Q*

curr\_Q = policy\_net(state\_batch).gather(1, action\_batch)

4) *TODO: fill expected\_Q*

next\_state\_vals = torch.zeros(params.batch\_size)  
try:  
 not\_last\_states = torch.cat([state\_ for ind, state\_ in enumerate(batch.next\_state) if not\_done\_batch[ind].item() is not False])  
 if params.target\_update == 'none':  
 next\_state\_vals[not\_done\_batch] = policy\_net.forward(not\_last\_states).max(1)[0].detach()  
 else:  
 next\_state\_vals[not\_done\_batch] = target\_net.forward(not\_last\_states).max(1)[0].detach()  
  
except:  
 print('there are no non last states')  
  
expected\_Q = reward\_batch + (params.gamma \* next\_state\_vals)  
expected\_Q = expected\_Q.view(params.batch\_size, 1)

5) *TODO: Implement soft target update.*

for target\_param, policy\_param in zip(target\_net.parameters(), policy\_net.parameters()):  
 target\_param.data.copy\_(params.tau \* policy\_param.data + (1.0 - params.tau) \* target\_param.data)

6) *TODO: Implement hard target update.*

if i\_episode % params.target\_update\_period == 0:  
 target\_net.load\_state\_dict(policy\_net.state\_dict())

7) *TODO: store weights*

if params.target\_update in ['soft', 'hard']:  
 torch.save(target\_net.state\_dict(), experiment\_name + '\_best.dat')  
else:  
 torch.save(policy\_net.state\_dict(), experiment\_name + '\_best.dat')

buffer.py:

1. todo push

def push(self, \*args):  
 *"""Saves a transition."""* # TODO  
 transition = Transition(args[0],args[1],args[2],args[3],args[4])  
 if self.\_\_len\_\_() >= self.capacity:  
 self.memory = self.memory[1:]  
 self.memory.append(transition)

1. todo sample

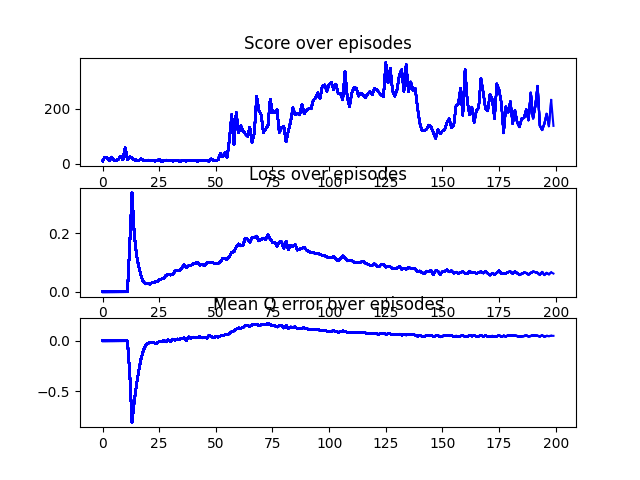
def sample(self, batch\_size):  
 # TODO  
 sample = random.sample(self.memory,batch\_size)  
 return sample

**second part - Effect of Replay Buffer and Target Network**

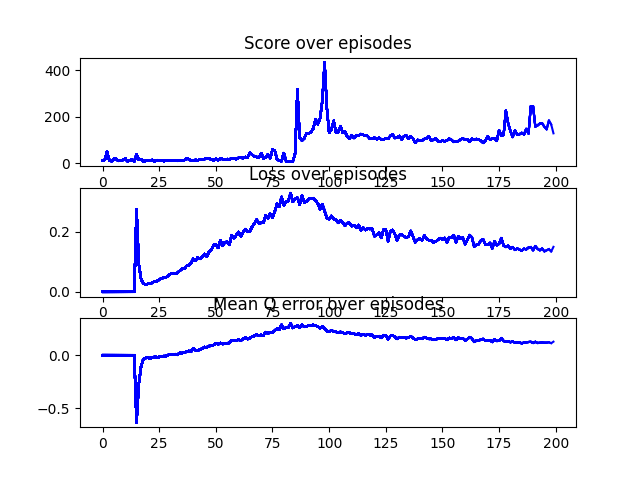
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target update | With replay with target | With replay without target | Without replay  with target | Without replay  without target |
| Hard | 226.6 | x | 20.2 | x |
| None | x | 23.8 | x | 28.2 |
| Soft | 256.4 | x | 23.4 | x |

**With replay and soft target update:**

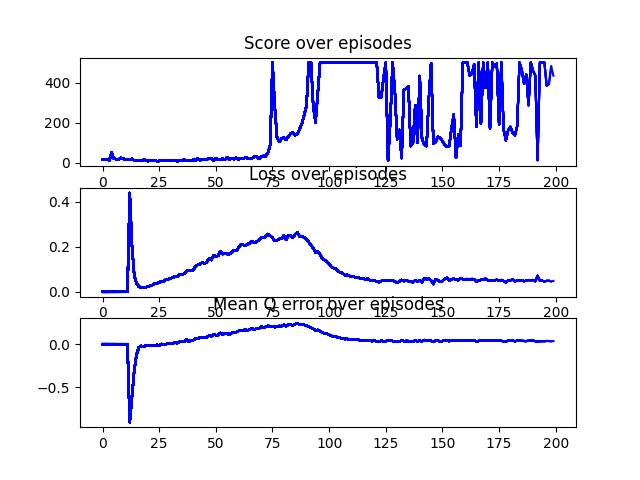
Experiment 1:



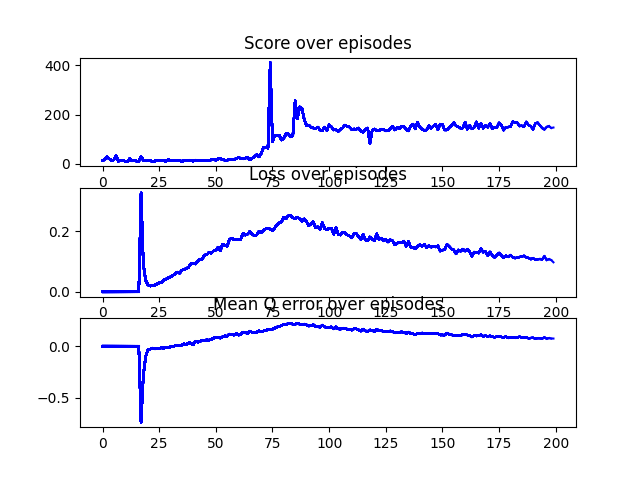
Experiment 2:



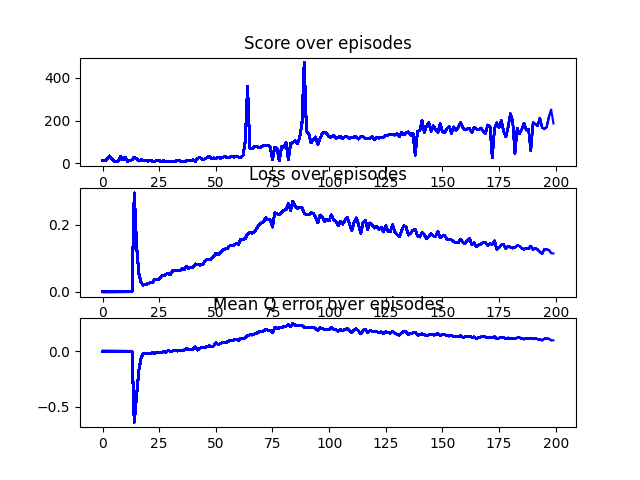
Experiment 3:



Experiment 4:



Experiment 5:



Explanation:

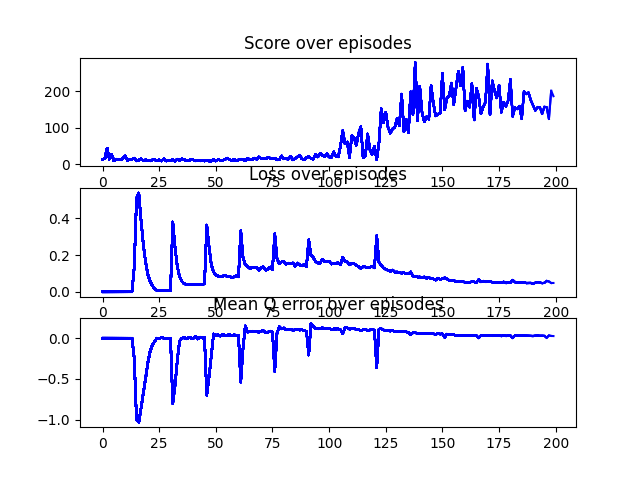
This worked best for me - the max score was 500 in one of the experiments.

In most of the experiments except in experiment 3 we see the same pattern in the loss and mean Q error, it increases constantly until around episode 100 and then changes direction and constantly decreasing.

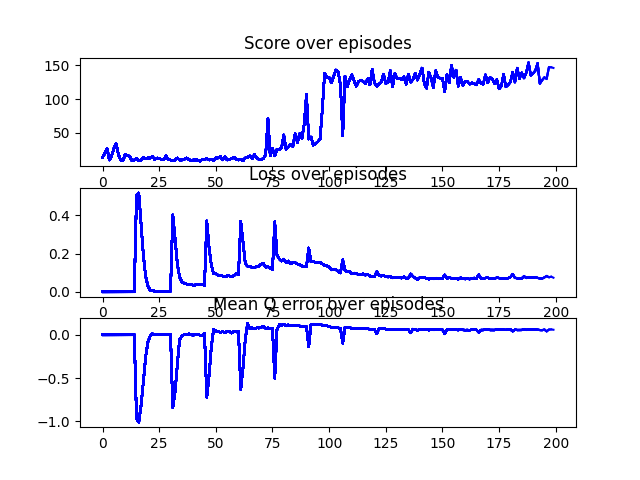
The reason for this behavior is and the decreasing in the loss and mean Q error after about 100 episodes is that the weights updated towards the optimal weights meaning taking better actions which maximizes discounted future rewards.

The constant direction of the increase and decrease parts is due to the small and constant changes (after each episode) we do to the target network and the difference between the policy and the target networks is relatively small.

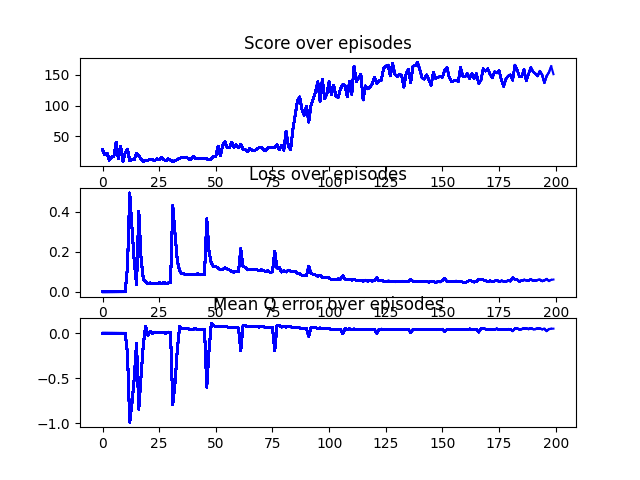
**With replay and hard target update:**

Experiment 1:

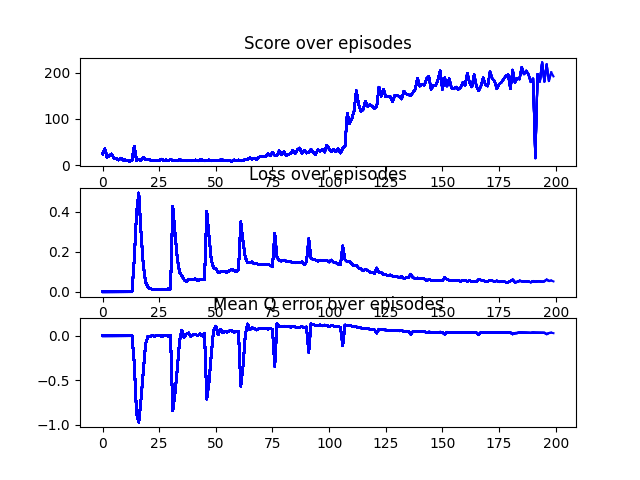
Experiment 2:



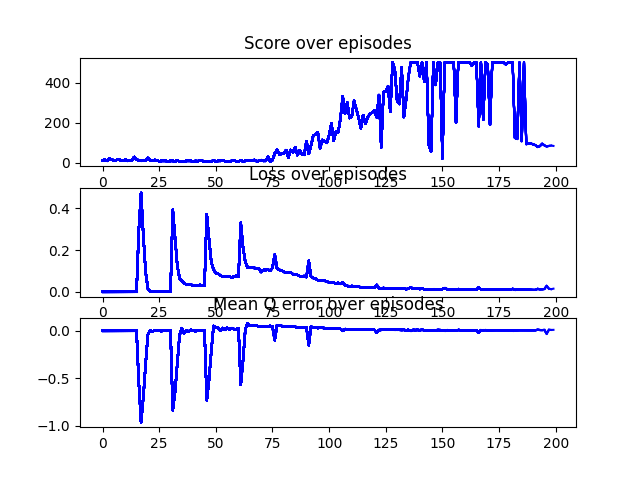
Experiment 3:



Experiment 4:



Experiment 5:



Explanation:

This was the second best setup.

We see in all the experiments the same behavior in the loss and mean Q error – it increases

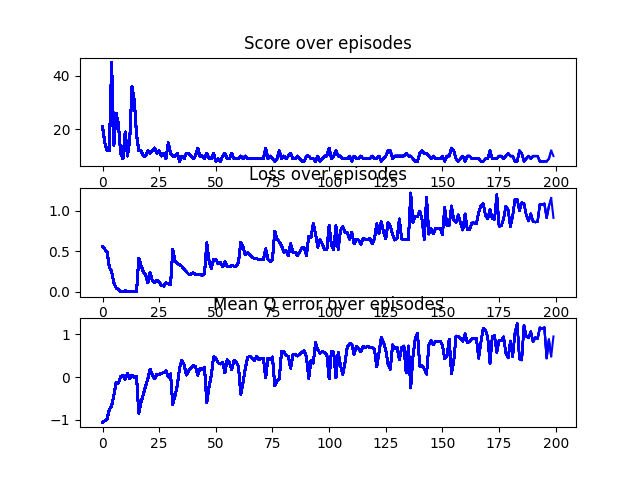
and decreases in cycles where each cycle (update after k episodes) the loss and mean Q error decreases.

These large errors are caused due to the gap in the updates between the policy and target networks. The policy network is updated every episode while the target network updated after 15 episodes and the pick at each cycle is observed after 15 episodes until the networks converge and each update step becomes very minor.

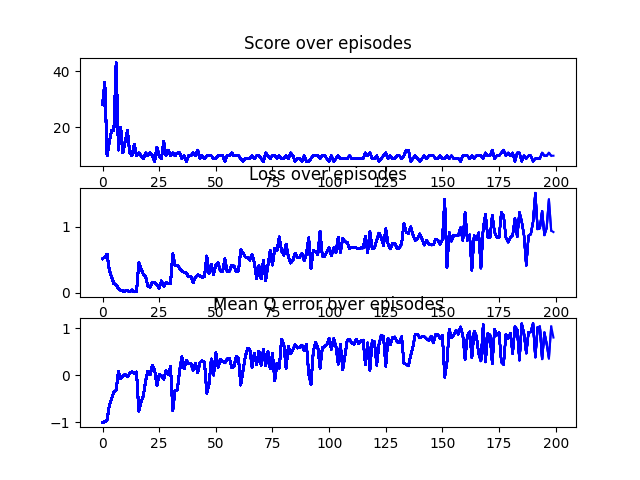
From about episode 125 we see that the networks converged and there value of the loss and mean Q error is stable.

**Without replay with target hard update:**

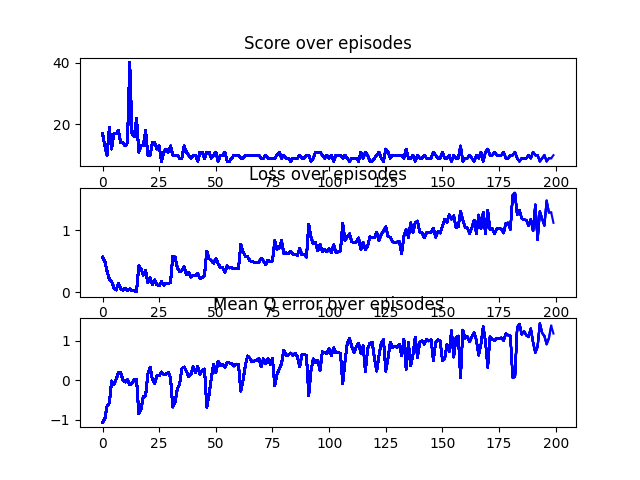
Experiment 1:

****

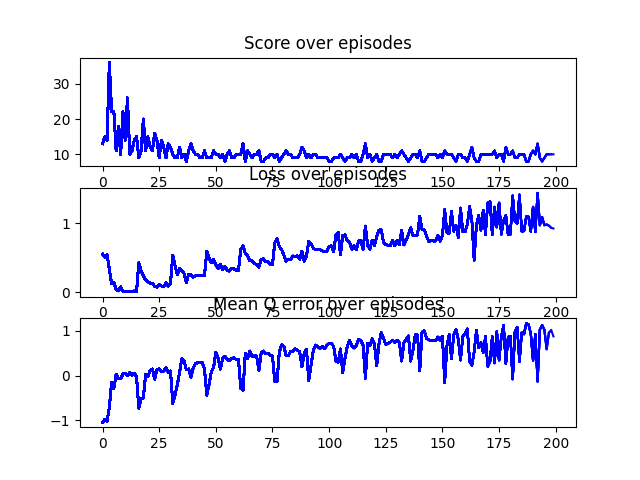
Experiment 2:



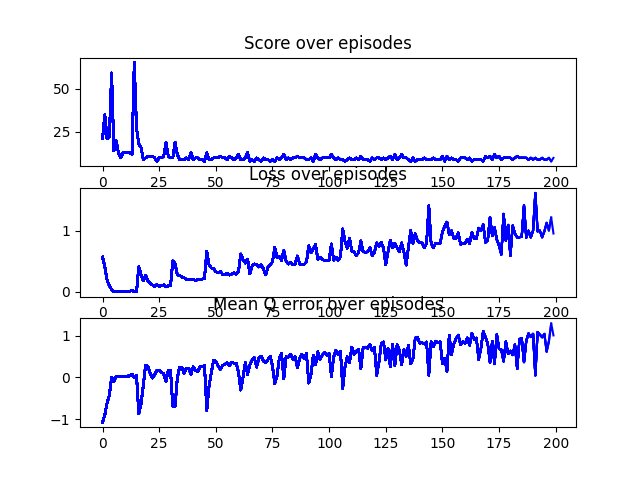
Experiment 3:



Experiment 4:



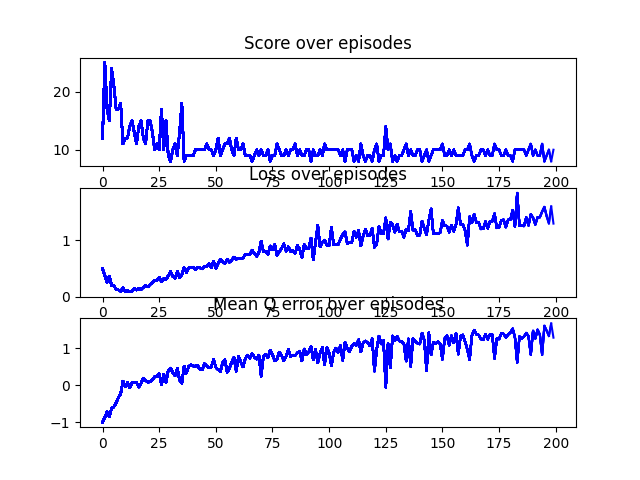
Experiment 5:



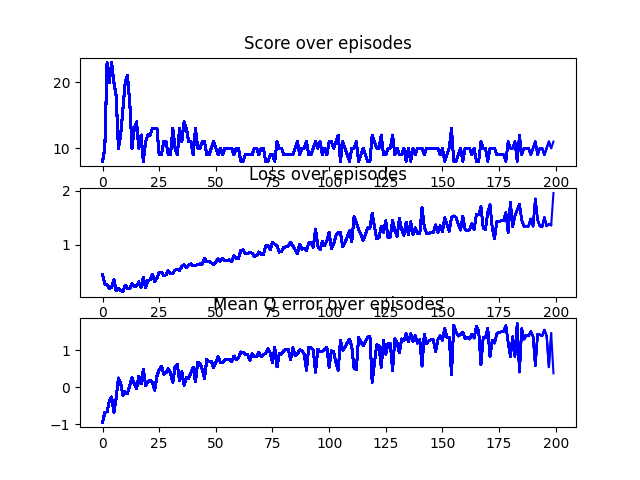
Explanation for this setup is combined with the rest 3 setups at the end.

**Without replay with target soft update:**

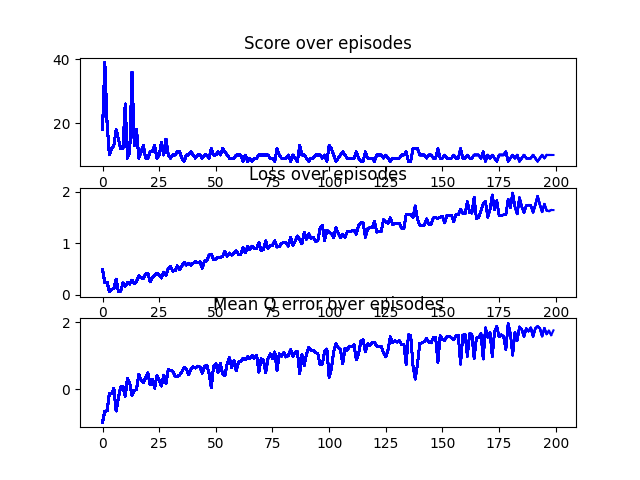
Experiment 1:



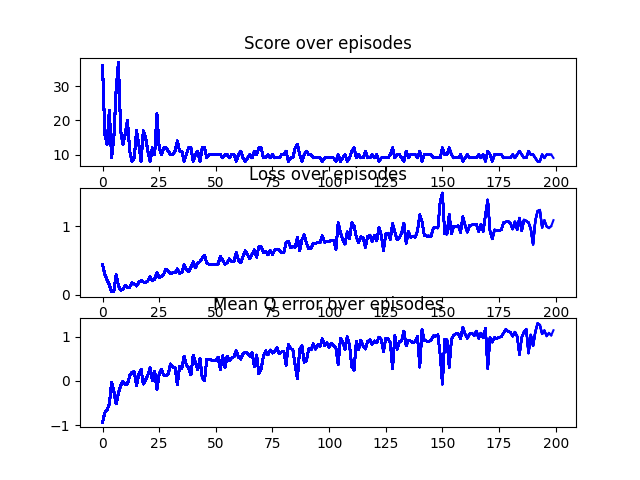
Experiment 2:



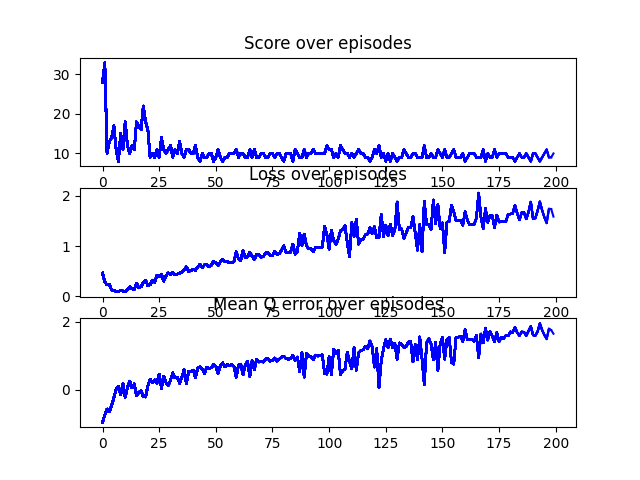
Experiment 3:



Experiment 4:



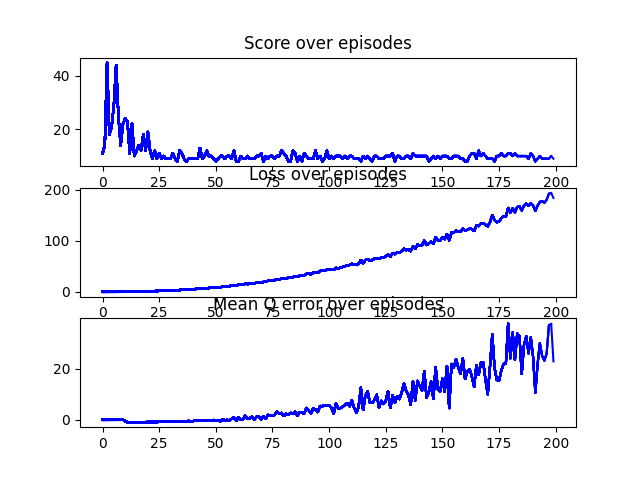
Experiment 5:



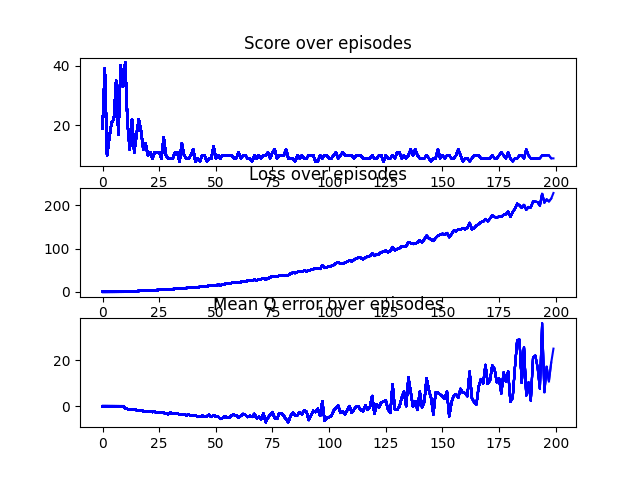
Explanation for this setup is combined with the rest 3 setups at the end.

**With replay without target:**

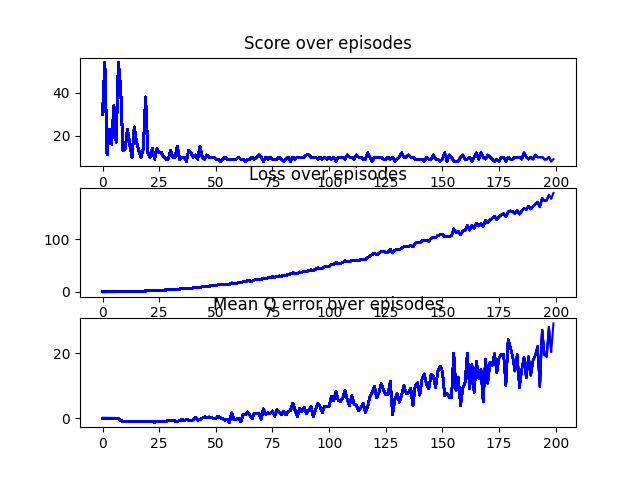
Experiment 1:



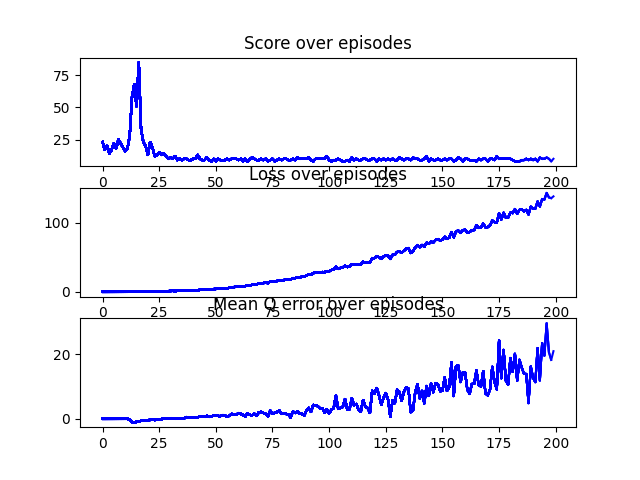
Experiment 2:



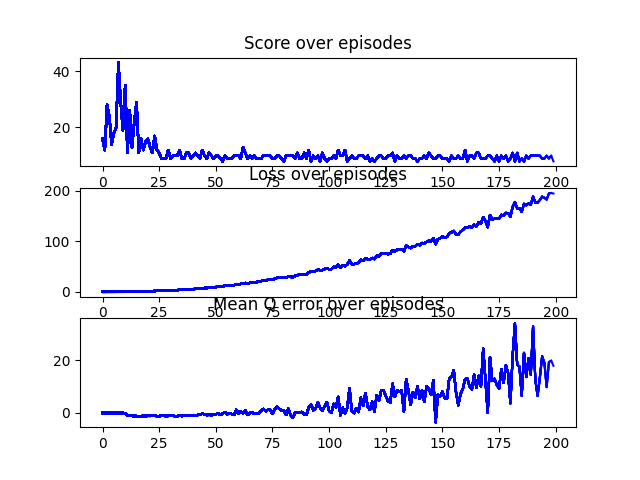
Experiment 3:



Experiment 4:



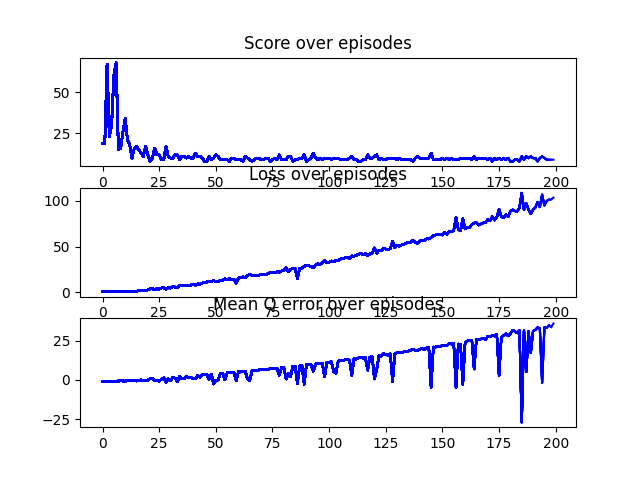
Experiment 5:



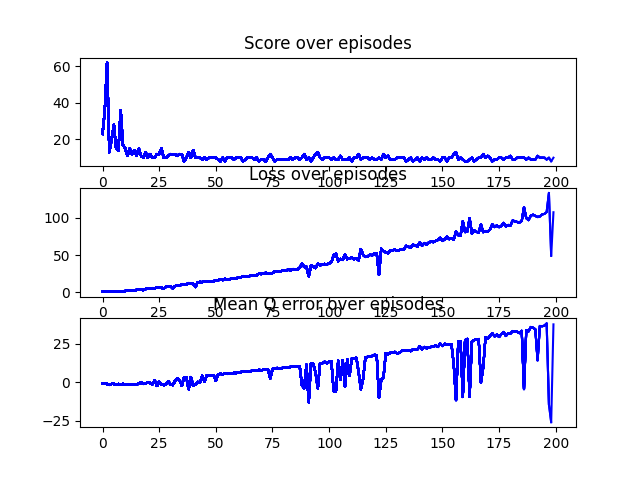
Explanation for this setup is combined with the rest 3 setups at the end.

**Without replay without target:**

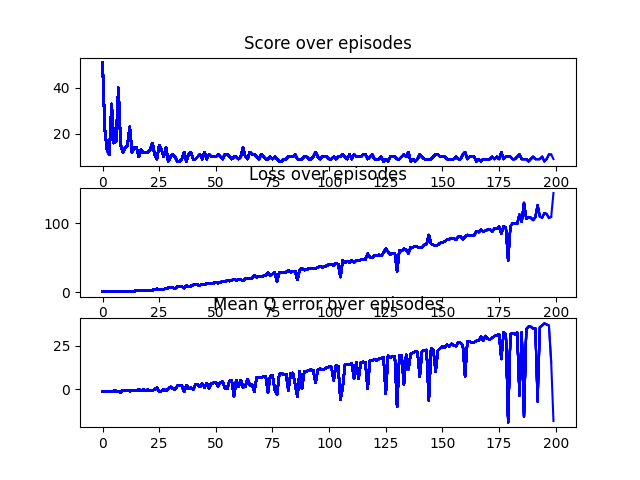
Experiment 1:



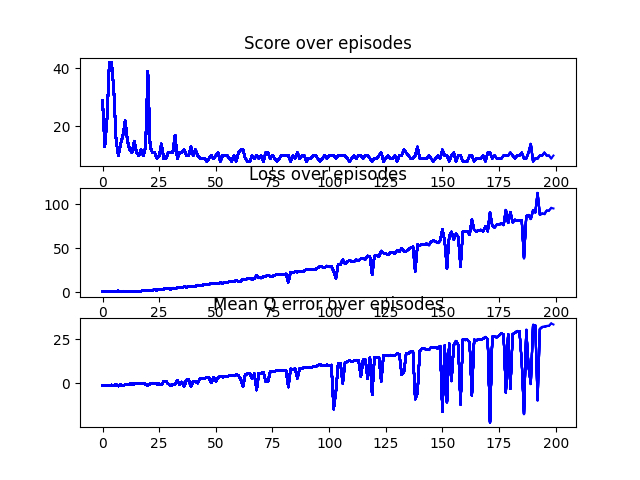
Experiment 2:



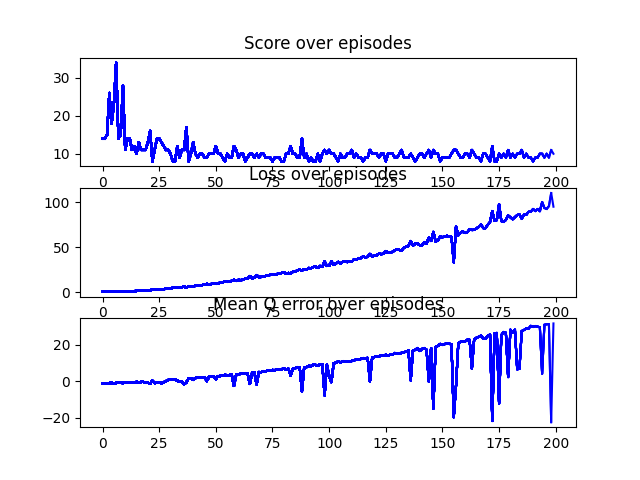
Experiment 3:



Experiment 4:



Experiment 5:



Explanation for the following setups – without replay with hard target update/ without replay with soft target update/ without replay without target/ with replay without target:

We observe the same pattern for all 4 setups while the only difference is the range of the loss and mean Q error.

There is a constant increase in the loss and the mean Q error in all the experiments and it means that the networks didn't converge.

We infer that to achieve convergence we in both target network and replay buffer and the effect of each one of them is almost the same as we can see the results are almost identical in all 4 setups and increases dramatically only when both applied as we see in the first 2 setups.