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)(2)、Double DQN(3)以及Dueling DQN(4)，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。

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，本文之后所提及的DQN都是此版本的DQN(2)。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(3)。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”。这种方式可以使 Q 值估计更精确。

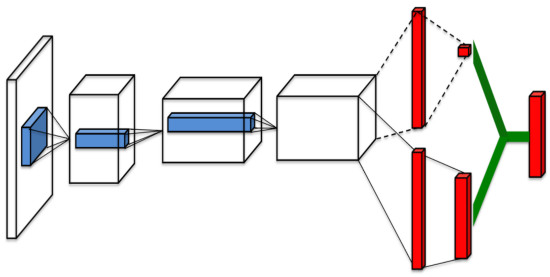

Figure 1: Dueling DQN 网络结构(4)

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 vector for each action $value = valueNet(hiddenLayer)$+normalize($advNet(hiddenLayer)$)
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。

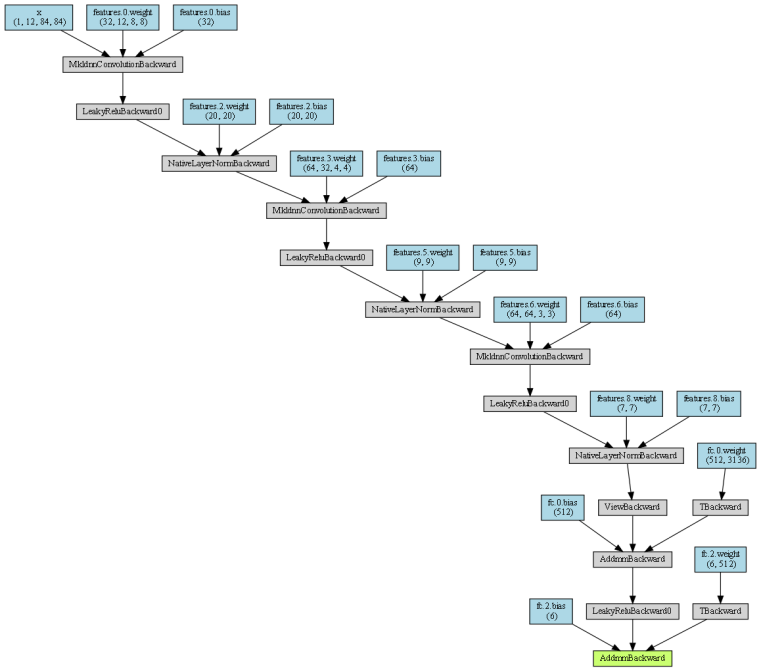

Figure 2: DQN和Double DQN网络参数

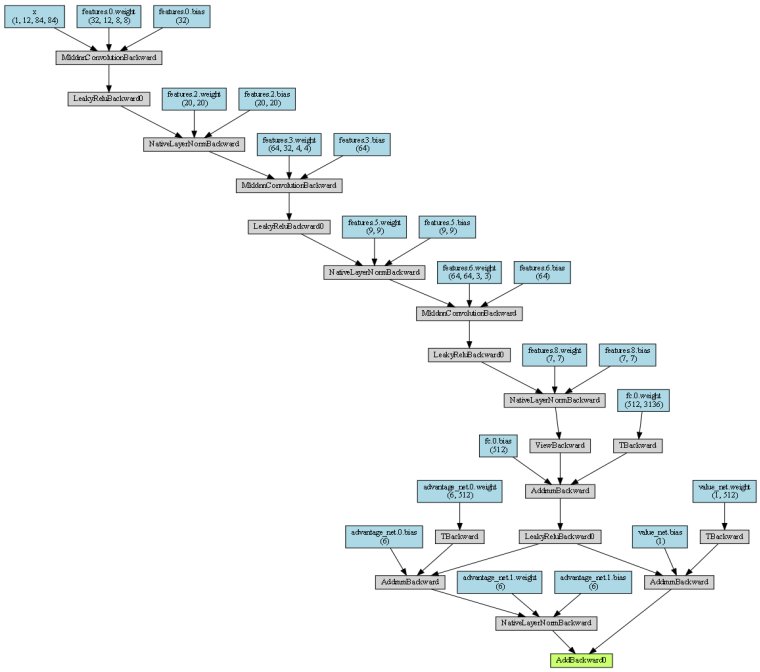

Figure 3: Dueling DQN网络参数

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。

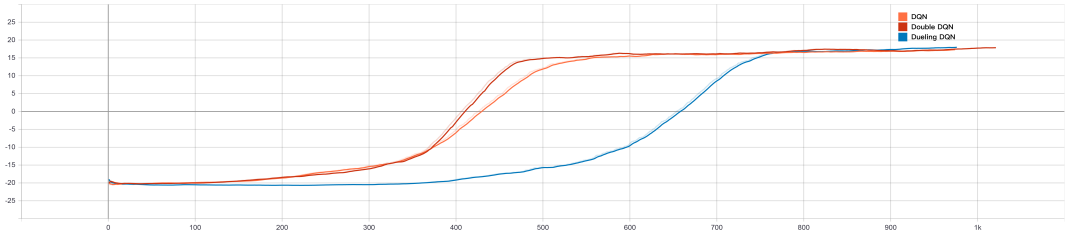

Figure 4: 最优平均奖励变化曲线

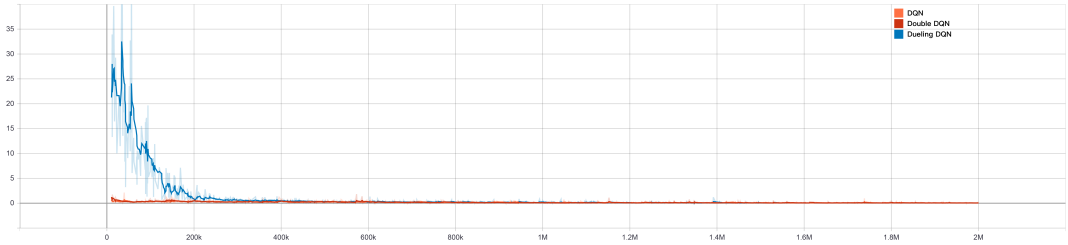

Figure 5: 每帧损失变化曲线

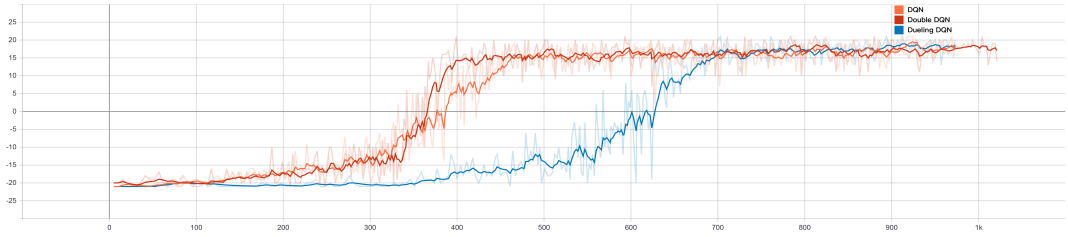

Figure 6: 平均奖励变化曲线

观察Figure 4和Figure 6可以发现，在DQN的训练过程中，刚开始时模型的平均奖励提升较平缓，在训练中段模型性能会快速提升，最后逐渐收敛。并且在当前参数设置下，Double DQN收敛速度最快，DQN略逊之，而Dueling DQN的收敛速度相较于前两者明显缓慢。在模型收敛后的性能表现上，Dueling DQN最好，Double DQN次之，原始DQN最差，但是它们之间性能差距不大，基本可以认为在误差范围之内。观察Figure 5可以发现，DQN和Double DQN的loss变化曲线类似，并且loss的波动在整个训练过程中都比较小，但是Dueling DQN在训练开始时的loss会大很多，随着训练轮次增加loss才会恢复到较低水平。同时从Figure 6中可以看出Double DQN训练过程中的平均奖励波动最小，而Dueling DQN有较大的波动，可见Double DQN确实能有效平稳训练过程。

5 小结

5.1 关于算法本身

DQN通过神经网络计算的方式计算 Q 值，相较于传统Q-learning中采用的表格记录的方式能更好地处理复杂环境，因此现实中的应用场景也更加广

泛。此外DQN也和Q-learning会产生过度估计问题，在Double DQN中通过解耦动作选择网络和 Q 值计算网络可以一定程度上减小的误差，从实际上的实验结果来看，Double DQN的训练过程也确实比DQN更快更稳定，同时对模型收敛后的性能也几乎没有影响。Dueling DQN通过将网络输分为价值函数和优势函数一定程度上提升了模型的性能，但在本问题上提升的效果不是很明显，并且对模型的收敛速度有较大的影响，也许Dueling DQN在更加复杂的问题上更能体现出效果。此外无论是DQN还是其变体，训练模型的时间成本都非常高。这主要有三方面的原因：一是模型本身需要较多轮次的参数更新才能收敛，并且在此过程中需要从环境中大量采样；二是由于训练初始阶段的奖励和损失变化幅度不大，我们很难在训练初期判断网络是否收敛，只有在实验接近完成时才能知道结果；三是DQN的超参数和网络结构过多，而神经网络的性能较大地依赖于这些参数的设置，我们需要尝试多种组合才能找到较优的超参数和网络结构。

5.2 关于实验过程

最开始我直接使用了原版DQN(1)进行训练，但是模型在跑完两百万帧后没有收敛的迹象，在尝试增加层归一化并修改优化器后依旧无法收敛，故选择Nature DQN(2)作为baseline进行对比实验。此外，我还尝试了带优先级的采样方式，由于原版的Prioritized Replay Buffer算法(6)比较复杂，我在实验过程中尝试编写了简化版本：每次采样若干倍于minibatch的数据，然后从中选取TD Error最大若干个进行参数更新。但是这样采样会导致训练过程波动较大，最后模型没能很好地收敛，也许原版算法能缓解这类问题。此外，神经网络模型的性能存在一定的随机性，不是每次训练都能达到最佳状态，因此本报告中的实验数据来自于几次运行产生的最优模型。由于时间限制，我只是凭直觉尝试了一组超参数设置和网络结构，我相信对网络进行一定优化能进一步提升模型的效果，尤其在Dueling DQN中值函数和优势函数的设计还有很大的改善空间。最后，强化学习实验的试错成本太高，即使针对如此简单的实验环境，我也花费了大量的时间训练模型才勉强达到预定的目标平均奖励。

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] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

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

[4] 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.

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

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