强化学习 HW4

— 饶翔云 520030910366

Problem 1

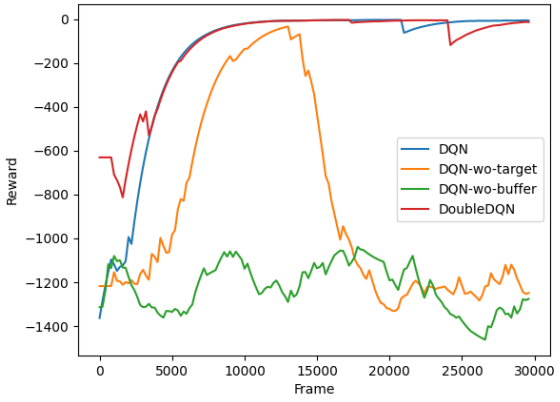
Coding:

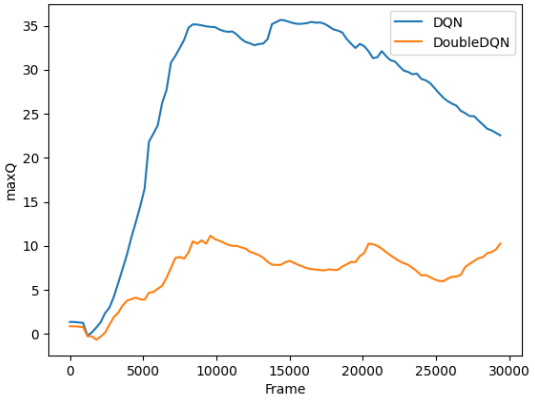
```
if USE_DOUBLE_Q:
        """ ------------- Programming 1:  ------------- """
        # next_q_value = """ YOUR CODE HERE """
        action = torch.max(next_q_values, 1)[1].unsqueeze(1)
        next_q_value = next_q_values_target.gather(1, action).squeeze(1)
        """ ------------- Programming 1 ------------- """
    else:
        if USE_TARGET_NET:
            """ ------------- Programming 2:  ------------- """
            # next_q_value = """ YOUR CODE HERE """
            action = torch.max(next_q_values_target, 1)[1].unsqueeze(1)
            next_q_value = next_q_values_target.gather(1, action).squeeze(1)
            """ ------------- Programming 2 ------------- """
        else:
            action = torch.max(next_q_values, 1)[1].unsqueeze(1)
            next_q_value = next_q_values.gather(1, action).squeeze(1)

    """ ------------- Programming 3:  ------------- """
    # expected_q_value = """ YOUR CODE HERE """
    # 根据bellman方程计算
    expected_q_value = reward + gamma * next_q_value * (1 - done)
    """ ------------- Programming 3 ------------- """
```

```
""" ------------- Programming 5: ---------- """
        """ YOUR CODE HERE """
        # 将状态、动作、奖励、下一个状态、是否结束标志放入一个元组并放入buffer中
        self.buffer.append((state, action, reward, next_state, done))
        """ ------------- Programming 5 ------------- """
```

```
""" YOUR CODE HERE """
        # 初始化状态、动作、奖励、下一个状态、是否结束标志
        batch_state = np.zeros((batch_size, 3))
        batch_action = []
        batch_reward = []
        batch_next_state = np.zeros((batch_size, 3))
        batch_done = []
        # 按照batch_size的大小从buffer中随机抽取样本
        Sample = random.sample(self.buffer, batch_size)
        # 将抽取的样本分别放入对应的列表中
        for i in range(batch_size):
            batch_state[i] = Sample[i][0]
            batch_action.append(Sample[i][1])
            batch_reward.append(Sample[i][2])
            batch_next_state[i] = Sample[i][3]
            batch_done.append(Sample[i][4])

        return batch_state, batch_action, batch_reward,
    batch_next_state, batch_done
```

Results:





Answers:

(a)

根据我对图1的观察，DQN的表现明显优于DQN-wo-target，且随着frame增大，表现差异越来越大，并且在30000帧内DQN的表现呈现上升趋势而DQN-wo-target呈现下降趋势。我认为背后的原因是由于DQN-wo-target使用current-model进行选取动作和Q估值，导致网络参数更新过于频繁，产生混乱，无法克服过高估计，并最终导致结果差。可以说，双网络的策略克服了单网络难以收敛的问题，使得模型效果更好。

(b)

根据我对图1的观察，DQN的表现明显优于DQN-wo-buffer。这个原因在于DQN通过使用buffer实现了经验回放，充分利用了之前的经历，使得训练更加高效，模型更富有远见，而DQN-wo-buffer只能使用每次交互得到的经验来训练，从而效果优于DQN-wo-buffer。

(*c*)

根据我对图1图2的观察，DDQN的表现在20000帧后开始优于DQN，而且maxQ始终保持在一个合理的范围内。这是由于两个网络之间选取动作计算Q值的方法不同。DDQN通过使用当前网络选取动作，用评估网络计算Q值的方法将选取动作和计算Q值解耦。而DDQN由于每次选择的根据是当前Q网络的参数，并不是像DQN那样根据target-Q的参数，所以当计算target值时是会比原来小一点的。（因为计算target值时要通过target-Q网络，在DQN中原本是根据target-Q的参数选择其中Q值最大的action，而现在用DDQN更换了选择以后计算出的Q值一定是小于或等于原来的Q值的）这样在一定程度上降低了overestimation，使得Q值更加接近真实值。）从而克服了过高估计的问题，使模型进一步优于DQN。