浙江大学

本 科 实 验 报 告

姓名： 严轶凡

学院： 控制科学与工程学院

系： 自动化

专业： 自动化（控制）

学号： 3200100917

指导教师： 周建光

年 月 日

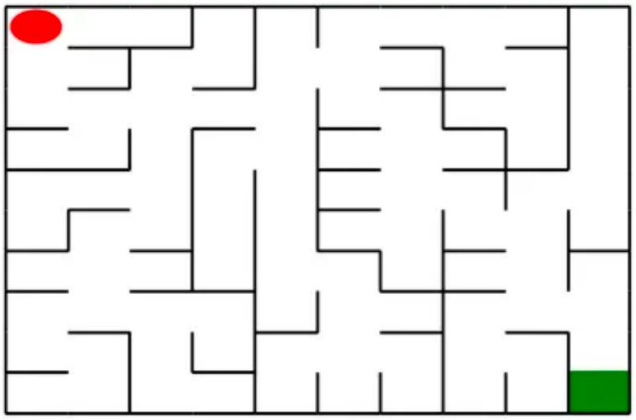
浙江大学 实验报告

一、实验目的和要求（必填）

在本实验中，要求分别使用基础搜索算法和 Deep QLearning 算法，完成机器人自动走迷宫。

二、实验内容和原理（必填）

实验内容：



如上图所示，左上角的红色椭圆既是起点也是机器人的初始位置，右下角的绿色方块是出口。

游戏规则为：从起点开始，通过错综复杂的迷宫，到达目标点(出口)。

在任一位置可执行动作包括：向上走 'u'、向右走 'r'、向下走 'd'、向左走 'l'。

执行不同的动作后，根据不同的情况会获得不同的奖励，具体而言，有以下几种情况：

（1）撞墙

（2）走到出口

（3）其余情况

实验原理：

（一）强化学习算法介绍

强化学习作为机器学习算法的一种，其模式也是让智能体在“训练”中学到“经验”，以实现给定的任务。

但不同于监督学习与非监督学习，在强化学习的框架中，我们更侧重通过智能体与环境的交互来学习。

通常在监督学习和非监督学习任务中，智能体往往需要通过给定的训练集，辅之以既定的训练目标（如最小化损失函数），通过给定的学习算法来实现这一目标。

然而在强化学习中，智能体则是通过其与环境交互得到的奖励进行学习。

这个环境可以是虚拟的（如虚拟的迷宫），也可以是真实的（自动驾驶汽车在真实道路上收集数据）。

在强化学习中有五个核心组成部分，它们分别是：环境（Environment）、智能体（Agent）、状态（State）、动作（Action）和奖励（Reward）。

在某一时间节点 t：

- 智能体在从环境中感知其所处的状态 s_t
- 智能体根据某些准则选择动作 a_t
- 环境根据智能体选择的动作，向智能体反馈奖励 r_{t+1}

（二）QLearning 算法

Q-Learning 是一个值迭代（Value Iteration）算法。

与策略迭代（Policy Iteration）算法不同，值迭代算法会计算每个”状态“或是”状态-动作“的值（Value）或是效用（Utility），然后在执行动作的时候，会设法最大化这个值。

因此，对每个状态值的准确估计，是值迭代算法的核心。

通常会考虑最大化动作的长期奖励，即不仅考虑当前动作带来的奖励，还会考虑动作长远的奖励。

Q-learning 算法将状态（state）和动作（action）构建成一张 Q_table 表来存储 Q 值，Q 表的行代表状态（state），列代表动作（action）：

Q-Table	a_1	a_2
s_1	$Q(s_1,a_1)$	$Q(s_1,a_2)$
s_2	$Q(s_2,a_1)$	$Q(s_2,a_2)$
s_3	$Q(s_3,a_1)$	$Q(s_3,a_2)$

在 Q-Learning 算法中，将这个长期奖励记为 Q 值，其中会考虑每个 ”状态-动作“ 的 Q 值，具体而言，它的计算公式为：

$$Q(s_t,a)=R_{t+1}+\gamma\times\max_a Q(a,s_{t+1})$$

计算得到新的 Q 值之后，一般会使用更为保守地更新 Q 表的方法，即引入松弛变量 alpha ，按如下的公式进行更新，使得 Q 表的迭代变化更为平缓。

$$Q(s_t,a)=(1-\alpha)\times Q(s_t,a)+\alpha\times\left(R_{t+1}+\gamma\times\max_a Q(a,s_{t+1})\right)$$

在强化学习中，探索-利用 问题是非常重要的问题。

具体来说，根据上面的定义，会尽可能地让机器人在每次选择最优的决策，来最大化长期奖励。

但是这样做有如下的弊端：

1. 在初步的学习中，Q 值是不准确的，如果在这个时候都按照 Q 值来选择，那么会造成错误。
2. 学习一段时间后，机器人的路线会相对固定，则机器人无法对环境进行有效的探索。

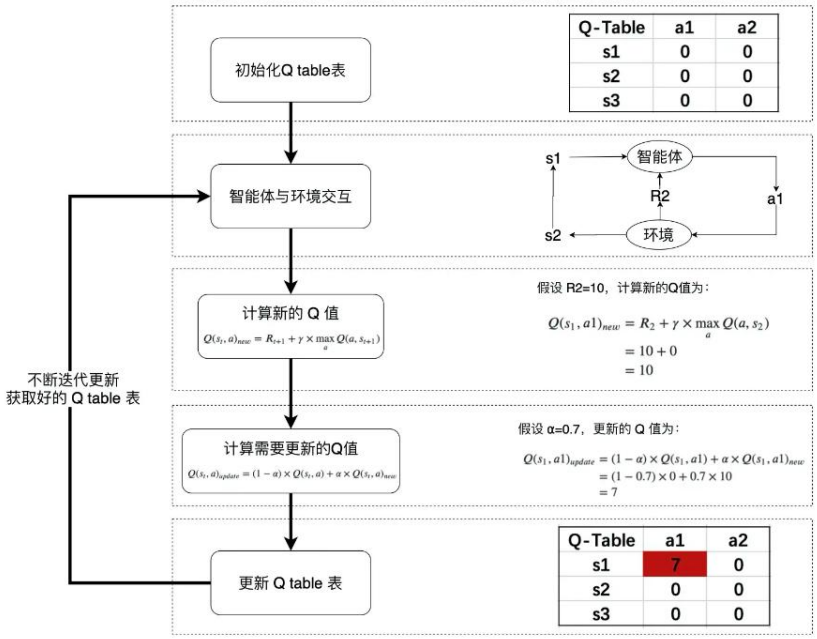
因此需要一种办法，来解决如上的问题，增加机器人的探索。

通常会使用 epsilon-greedy 算法：

1. 在机器人选择动作的时候，以一部分的概率随机选择动作，以一部分的概率按照最优的 Q 值选择动作。

2. 同时，这个选择随机动作的概率应当随着训练的过程逐步减小。

Q-learning 算法的学习过程如下：



（三）DQN（Deep Q-Learning）算法

强化学习是一个反复迭代的过程，每一次迭代要解决两个问题：给定一个策略求值函数，和根据值函数来更新策略。而 DQN 算法使用神经网络来近似值函数。

DQN 算法流程：

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-}=\theta$

For episode = 1, M **do**

  Initialize sequence $s_1=\{x_1\}$ and preprocessed sequence $\phi_1=\phi(s_1)$

  For $t=1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t=\text{argmax}_a Q(\phi(s_t),a;\theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1}=s_t,a_t,x_{t+1}$ and preprocess $\phi_{t+1}=\phi(s_{t+1})$

    Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j,a_j,r_j,\phi_{j+1})$ from D

    Set $y_j=\begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j+\gamma\max_{a'}\hat{Q}(\phi_{j+1},a';\theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j-Q(\phi_j,a_j;\theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q}=Q$

  End For

End For

装订线

四、操作方法与实验步骤

（一）实现深度优先搜索算法

使用深度优先算法实现从迷宫起点到迷宫终点路线的搜索。

使用 DFS 递归算法，使用一个栈存储当前路径，并使用全局数组 visited 记录已经访问过的位置。

每到达一个新的位置，首先检查该位置是否访问过，若未访问过，则将该位置进行标记。在当前能够进入的下一位置的选项中，选择还未访问过的位置进入，进入的位置压栈。若当前位置为迷宫终点，则返回 1，程序结束，若当前位置不是终点且没有可以继续访问的相邻位置，则返回 0，从栈中弹出当前位置，直到回退到上一个可选择的分叉为止。

调用该 DFS 递归算法，即可得到迷宫从起点到终点的路径 path。

算法代码如下：

```
def DFS(maze: Maze, now_pos, path: list, visited: list):
    if now_pos == maze.destination:
        return 0
    visited.append(now_pos)
    for action in maze.can_move_actions(now_pos):
        next_pos = tuple(map(sum, zip(now_pos, maze.move_map[action])))
        if not next_pos in visited: # 下一个位置未被访问过
            path.append(action)
            if DFS(maze, next_pos, path, visited):   # 进入下一个位置
                path.pop()
            else:
                return 0
    return 1
```

（二）实现 DQN 算法

实现 DQN 算法，首先应该设置地图中不同 action 的奖励函数 reward。

在本次实验中，“撞墙”设置为+100，“目的地”设置为-500，“普通的移动”设置为-0.1。

为了使机器人能够在初期尽可能的探索地图，并且确保能够至少抵达一次终点，机器人在探索状态下的模式设置为“explore”，此时每一步的随机概率设置为 0.5，以免于局部收敛而陷入死循环无法抵达终点。

若机器人已经成功探索到终点了，则将机器人设置为“train”模式，进行 Q 值表的训练。

但在训练过程中，发现机器人依然容易出现局部收敛的状况，例如位置 A 的最佳 Q 值指向位置 B，而位置 B 的最佳 Q 值指向位置 A，这种情况容易导致机器人进入该位置后便无法正常探索地图，并且累计的循环无法有效的通过训练神经网络模型来避免这种情况。

为此，优化了奖励函数，增加“回到起点”的奖励设置为+500，之后有效避免了机器人在起点附近往复循环的问题。同时使用一个队列记录了机器人前两个抵达过的位置，若出现多次重复在几个位置中循环的情况，将对机器人的移动施加惩罚，从而避免该情况的出现。

（三）强化学习网络模型设置

DQN 面临着几个挑战：

1. 深度学习需要大量带标签的训练数据；

2. 强化学习从 scalar reward 进行学习，但是 reward 经常是 sparse, noisy, delayed;

3. 深度学习假设样本数据是独立同分布的，但是强化学习中采样的数据是强相关的

因此，DQN 采用经验回放（Experience Replay）机制，将训练过的数据进行储存到 Replay Buffer 中，以便后续从中随机采样进行训练，好处就是：数据利用率高；减少连续样本的相关性，从而减小方差（variance）。

在本次强化学习模型中同样采用了经验回放机制，训练时每次随机抽取 batch 的样本数据进行训练，同时使用两个同步的神经网络，一开始两个神经网络的参数是相同的，其中一个参数相对固定的就是 target-net，用来获取 Q-target 值，而另一个不断更新的网络用来获取 Q-估计值，但每过 N 次动作后都会将估计网络的参数同步给目标网络，所以称 Target-network 是相对固定的. 这样做的原因是因为，Q-target 和 Q 估计两者之间的差就是损失函数，我们就是要使两者的差不断缩小，所以需要将 target 相对固定住这样方便收敛。

神经网络结构使用简单的深度神经网络，将二维输入（x，y）映射为四维向量即四个方向的 Q 值。

神经网络结构如下：

```
class QNetwork(nn.Module, ABC):
    """Actor (Policy) Model."""
    def __init__(self, state_size: int, action_size: int, seed: int):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
        """
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.input_hidden = nn.Sequential(
            nn.Linear(state_size, 512),
            nn.ReLU(False),
            nn.Linear(512, 512),
            nn.ReLU(False),
        )
        self.final_fc = nn.Linear(512, action_size)
    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = self.input_hidden(state)
        return self.final_fc(x)
```

（四）强化学习模型训练过程

首先在训练初期，将机器人行为选择的随机概率调到较高，以便于机器人更好的探索地图。并且概率每轮逐渐收敛。初始概率 epsilon0 设置为 0.8。

对机器人中的 DQN 网络模型进行多轮训练，使用成员变量 train_update 进行模型参数的更新，使用 batch 随机采样中的期望值 Q-target 和预测值 Q-expect 进行比较。

优化器 optimizer 使用 Adam 优化器；

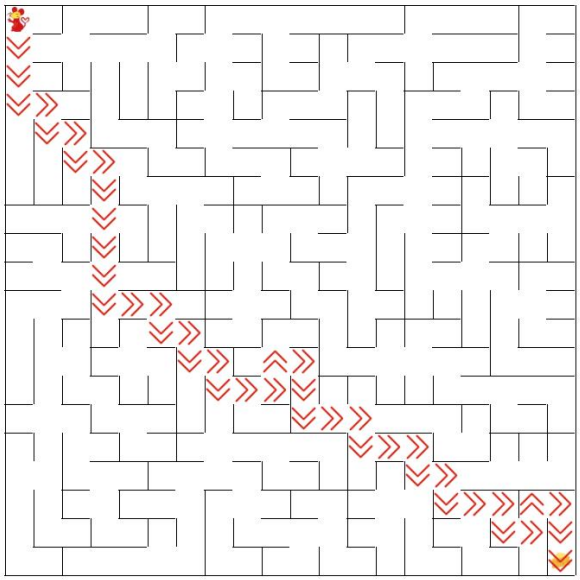
损失函数 criterion 使用交叉熵函数 CrossEntropyLoss()。

训练完毕后，调用成员变量 test_update 进行 Q 值表生成路径的测试，若成功，则模型训练完毕，Q 值表能够正确指示到达终点的路径。若未成功抵达终点，说明 DQN 神经网络模型未能完全收敛，则需要继续进行训练。

五、实验数据记录和处理

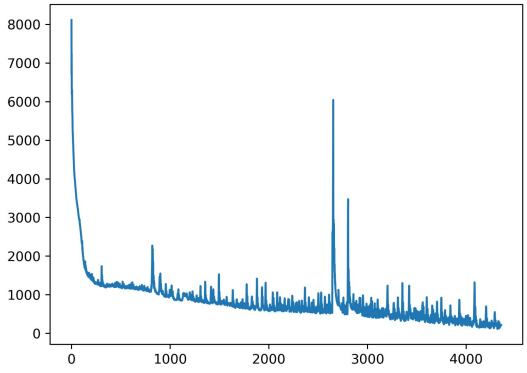
（一）深度优先算法

运行结果如下：

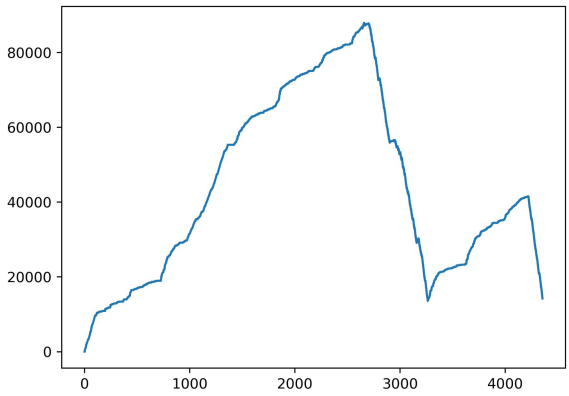


（二）DQN 算法

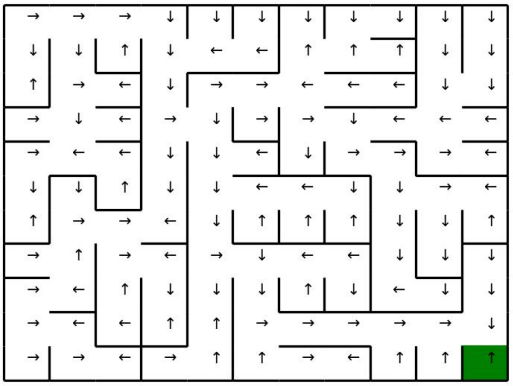
以较大的地图尺寸 maze_size = 11 为例，DQN 训练过程中，每轮训练过程中 Q 值表真实值和预测值之间的误差变化如下，可见随着训练过程的增加，误差趋于减小，说明 DQN 神经网络正在不断逼近真实的地图真值表。



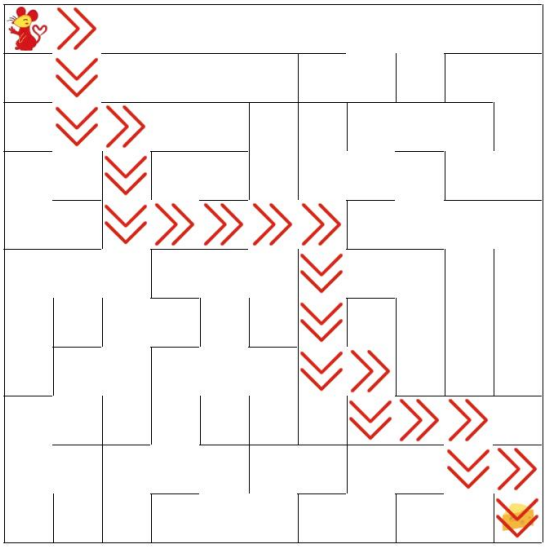
训练过程中，机器人从环境中的得到的累计奖励 reward 如下图所示，在前半段训练中由于机器人一直未知终点的位置，也从未抵达终点，而后呈现下降趋势。最终较佳的路径应该有尽可能小的累计奖励，说明在路径决策上的失误（如撞墙、死循环等）显著减小。



根据训练完成后 DQN 神经网络给出的地图 Q 值表，给出地图中每一个位置上最佳的位置选择，结果如图所示，可见该训练后的神经网络生成的 Q 值表能够成功给出迷宫的正确路径。



在测试系统中，该 DQN 算法也成功完成了迷宫并通过了测试。



七、讨论、心得

通过本次实验，我对机器学习训练深度模型的基本步骤有了更深的理解，同时掌握了使用 pytorch 框架进行深度学习框架搭建的流程和原理。

在强化学习训练初期，出现了较为严重的局部收敛问题，后续通过改进奖励函数和模型结构以及训练过程，成功较好的解决了该问题。

同时，我对 bellman 方程有了更深的理解，并且对 DQN 算法的流程和调参过程有了进一步的掌握。

有一些场景下，我们希望学习到一些随机的策略。譬如在德州扑克中，如果智能体采取策略是固定性的，那么很容易因为被对手预测到行为模式而被针对；虽然基于学习到的表后,我们也可以采用贪心的方式使得学到目标策略获得随机性，但却无法直接优化动作的概率分布。而基于策略的方法，比如 Policy Gradient 可以直接优化动作的概率分布。