强化学习：作业三

傅浩敏 MG20370012

2020年12月9日

1 作业内容

我们需要在gym Atari环境中实现DQN算法及其变体。本实验的实验环境是Atari Game Pong，Agent需要操控球拍与系统互相击球，未接到球则对方计一分，先取得21分者获胜。实验目标是训练DQN及其变体作为Agent获得游戏胜利，并使获胜时的分差尽可能大。在本次实验中，我分别实现和训练了DQN(1)、Double DQN(2)以及Dueling DQN(3)，并评估了它们在训练过程中的表现和它们在上述游戏中的性能。同时我也实现了Prioritized Replay Buffer(4)，但是完整的算法受限于我的硬件性能无法运行，因此我对论文中的算法进行了一定的简化，并且评估了简化后的算法对DQN训练过程的影响。

2 实现过程

2.1 算法描述

Q-learning(5) 在传统Q-learning算法中，我们使用一张 Q 表来记录环境状态 s 以及该状态对应各个动作 a 的长期回报值 $Q(s,a)$，并使用下式来更新 Q 表:

$$\begin{cases} a' = \arg\max_x Q(s',x) \\ Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \end{cases}$$

其中 s' 是在状态 s 下执行动作 a 后的新状态，α 和 γ 分别为学习率和折扣系数。

DQN 在DQN中，我们不使用表型数据结构记录 Q 值，而是用一个深度神经网络来计算不同的状态-动作对应的 Q 值。DQN相较于传统的Q-learning算法能更好地处理状态-动作空间较大的场景。在DQN中，我们需要最小化TD error，既使网络输出 $Q(s,a)$ 逼近于长期回报的估计值 $r +$

$\gamma Q(s', \arg\max_x Q(s', x))$，其中 γ 为折扣系数。我使用均方误差作为损失函数，因此神经网络优化器需要最小化下式：

$$\|r + \gamma Q(s', \arg\max_x Q(s', x)) - Q(s, a)\|_2^2$$

注意到，我们在改变网络参数的时候也会改变优化目标，因此我们需要复制一份原神经网络作为目标网络，从而使优化目标相对稳定。因此改写损失如下：

$$\|r + \gamma Q'(s', \arg\max_x Q'(s', x)) - Q(s, a)\|_2^2$$

其中 Q 为原网络输出，Q' 为目标网络输出。在经过一段时间的训练后，我们需要将原网络参数复制到目标网络上。

Double DQN 该变体是对DQN中训练目标的优化。在Q-learning算法中，采用 $\arg\max$ 选择动作会导致对 Q 值的估计有一定程度的偏高(2)。Double DQN通过解耦动作选择和 Q 值估计来减少偏差值。即在当前网络上对状态 s' 通过 $\arg\max$ 选择动作 a'，而在目标网络上计算 s', a' 对应的 Q 值。因此可将损失函数改写为：

$$\|r + \gamma Q'(s', \arg\max_x Q(s', x)) - Q(s, a)\|_2^2$$

通过这种方式，可以缓解估计值偏高的问题。

Dueling DQN 该变体是对DQN中网络结构的优化。在原始DQN中，神经网络模型直接输出了各个动作对应的 Q 值；而在Dueling DQN中，模型的输出会由Value Function和Advantage Function两个子网络的输出相加获得。其中Value Function仅输出对应状态的 Q 值，而Advantage Function则会输出对应状态各个动作对 Q 值的“增益”。通过这种方式可以使网络更加精确地估计 Q 值。

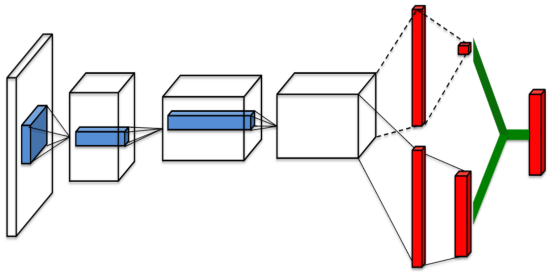

Figure 1: Dueling DQN 网络结构(3)

2.2 代码实现

DQN学习过程伪代码如下：

Algorithm 1 DQN learning function

Input: current step $step$,update interval $update$, minibatch k, discount rate γ, sample buffer $buffer$, model π, target model π'.
Sample state, next state, action, reward and done mask from buffer $s0, s1, a, r, done = buffer.sample(k)$
Calculate models' outputs $modelOutputs = \pi(s0)$, $targetModelOutputs = \pi'(s1)$
Find max value in actions $targetMax$=max($targetModelOutputs$)
Calculate target $target$=$r + \gamma * (1 - done) * targetMax$
Find value of the actions $presentValue = modelOutputs[a]$
Calculate MSE loss $loss = \|presentValue - target\|_2^2$
Update parameters $\pi.update()$
if fr mod $update$ == 0 **then**
    Update target model $\pi' = \pi$
end if

Double DQN只在Algorithm 1的基础上优化了动作选择的过程：

Algorithm 2 Double DQN learning function

Input: current step $step$,update interval $update$, minibatch k, discount rate γ, sample buffer $buffer$, model π, target model π', loss function $mseLoss$.
Sample state, next state, action, reward and done mask from buffer $s0, s1, a, r, done = buffer.sample(k)$
Calculate models' outputs $modelOutputsS0 = \pi(s0)$, $modelOutputsS1 = \pi(s1)$, $targetModelOutputs = \pi'(s1)$
Find actions of max value $maxActions$=arg $\max_a(targetModelOutputsS1)$
Find target values $targetMax = targetModelOutputs[maxActions]$
Calculate target $target$=$r + \gamma * (1 - done) * targetMax$
Find value of the actions $presentValue = modelOutputsS0[a]$
Calculate MSE loss $loss = \|presentValue - target\|_2^2$
Update parameters $\pi.update()$
if fr mod $update$ == 0 **then**
    Update target model $\pi' = \pi$
end if

Dueling DQN则是修改了网络结构:

Algorithm 3 Dueling DQN network

1: **Input:** observation $image$, CNN sub-network cnn, full connection layer fc, value network $valueNet$, advantage network $advNet$
2: **Output:** Q value vector for each actions $value$
3: Calculate image features $features = cnn(image)$
4: Calculate hidden layer $hiddenLayer = fc(features)$
5: Calculate value $value = valueNet(hiddenLayer) + advNet(hiddenLayer)$
6: **return** $value$

3 复现方式

3.1 训练复现

首先在主文件夹下运行 `pip install -r requirements.txt` 安装依赖，如果已安装依赖也可跳过此步骤。`code` 文件夹中包含的三个python文件 `atari_dqn.py`，`atari_ddqn.py`，`atari_dueldqn.py` 分别对应了DQN、Double DQN和Dueling DQN，在 `code` 文件夹下运行 `python xxx.py --train` 可复现对应的训练过程。比如想复现DQN的训练过程则可在 `code` 文件夹下运行 `python atari_dqn.py --train`。

3.2 测试复现

依然可以在主文件夹下运行 `pip install -r requirements.txt` 安装依赖，如果已安装依赖也可跳过此步骤。`code/model` 文件夹下存放了训练好的模型参数，其中Dueling DQN成功达到了预定的训练目标。在 `code` 文件夹下运行 `python atari_dqn.py --test --model_path=model/dqn/model_last.pkl` 可以复现DQN的测试结果；运行 `python atari_ddqn.py --test --model_path=model/ddqn/model_last.pkl` 可以复现Double DQN的测试结果；运行 `python atari_dueldqn.py --test --model_path=model/dueldqn/model_best.pkl` 可以复现Dueling DQN的测试结果。

3.3 参数介绍

为了更好地对比不同算法的实验效果，对于DQN、Double DQN和Dueling DQN我使用了完全相同的超参数设置。同时，DQN和Double DQN使用了相同的网络结构，而Dueling DQN仅在模型最后的全连接层有所改动。

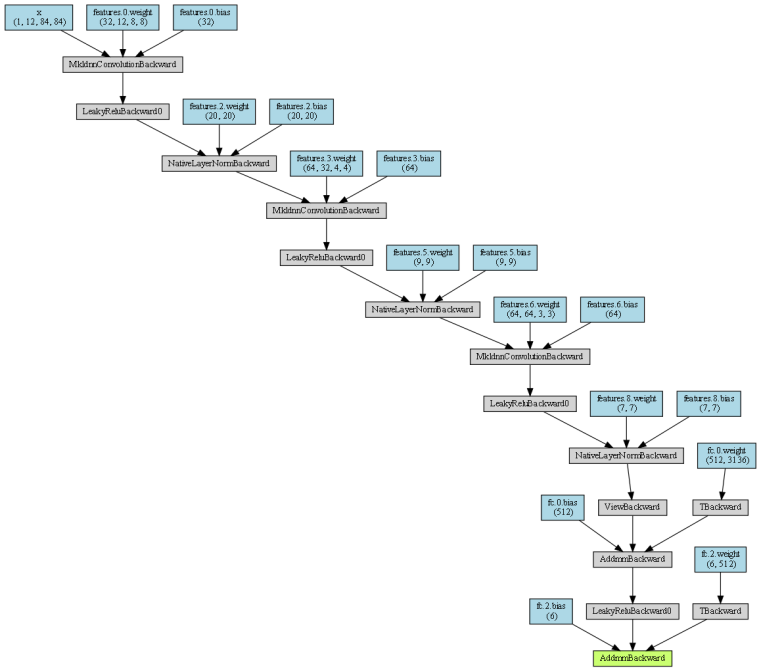

Figure 2: DQN和Double DQN网络参数

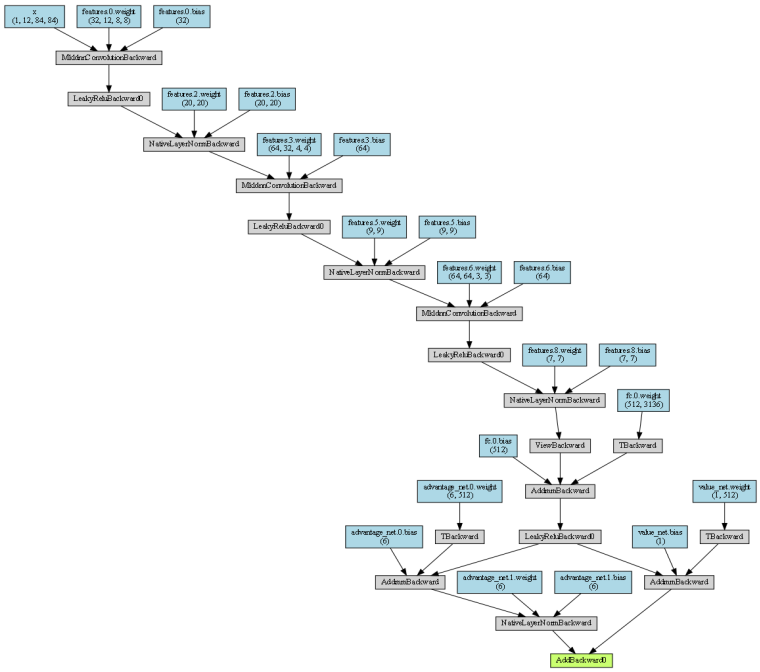

Figure 3: Dueling DQN网络参数

Figure 2和Figure 3分别展示了DQN和Dueling DQN的两种网络参数。在原有卷积神经网络的基础上，我穿插了一些归一化层以提升网络收敛速度，我使用了均方差损失并且改用Adam优化器对网络参数进行优化。

Table 1: 关键超参数

名称	默认值	说明
learning_rate	1e-5	学习率 α
gamma	0.99	折扣系数 γ
frames	2000000	训练总帧数
update_tar_interval	1000	目标网络参数更新周期
batch_size	32	minibatch帧大小
max_buff	100000	最大帧缓存容量
win_reward	18	目标平均奖励

4 实验效果

4.1 实验图表展示与分析

DQN、Double DQN和Dueling DQN的累计奖励和样本训练量之间关系如下。也可以在 `code` 文件夹下运行 `tensorboard --logdir ./model` 并在浏览器中打开链接查看详细的训练过程。

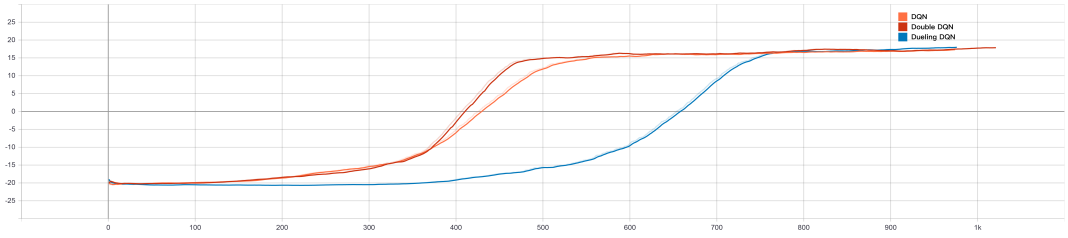

Figure 4: 最优平均奖励

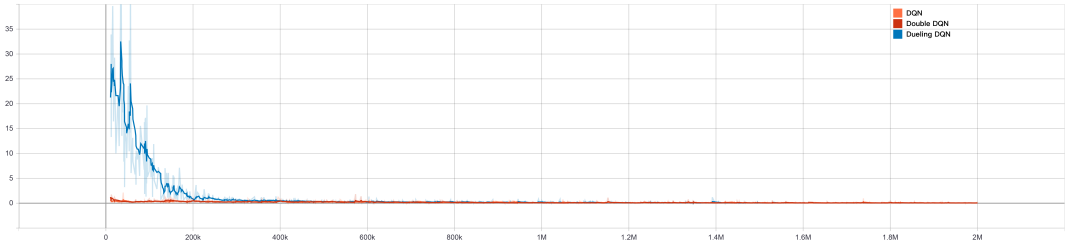

Figure 5: 每帧损失

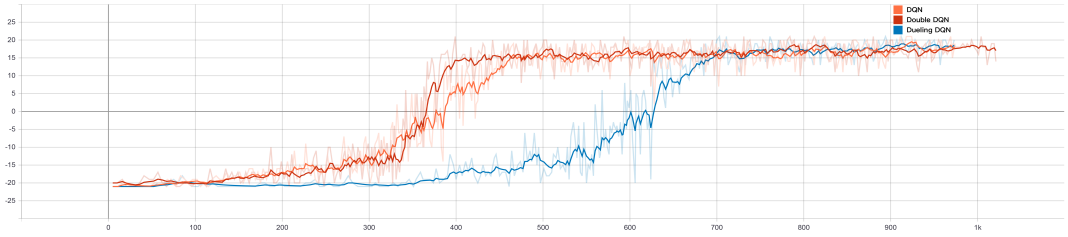

Figure 6: 平均奖励

5 小结

5.1 关于算法本身

5.2 关于实验过程

参考文献

[1] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[2] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[J]. arXiv preprint arXiv:1509.06461, 2015.

[3] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning[C]//International conference on machine learning. PMLR, 2016: 1995-2003.

[4] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.

[5] Watkins C J C H, Dayan P. Q-learning[J]. Machine learning, 1992, 8(3-4): 279-292.