强化学习第三次实验

徐宽 - SA21229033

一、实现过程

1.1 环境配置

先按照助教给的教程来

- 新建一个 conda 环境，安装 numpy + gym
- `conda install -c conda-forge atari_py`
- 配置 tensorflow-gpu

> 然后我发现这一步是真的太复杂啦！
>
> 回想起之前用 conda 一行代码配置 pytorch_gpu 的经历，我找了找 tensorflow 有没有类似的，还真有！

前面的直接不要，推倒重来，因为下面一行代码，就直接新建了一个配置好 tensoeflow-gpu 的 conda 虚拟环境

- `conda create -n tf tensorflow-gpu=1.14.0`

非常的方便，不用考虑各种版本对应的事，感谢 conda 团队！

> 强烈建议助教之后可以教大家用这种方法，对小白来讲真的是太友好了！

- 之后就是简单的安装库：`pip install gym matplotlib pandas`
- 配置游戏环境 `conda install -c conda-forge atari_py`

1.2 安装 ffmpeg

在`win10`上使用 `Monitor` 监控器保存训练游戏的视频时出现了错误：

> gym.error.DependencyNotInstalled: Found neither the ffmpeg nor avconv executables. On OS X, you can install ffmpeg via 'brew install ffmpeg'. On most Ubuntu variants, 'sudo apt-get install ffmpeg' should do it. On Ubuntu 14.04, however, you'll need to install avconv with 'sudo apt-get install libav-tools'.

提示信息只给了 Mac OS X 和 Ubuntu 系统的解决方案，对于 Windows 用户貌似不太友好啊。不过既然缺少库，自然是先奉上 `pip` 大法：`pip install ffmpeg`，然而并没有用。

最后参考一篇博客 下载了 ffmpeg，配置好环境变量的路径之后终于搞定了！

1.3 显存不足?

跑了57个 Episode 后，报错

> OP_REQUIRES failed at save_restore_v2_ops.cc:137

查了一下貌似是内存不足导致模型无法保存。。。于是我把 pycharm 的最大内存设置为 2048M

结果甚至连报错信息都不给就直接 Process finished with exit code -1073741819 (0xC0000005)

我还尝试了重启大法等，最后弄了很久没解决，于是我决定不保存模型了，就让程序一直跑就完事了。

二、代码详解

- 状态空间预处理： `class StateProcessor()`

  该函数量化了游戏图像中可能的像素值，将 210x160x3 的 RGB 图像转化为 84x84 的灰度图像，大大减小了观测状态空间。

- 定义深度神经网络来估计 Q 值：`class Estimator()`

  其中网络结构是 3 个卷积层加一个全连接层，用 `predict()` 预测动作值函数，用 `update()` 更新网络，这个部分我没有做任何改动，就不多说了。

- 策略：`make_epsilon_greedy_policy()`

  基于给定的 Q 值估计器和 ϵ 创建贪心策略

- 主要函数：`deep_q_learning()`

  1. **Populate replay memory**

     ```
     replay_memory = populate_replay_buffer( sess, env, state_processor, replay_memory_init_size, VALID_ACTIONS, Transition, policy )
     ```

     初始填充经验回放缓冲区

  2. **Target network update**

     ```
     if total_t % update_target_estimator_every == 0:
        copy_model_parameters(sess, q_estimator, target_estimator)
        print("\nCopied model parameters to target network.")
     ```

     每隔 update_target_estimator_every=10000 个 step， 将 q_estimator（主DQN网络） 的参数复制到 target_estimator（目标网络）上。目标网络是主 DQN 网络的副本，这样做的目的是打破训练数据之间的时序关联性，从而提高训练的稳定性。

  3. **Take a step in the environment**

```
action_probs = policy(sess, state, epsilon)
action = np.random.choice(np.arange(len(action_probs)), p=action_probs)
next_state, reward, done, _ = env.step(VALID_ACTIONS[action])
next_state = state_processor.process(sess, next_state)
next_state = np.append(state[:, :, 1:], np.expand_dims(next_state, 2),
axis=2)
```

基于当前状态，根据 $\epsilon-$ 贪婪策略函数得到此时的动作，执行该动作，然后得到下一个状态。这样便完成了一个 step 的更新。

4. **Save transition to replay memory**

```
replay_memory.append(Transition(state, action, reward, next_state,
done))
```

保存经验回放缓存

5. **Sample a minibatch from the replay memory**

```
samples = random.sample(replay_memory, batch_size)
states_batch, action_batch, reward_batch, next_states_batch, done_batch
= map(np.array, zip(*samples))
```

随机从 replay_memory 中取出 batch_size 个状态

6. **use minibatch sample to calculate q values and targets**

```
q_values_next = target_estimator.predict(sess, next_states_batch)
targets_batch = reward_batch + np.invert(done_batch).astype(np.float32)
* discount_factor * np.amax(
   q_values_next, axis=1)
```

使用目标网络得到下一个 minibatch 的 Q 值，然后根据公式计算该状态的目标值

7. **Perform gradient descent update**

```
states_batch = np.array(states_batch)
loss = q_estimator.update(sess, states_batch, action_batch,
targets_batch)
```

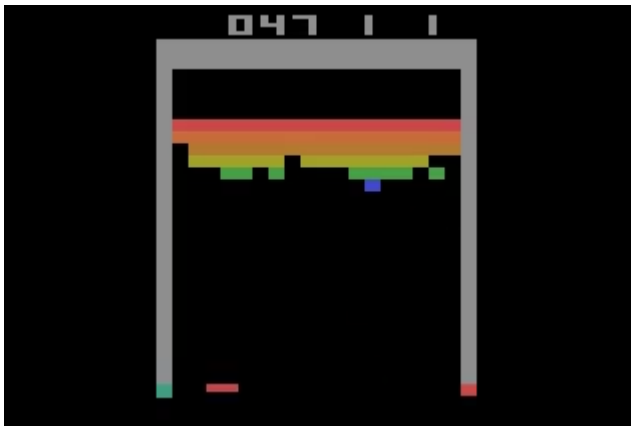
根据当前状态、动作、目标值来更新网络，返回 loss

不

三、结果展示

3.1 最高得分

4150epi 的结果，这不太智能的智能体玩到了 47 分，还不错，我快乐了



最后（跑了21h）

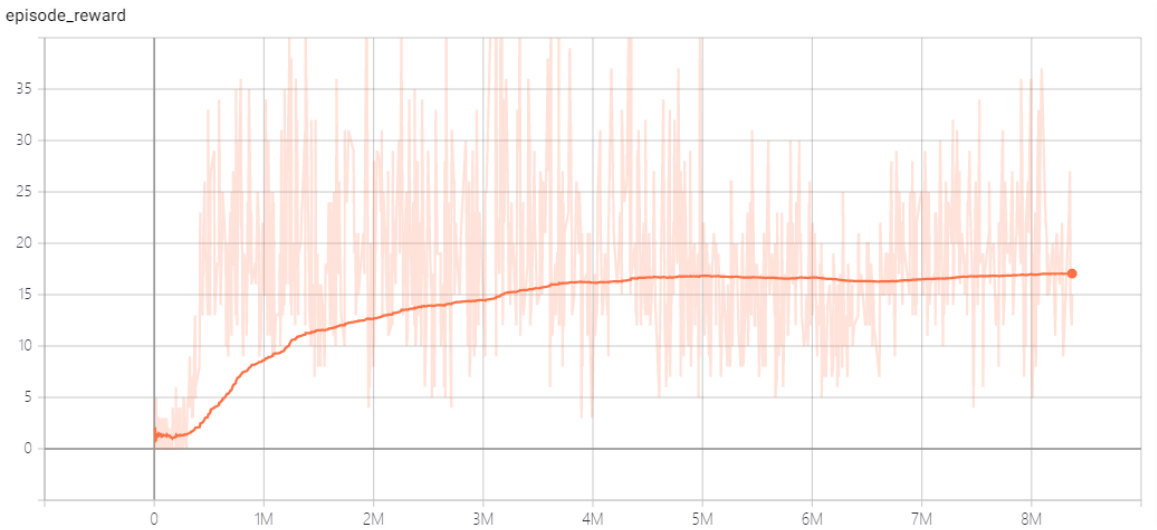
```
Step 968 (8370393) @ Episode 10000/10000
Episode Reward: 15.0

Process finished with exit code 0
```

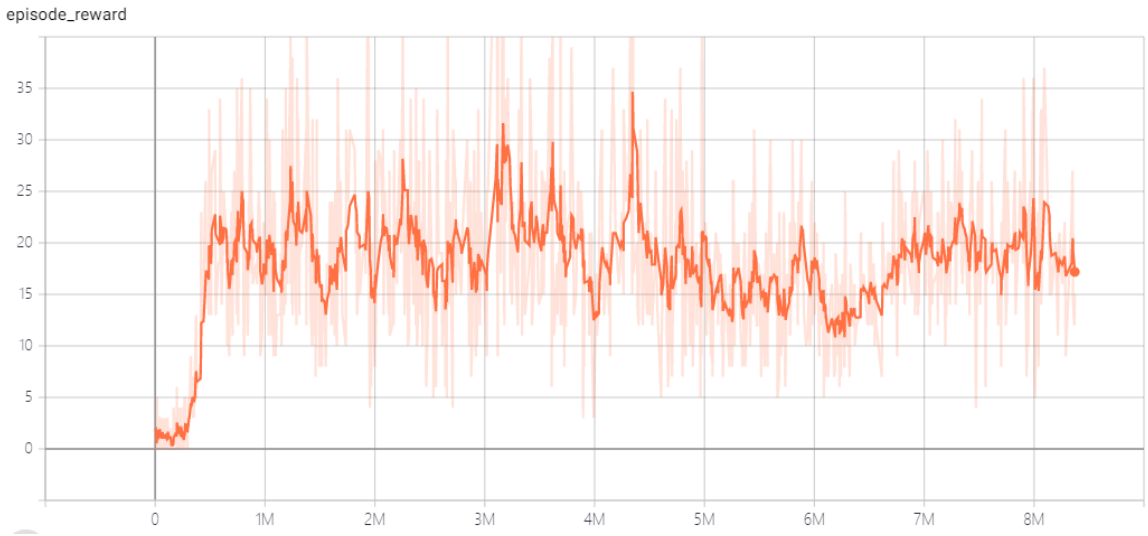
看来助教哥哥说跑到 15 就可以，是有依据的，这恒河里。

3.2 reward 曲线

- smoothing 拉满



- smoothing = 0.8



> episode_length 的曲线跟 reward 长得差不多，就不放了。

3.3 其它曲线

- epsilong

> 这个纵坐标怎么这么奇怪，是出 bug 了么

- q-value