实验 3. 强化学习实践

MG1733098，周华平，zhp@smail.nju.edu.cn

2017 年 12 月 31 日

综述

与传统的监督、无监督学习不同，强化学习的过程需要与学习环境进行交互，利用环境反馈信息进行学习。在本次实验中，我们从学习环境的安装、常用强化学习算法的实现和强化学习算法改进这些方面完整的体会了一次强化学习研究的过程。具体说来，本次实验我使用 Python 语言分别实现了 Q-learning、DQN 以及 Improved DQN 这三个强化学习算法，并在 Gym 实验环境下对 CartPole、MountainCar 以及 Acrobot 进行了训练。其中 Q-learning 利用 numpy 实现；而 DQN 和 Improved DQN 则通过 PyTorch 搭建神经网络来实现。

实验二.

Q-learning 实现

Q-learning 算法中使用了 ϵ-贪心法，基于一个概率来对探索和利用进行折中。若尝试次数非常大，那么在一段时间后，Action 的 Reward 都能很好地近似出来，不需要再探索。因此，对于 Q-learning 以及之后的 DQN，假设当前尝试次数为 t，我们使用公式 (1) 来对 ϵ 进行更新。

$$\epsilon = eps\_end + (eps\_start - eps\_end) * \exp(-t/eps\_decay) \tag{1}$$

基于类似的理由，对于 learning rate α，我们可以采用公式 (2) 进行更新。

$$\alpha = lr\_end + (lr\_start - lr\_end) * \exp(-t/lr\_decay) \tag{2}$$

在 Python 中实现 Q-learning 算法主要利用了 numpy 提供的一些函数。我们将 Q 值函数用 Python 的 dict 来表示，用 defaultdict (**lambda**: np.zeros(action_space.n)) 来对 Q table 进行初始化。Q table 的 key 是经过离散化后的状态，value 是所有 Action 对应的 Q 值构成的 ndarray。这里的 defaultdict 是 Python 提供的 collections 容器，作用是在调用 get 时对 value 进行初始化操作。我们通过一个 lambda 表达式将 qtable 中的所有 Q 值都初始化为 0。算法其他部分的实现比较直观，基本和伪代码中的结构一一对应，因此这里不过多赘述，具体实现详见 myQLearning.py。

需要特别指出的是，为了加快训练算法的收敛速度，我对每个任务的 Reward 进行了简单的重新定义。具体来说，对于 CartPole，当轨迹在某一步提前结束时，该步的 Reward 为 -100；而对于 MountainCar 和 Acrobot，当轨迹在某一步提前结束时，该步的 Reward 为 100。这里的定义利用了一个非常直接的观察：CartPole 需要坚持的越久越好，而 MountainCar 和 Acrobot 则越快完成越好。除此之外，我并没有对状态参数进行额外的分析，从而“凑出”一个更好的 Reward，即使这样得到的训练效果可能会更好。我认为对于状态参数的分析并不应该由人来完成，如果 Reward 被人工定义为关于状态参数的一个复杂函数，则这个函数包含了我们对于这个任务的一些先验知识，而这些本应该由算法在训练过程中慢慢学习。如果利用了这些复杂的先验知识的话，我感觉和在游戏中开外挂没什么区别。

Q-learning 训练

对于 Q-learning 中的几个任务，我们统一采用np.linspace()来对状态空间进行离散化，具体参数如表 1所示。

表 1: Q-learning 离散化参数

	CartPole	MountainCar	Acrobot
start	$(-2.4, -1.1, -12°, -1.5)$	$(-1.2, -0.07)$	$(-1, -1, -1, -1, -4\pi, -9\pi)$
stop	$(2.4, 1.1, 12°, -1.5)$	$(0.6, 0.07)$	$(1, 1, 1, 1, 4\pi, 9\pi)$
num	$(5, 5, 9, 5)$	$(20, 20)$	$(10, 10, 10, 10, 10, 10)$

Q-learning 的超参数设置如表 2所示。

表 2: Q-learning 超参数设置

超参数	参数意义	CartPole	MountainCar	Acrobot
discount	Q-learning 算法中的 γ	0.99	0.99	0.9
lr_start	α 的初始值	0.9	0.9	0.9
lr_end	α 的结束值	0.001	0.0015	0.0015
lr_decay	α 的衰减权重	1000	200	200
eps_start	ϵ 的初始值	0.9	0.9	0.9
eps_end	ϵ 的结束值	0.05	0.05	0.05
eps_decay	ϵ 的衰减权重	1000	200	200

对于每个任务，我将多次训练中得到的最好的策略进行测试。对于 CartPole、MountainCar 以及 Acrobot，我分别测试了 100 条轨迹，最终得到 reward 的均值和标准差如表 3所示。

表 3: Q-learning Average Reward

	CartPole	MountainCar	Acrobot
$mean \pm std$	385.29 ± 69.15	-141.68 ± 22.79	-170.70 ± 37.79

实验三.

DQN 实现

DQN 在基本的 Deep Q-Learning 算法的基础上使用了 Experience Replay 经验池，通过将训练得到的数据储存起来然后随机采样的方法来降低数据样本的相关性，进而提升了性能。

在本实验中，我选择使用 PyTorch 来实现 DQN。在定义 Q 值网络时我使用了 MLP，其中网络结构由 3 层 Linear 构成；激活函数使用 PReLU，同时在每个隐层中增加 Batch Norm 来对相应的 activation 做规范化操作。DQN 中的神经网络和梯度计算的实现主要利用了 PyTorch 提供的 Optimizer 以及 loss 函数。具体说来，在 DQN 中我采用optim.Adam作为优化函数，用nn.MSELoss()来计算均方误差。ϵ 的更新同公式 (1)。其余的算法实现细节详见 myDQN.py。

Q 值网络的定义如下所示：

```
class DQN(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dim):
        super(DQN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.PReLU(),
        )
        self.layer2 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.PReLU(),
        )
        self.out = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        return self.out(x)
```

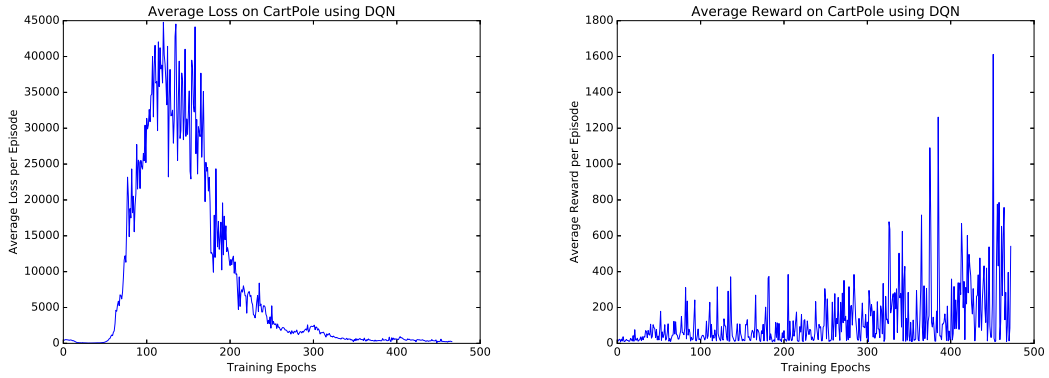
DQN 训练

DQN 的超参数设置如表 4所示。

表 4: DQN 超参数设置

超参数	参数意义	CartPole	MountainCar	Acrobot
memory_size	Replay Memory 的大小	10000	10000	5000
batch_size	mini-batch 的大小	128	128	128
hidden_dim	DQN 的隐层维度	50	50	50
discount	DQN 算法中的 γ	0.99	0.99	0.99
learning_rate	DQN 算法中的 α	0.001	0.001	0.001
eps_start	ϵ 的初始值	0.9	0.9	0.9
eps_end	ϵ 的结束值	0.05	0.05	0.05
eps_decay	ϵ 的衰减权重	200	50	200

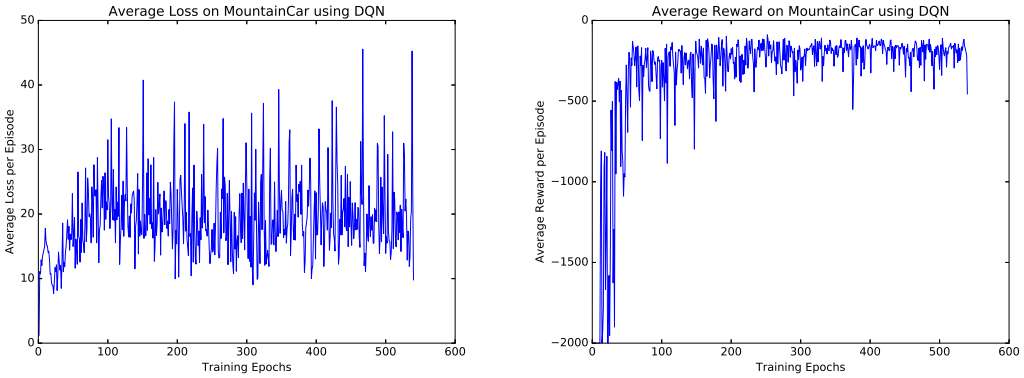
DQN 在 CartPole 上的实验结果如图 1所示。可以观察到 Loss 在超过 450 轮后达到收敛的状态。由于 ϵ 的最小值被设置为 0.05，因此即使 Training 了较多轮数，DQN 依旧会以 5% 的概率随机选择 Action。而 CartPole 似乎对于错误的 Action 比较敏感，当随机到错误的 Action 时，可能会导致该轨迹提前结束。因此在 Training 阶段 Reward 似乎并没有收敛到一个固定值，然而我们可以观察到随着 Training 轮数的增加，Reward 的上限也在不断提高，这也从侧面体现出了训练是有效果的。



(a) Average Loss　　(b) Average Reward

图 1: Training Result of CartPole using DQN

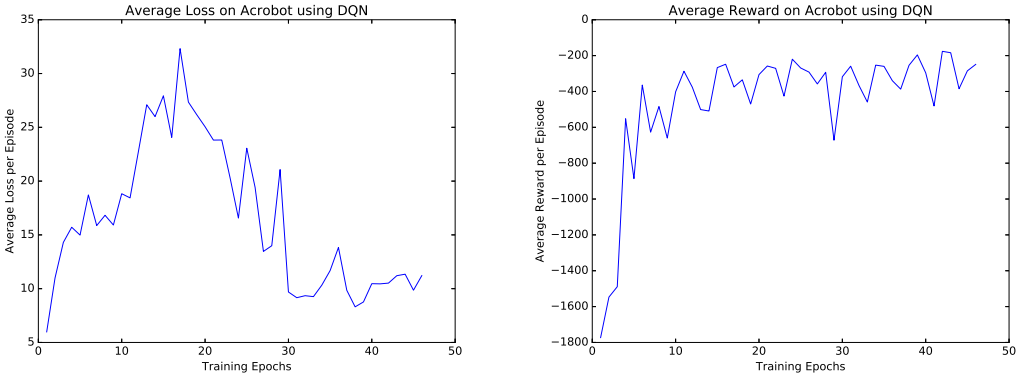
DQN 在 MountainCar 上的实验结果如图 2所示。其中 Reward 在超过 200 轮之后达到收敛的状态，而 Loss 也在超过 200 轮之后达到了基本稳定的状态。



(a) Average Loss (b) Average Reward

图 2: Training Result of MountainCar using DQN

DQN 在 Acrobot 上的实验结果如图 3所示。在超过 40 轮之后，Loss 达到了较低的水平，并且 Reward 也趋近于收敛。



(a) Average Loss (b) Average Reward

图 3: Training Result of Acrobot using DQN

对于每个任务，我将多次训练中得到的最好的策略进行测试。对于 CartPole 我测试了 50 条轨迹，对于 MountainCar 和 Acrobot 我分别测试了 100 条轨迹。最终得到 reward 的均值和标准差如表 5所示。

表 5: DQN Average Reward

	CartPole	MountainCar	Acrobot
$mean \pm std$	20000 ± 0	$-183.44 \pm 26.6669533318$	-86.84 ± 18.1117199625

实验四.

Improved DQN 实现

相比于 DQN，Improved DQN 进行了一个微小的改动：增加了目标 Q 网络 $\hat{Q}$。在计算目标 Q 值时，我们使用专门的目标 Q 网络 $\hat{Q}$ 来计算，而不是直接使用预更新的 Q 网络 Q。这样做的目的是为了减少目标计算与当前值的相关性。

在 DQN 中，目标 Q 网络会随着 Q 的更新而动态变化，这样不利于计算目标 Q 值，导致目标 Q 值和当前的 Q 值相关性较大，因此 Improved DQN 提出单独使用一个目标 Q 网络。而 $\hat{Q}$ 中的参数是通过延迟更新的方式从 Q 中获得：在每训练了 C 步之后，Improved DQN 将当前 Q 的参数值复制给 $\hat{Q}$。

Improved DQN 的实现基于 DQN，因此代码部分的改动比较少。另外在 Improved DQN 中需要新增一个超参数 target_c，用以表示 $\hat{Q}$ 的更新频率。

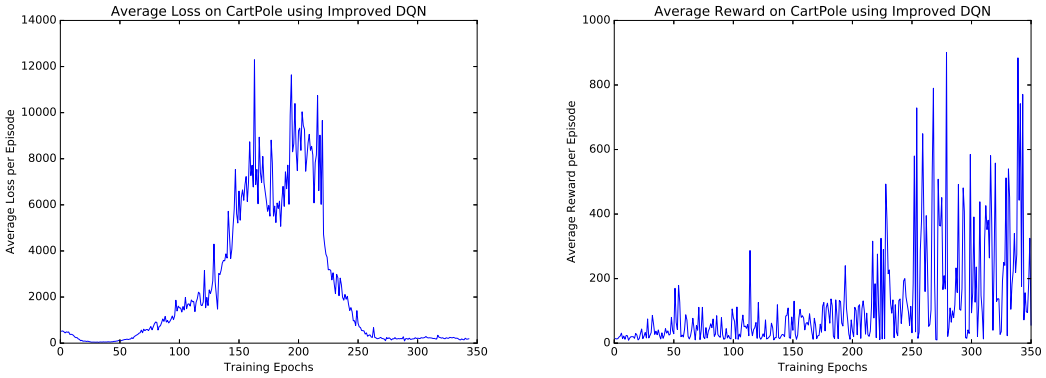
Improved DQN 训练

Improved DQN 的超参数设置如表 6所示。

表 6: Improved DQN 超参数设置

超参数	参数意义	CartPole	MountainCar	Acrobot
memory_size	Replay Memory 的大小	10000	10000	10000
batch_size	mini-batch 的大小	128	128	128
hidden_dim	DQN 的隐层维度	50	50	50
target_c	$\hat{Q}$ 的更新频率	10	10	5
discount	DQN 算法中的 γ	0.99	0.99	0.99
learning_rate	DQN 算法中的 α	0.001	0.001	0.001
eps_start	ϵ 的初始值	0.9	0.9	0.9
eps_end	ϵ 的结束值	0.05	0.05	0.05
eps_decay	ϵ 的衰减权重	200	50	200

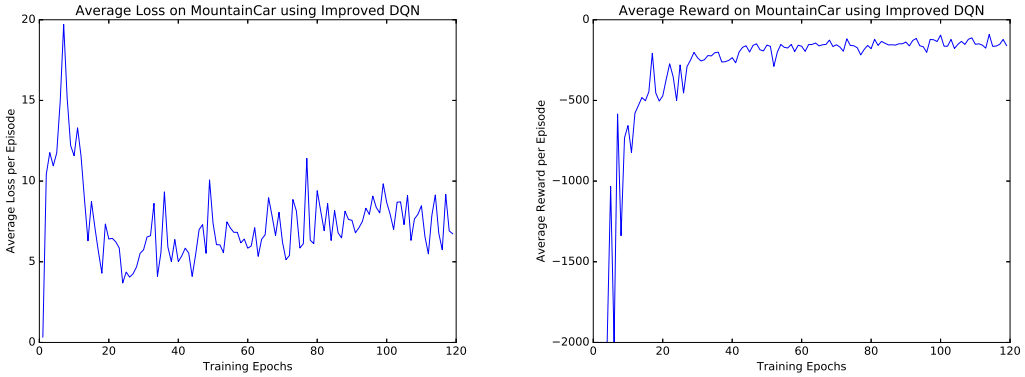
Improved DQN 在 CartPole 上的实验结果如图 4所示。可以观察到 Loss 在超过 300 轮后达到收敛的状态。



(a) Average Loss (b) Average Reward

图 4: Training Result of CartPole using Improved DQN

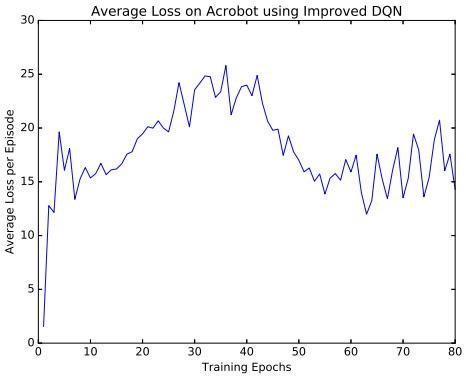
Improved DQN 在 MountainCar 上的实验结果如图 5所示。其中 Reward 在超过 60 轮之后达到收敛的状态，而 Loss 也在超过 60 轮之后达到了基本稳定的状态。



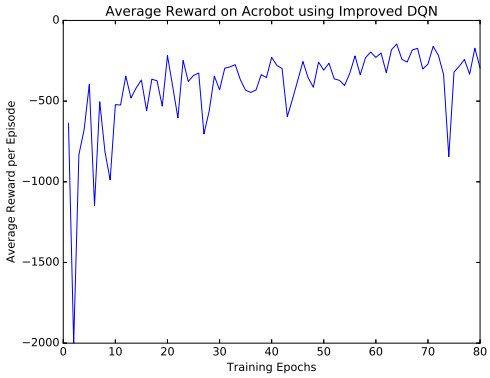
(a) Average Loss (b) Average Reward

图 5: Training Result of MountainCar using Improved DQN

Improved DQN 在 Acrobot 上的实验结果如图 6所示。在超过 50 轮之后，Loss 达到了较低的水平，并且 Reward 也趋近于收敛。



(a) Average Loss



(b) Average Reward

图 6: Training Result of Acrobot using Improved DQN

对于每个任务，我将多次训练中得到的最好的策略进行测试。对于 CartPole 我测试了 50 条轨迹，对于 MountainCar 和 Acrobot 我分别测试了 100 条轨迹。最终得到 reward 的均值和标准差如表 7所示。

表 7: Improved DQN Average Reward

	CartPole	MountainCar	Acrobot
$mean \pm std$	20000 ± 0	$-107.54 \pm 13.5273205033$	-84.31 ± 18.8024971746

Improved DQN 与 DQN 的异同

在训练过程中可以明显观察到 Improved DQN 相比于 DQN 表现得更加稳定，收敛速度也更快。以 MountainCar 为例，在使用 DQN 时，很多时候并没有办法训练出一个可行的模型：经过几十轮的训练，每一轮的 Reward 依旧为-2000。而在前一节中得到的 DQN 在 MountainCar 上的比较好的训练结果似乎带有一定的偶然成分；相比之下，Improved DQN 在 MountainCar 上几乎每次都能训练出一个可行的模型：在前十轮的训练中，Reward 就能突破-2000，并且一旦突破了-2000 以后，Reward 能够在之后的每一轮训练中快速增加，直至收敛。

在训练效果上，除了 MountainCar 以外，DQN 和 Improved DQN 在 CartPole 和 Acrobot 上的训练效果比较接近。而 DQN 之所以在 MountainCar 上的训练结果与 Improved DQN 差别较大，我认为主要是由于 DQN 训练出稳定的 MountainCar 模型比较困难：如果能够用 DQN 训练足够多的模型，其中最好的那个模型的效果应该和 Improved DQN 上得到的结果相近。

所以我认为 Improved DQN 在训练阶段比 DQN 表现得更加稳定，收敛速度也更快；而在训练模型收敛后，我猜测 DQN 与 Improved DQN 的训练效果应该会比较接近。

参考文献

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] 周志华. 机器学习. 清华大学出版社, 2016.