# 强化学习：作业三

丁云翔 191250026

2021 年 12 月 17 日

## 1 作业内容

在 gym Atari 环境中实现 DQN 算法及其变体。

## 2 实现过程

补全了 atari_ddqn.py 中的 learning 方法（该文件改名为 atari_dqn.py），使 DQN 算法能够运行，添加后的 learning 方法如下：

```python
def learning(self, fr):
    s0, s1, a, r, done = self.buffer.sample(self.config.batch_size)

    s0 = torch.tensor(s0, dtype=torch.float)
    s1 = torch.tensor(s1, dtype=torch.float)
    a = torch.tensor(a, dtype=torch.long)
    r = torch.tensor(r, dtype=torch.float)
    done = torch.tensor(done, dtype=torch.float)

    if self.config.use_cuda:
        s0 = s0.float().to(self.config.device)/255.0
        s1 = s1.float().to(self.config.device)/255.0
        a = a.to(self.config.device)
        r = r.to(self.config.device)
        done = done.to(self.config.device)

    # How to calculate Q(s,a) for all actions
    # q_values is a vector with size (batch_size, action_shape, 1)
    # each dimension i represents Q(s0,a_i)
    q_values = self.model(s0).to(self.config.device)
    next_q_values = self.model(s1).to(self.config.device)

    # How to calculate argmax_a Q(s,a)
    actions = q_values.max(1)[1]

    # Tips: function torch.gather may be helpful
    # You need to design how to calculate the loss
    q_value = q_values.gather(1, a).squeeze(1)
    expected_q_value = r + self.config.gamma * next_q_values.max(1)[0] * (1 - done)

    loss = (q_value - expected_q_value.detach()).pow(2).mean()

    self.model_optim.zero_grad()
    loss.backward()
    self.model_optim.step()

    if fr % self.config.update_tar_interval == 0:
        self.target_model.load_state_dict(self.model.state_dict())
    return loss.item()
```



复制一份 atari_dqn.py，并命名为 atari_ddqn.py，修改目标函数的计算公式，使 DDQN 算法能够运行，修改后的 learning 方法如下：

```python
def learning(self, fr):
    s0, s1, a, r, done = self.buffer.sample(self.config.batch_size)

    s0 = torch.tensor(s0, dtype=torch.float)
    s1 = torch.tensor(s1, dtype=torch.float)
    a = torch.tensor(a, dtype=torch.long)
    r = torch.tensor(r, dtype=torch.float)
    done = torch.tensor(done, dtype=torch.float)

    if self.config.use_cuda:
        s0 = s0.float().to(self.config.device)/255.0
        s1 = s1.float().to(self.config.device)/255.0
        a = a.to(self.config.device)
        r = r.to(self.config.device)
        done = done.to(self.config.device)

    # How to calculate Q(s,a) for all actions
    # q_values is a vector with size (batch_size, action_shape, 1)
    # each dimension i represents Q(s0,a_i)
    q_values = self.model(s0).to(self.config.device)
    next_q_values = self.model(s1).to(self.config.device)
    next_q_target_values = self.target_model(s1).to(self.config.device)

    # How to calculate argmax_a Q(s,a)
    actions = q_values.max(1)[1]

    # Tips: function torch.gather may be helpful
    # You need to design how to calculate the loss
    q_value = q_values.gather(1, a).squeeze(1)
    next_q_value = next_q_target_values.gather(1, next_q_values.max(1)[1].unsqueeze(1)).squeeze(1)
    expected_q_value = r + self.config.gamma * next_q_value * (1 - done)

    loss = (q_value - expected_q_value.detach()).pow(2).mean()
```

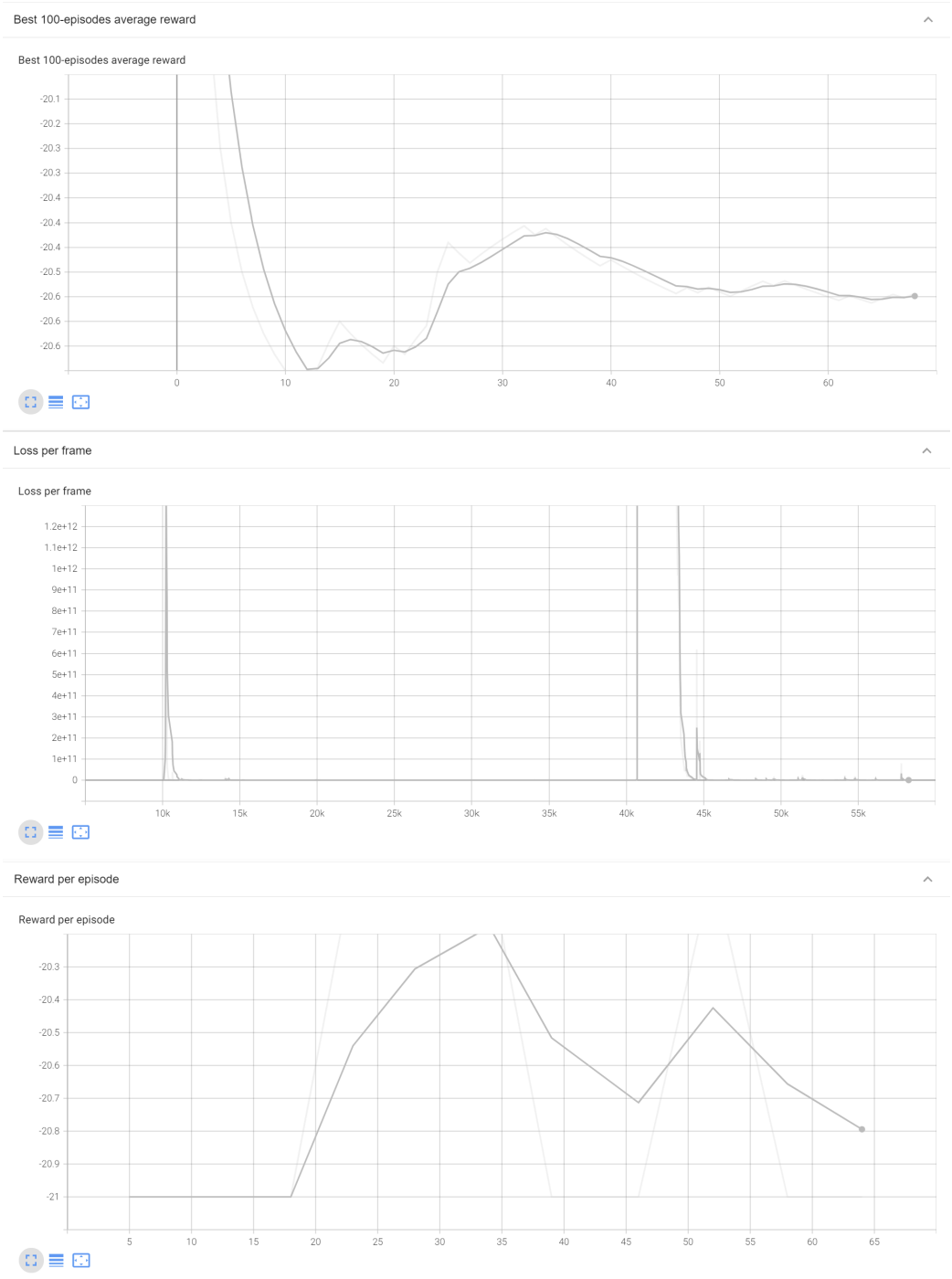
# 3 复现方式

复现 DQN: 在主文件夹下运行 `python atari_dqn.py --train`.
复现 DDQN: 在主文件夹下运行 `python atari_ddqn.py --train`.

# 4 实验效果

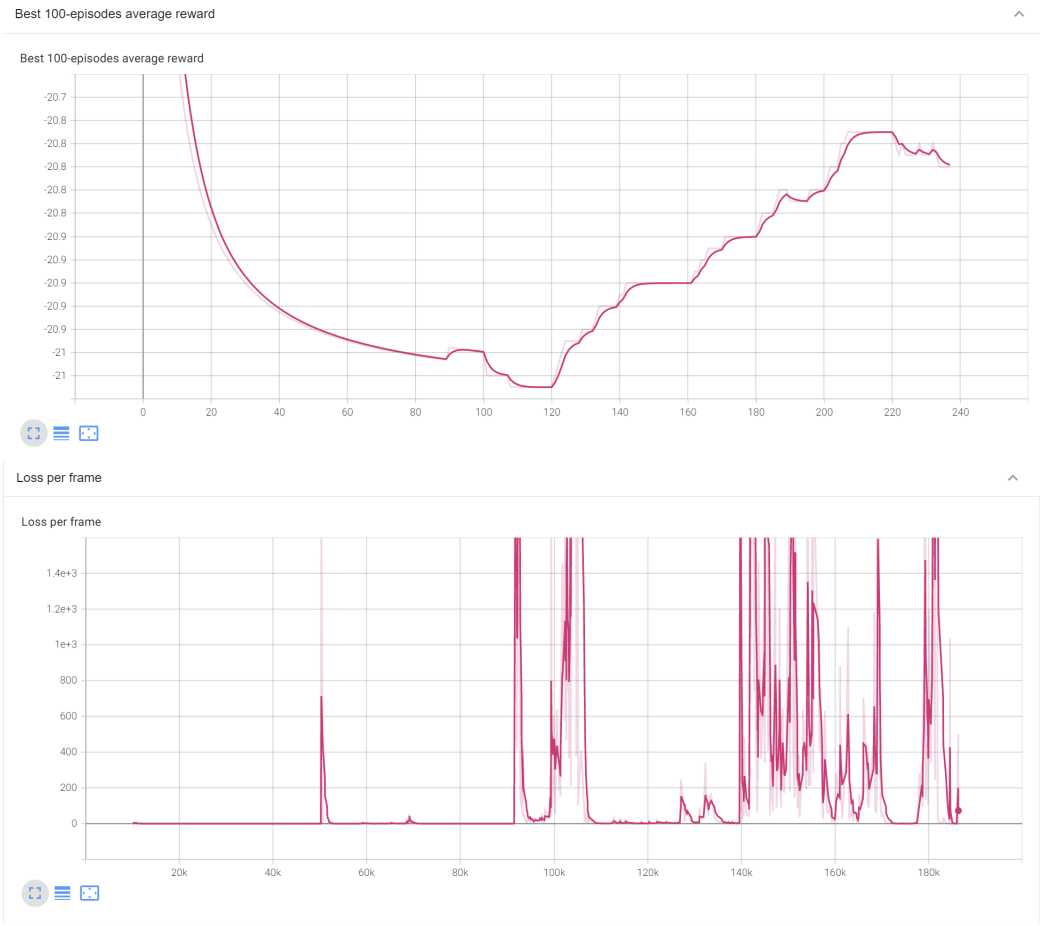
描述累计奖励和样本训练量之间的关系。

DQN：累计奖励随样本训练量增加在最低值附近波动，不知道是算法存在 bug 还是轮数太少还未收敛，因为每次跑到几万帧之后由于我自己的设备内存不足就跑不下去了，所以暂时还无法验证。这里取了坚持运行轮数最多的一次的结果，这一次跑了五万多帧：




游戏效果展示在 record_dqn.gif，agent 的板子会上下移动，但还不太能准确击球，可能是训练轮数太少。

DDQN：累计奖励随样本训练量增加越来越趋向于最低值，不知道是算法存在 bug 还是轮数太少还未收敛，还是因为每次跑到几万帧之后由于我自己的设备内存不足就跑不下去了，所以暂时还无法验证。这里取了坚持运行轮数最多的一次的结果，这一次达到了近二十万帧：




游戏效果展示在 record_ddqn.gif，agent 的板子几乎一直停留在最上方，感觉不太对劲，不知道是训练轮数太少还是算法的问题。

# 5 小结

在这次实验中，我发现自己之前没太接触过深度学习、不熟悉 pytorch 还是给我阅读框架代码和补全算法的过程带来了很大困难，而且轻薄本跑深度学习还是勉为其难了一些，训练太慢了，而且由于内存不足每次训练到几万帧就会强行退出，所以也很难验证算法公式写没写对、超参数设置的合不合适，修改一下重新训练，就又要一个小时，，由于算法前期偏探索，本身就有一定随机性，设备又只能坚持到前几万帧，还不一定看得出效果，写到后面确实有点绝望了。