中山大学计算机学院
人工智能
本科生实验报告

（2023 学年春季学期）

课程名称：Artificial Intelligence

教学班级	刘咏梅老师班级	专业（方向）	信息与计算科学
学号	22336313	姓名	郑鸿鑫

一、 实验题目

用 Deep Q-learning Network(DQN)玩 CartPole-v1 游戏

二、 实验内容

1. 算法原理

通过训练一个深度神经网络来近似 Q 函数，该网络能够根据当前的游戏状态预测每个可能动作的期望未来回报。在每个时间步骤，DQN 算法会选择一个动作，通常是通过 ε-贪婪策略，即大部分时间选择当前估计的最优动作，小部分时间随机选择动作以探索新的可能性。然后，算法会根据实际获得的奖励和新状态来更新 Q 值估计，这个过程涉及到经验回放和目标网络，以减少训练过程中的不稳定性并提高学习效率。随着时间的推移，网络逐渐学习到在给定状态下应该采取的动作，以最大化累积奖励，即让小车保持平衡并尽可能长时间地保持杆子竖直。

2. 伪代码

```
Procedure InitializeDQN(env, input_size, hidden_size, output_size)
    Define eval_net As QNet With input_size, hidden_size, output_size
    Define target_net As QNet With Same Parameters As eval_net
    Define optim As Adam Optimizer With eval_net Parameters And Learning Rate lr
    Define gamma As Discount Factor
```

```
    Define buffer As ReplayBuffer With Capacity
    Define loss_fn As MSELoss
    Define learn_step As 0
    Define eps As Initial Exploration Rate
    Define eps_min As Minimum Exploration Rate
    Define eps_decay As Exploration Rate Decay Factor
EndProcedure
Procedure DecayEpsilon(eps, eps_min, eps_decay)
    If eps > eps_min Then
        eps <- eps * eps_decay
        eps <- Max(eps, eps_min)
    EndIf
EndProcedure
Function ChooseAction(agent, obs)
    agent.DecayEpsilon()
    If Random() > eps Then
        Return Argmax Of agent.eval_net(obs) Without Gradients
    Else
        Return Random Action From env.action_space
    EndIf
EndFunction
Procedure StoreTransition(agent, obs, action, reward, next_obs, done)
    agent.buffer.push(obs, action, reward, next_obs, done)
EndProcedure
Procedure Learn(agent, args)
    If agent.buffer.len() < args.batch_size Then
        Return
    EndIf
    sample obs, actions, rewards, next_obs, dones From agent.buffer
    Compute q_eval, q_next, q_target For Sampled Transitions
    Calculate loss Using agent.loss_fn And Backpropagate Using agent.optim
    If learn_step Mod args.update_target = 0 Then
        Copy Weights From agent.eval_net To agent.target_net
    EndIf
    learn_step <- learn_step + 1
EndProcedure
Procedure Main(args)
    Initialize Environment And Agent
    Initialize episode_rewards And average_rewards Lists
    For i From 1 To args.n_episodes Do
        obs <- env.reset()
        episode_reward <- 0
        done <- False
        step_cnt <- 0
        While Not done And step_cnt < 500 Do
            action <- ChooseAction(agent, obs)
            next_obs, reward, done, info <- env.step(action)
            StoreTransition(agent, obs, action, reward, next_obs, done)
            episode_reward <- episode_reward + reward
            obs <- next_obs
            If agent.buffer.len() >= args.batch_size Then
                Learn(agent, args)
            EndIf
```

```
            step_cnt <- step_cnt + 1
        EndWhile
        Append episode_reward To episode_rewards
        Calculate And Append Average Reward To average_rewards
        Print Episode Information
    EndFor
    Plot episode_rewards And average_rewards
EndProcedure
Procedure PlotRewards(episode_rewards, average_rewards)
    Use Matplotlib To Plot episode_rewards And average_rewards
    Show Plot
EndProcedure
```

3. 关键代码展示（带注释）

- **神经网络代码：**

```python
class QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(QNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)
        #两层全连接层，参数为输入输出的大小
    def forward(self, x):
        x = torch.Tensor(x)
        x = F.relu(self.fc1(x))
         #激活函数，小于 0 时输出 0，大于 0 时输出原本的值
        x = self.fc2(x)
        return x
```

- **经验回放缓冲区代码：**

```python
class ReplayBuffer:#经验回放缓冲区
    def __init__(self, capacity):
        self.buffer = collections.deque(maxlen=capacity)
    #capacity 为缓冲区的最大容量，存储的最大转换数量
    #buffer 是用 collection.deque 创建的双端队列，新元素加入时如果队满，将会自
动删除最早加入的元素
    def len(self):
        return len(self.buffer)
    def push(self, *transition):
        self.buffer.append(transition)
    #将新的 transition 转入到缓冲区的末端
    def sample(self, batch_size):
        transitions = random.sample(self.buffer, batch_size)
        obs, actions, rewards, next_obs, dones = zip(*transitions)
        return np.array(obs), actions, rewards, np.array(next_obs),
dones
```

```
    #从缓冲区中随机采样 batch_size 个 transition，并最终转化为 Numpy 数组
    def clean(self):
        self.buffer.clear()
    #清空缓冲区中的所有元素
```

- **DQN 强化学习网络代码：**

对于 q_target 时使用 target_net 和目标网络不是每次都更新的说明：

目标网络

- $\left(r+\gamma \max_{a'} Q(s',a')\right)$可看作目标值，目标值跟随Q一直变化会给训练带来困难
- 将评估网络与目标网络分开，目标网络不训练，评估网络每更新若干轮后，用评估网络参数替换目标网络参数
- $L=\frac{1}{2}\left(Q_{eval}(s,a)-\left(r+\gamma \max_{a'} Q_{target}(s',a')\right)\right)^2$

```
class DQN:
    def __init__(self, env, input_size, hidden_size, output_size):
        self.env = env
        self.eval_net = QNet(input_size, hidden_size, output_size) #用于价值评估和动作选择
        self.target_net = QNet(input_size, hidden_size, output_size)#目标网络，定期从 eval_net 中复制权重
        self.optim = optim.Adam(self.eval_net.parameters(), lr=args.lr)
#使用 Adam 优化器
        self.gamma = args.gamma #折扣因子
        self.buffer = ReplayBuffer(args.capacity)
        self.loss_fn = nn.MSELoss() #损失函数
        self.learn_step = 0 #计数器，记录更新次数
        self.eps_decay = args.eps_decay
        self.eps = args.eps  # 初始探索率
        self.eps_min = args.eps_min  # 最小探索率
    def decay_eps(self):
        # 衰减探索率 ε 的值
        if self.eps > self.eps_min:
            self.eps *= self.eps_decay
            self.eps = max(self.eps, self.eps_min)
    def choose_action(self, obs):#动作选择函数
        self.decay_eps()  # 衰减 ε 值
```

```
        if random.random() > self.eps:
            with torch.no_grad():
                return self.eval_net(obs).argmax().item()
        else:
            return self.env.action_space.sample()
        #if-else 语句的解释：有 ε 的可能性随机探索动作，1-ε 可能性会计算 Q 值后
选取最佳动作

    def store_transition(self, *transition):
        self.buffer.push(*transition)
        #存储 transition 到经验回放缓冲区
    def learn(self):#训练网络
        if self.buffer.len() < args.batch_size:
            return
        obs, actions, rewards, next_obs, dones =
self.buffer.sample(args.batch_size)
        obs = torch.FloatTensor(obs)
        actions = torch.LongTensor(actions)
        rewards = torch.FloatTensor(rewards)
        next_obs = torch.FloatTensor(next_obs)
        dones = torch.FloatTensor(dones)
        #采样数据并转换为 torch 张量
        q_eval = self.eval_net(obs).gather(1, actions.view(-1,
1)).squeeze(1)
        #计算当前状态 Q 值
        q_next = self.target_net(next_obs).max(1)[0].detach()
        q_target = rewards + self.gamma * q_next * (1 - dones)
        #计算目标 Q 值
        loss = self.loss_fn(q_eval, q_target)
        self.optim.zero_grad()
        loss.backward()
        self.optim.step()
        #计算损失并反向传播
        if self.learn_step % args.update_target == 0:
            self.target_net.load_state_dict(self.eval_net.state_dict())
            #定期用 eval 网络的参数来更新 target 网络的参数
        self.learn_step += 1
```

- **对模型进行训练并记录 reward 变化代码：**

```
def main():
    env = gym.make(args.env)
    o_dim = env.observation_space.shape[0] #环境的观察空间维度
    a_dim = env.action_space.n #动作空间的数量
```

```
    agent = DQN(env, o_dim, args.hidden, a_dim) #初始化智能体
    episode_rewards = []
    average_rewards = []
    #两个用于存储回报的列表
    for i_episode in range(args.n_episodes):#对于每个训练周期
        obs = env.reset() #重置环境
        episode_reward = 0
        done = False #该轮是否结束的标志
        step_cnt = 0 #记录该轮步数
        while not done and step_cnt < 500:
            step_cnt += 1 #每做一个动作步数会自增，超过 500 时游戏胜利
            env.render() #渲染环境
            action = agent.choose_action(obs) #选择动作
            next_obs, reward, done, info = env.step(action)
            agent.store_transition(obs, action, reward, next_obs, done)
            #将信息存储到经验回放缓冲区
            episode_reward += reward
            obs = next_obs #更新 reward 和 obs
            if agent.buffer.len() >= args.batch_size:
                agent.learn()
                #经验回放区中数据足够时进行学习
        episode_rewards.append(episode_reward)
        if len(episode_rewards) >= 100:
            average_reward = np.mean(episode_rewards[-100:])
            average_rewards.append(average_reward)
        else:
            average_rewards.append(np.mean(episode_rewards))
        #计算每一百局内的平均 reward
        print(f"Episode: {i_episode}, Reward: {episode_reward}, Average
Reward (last 100): {average_rewards[-1]}")
        #每次训练周期结束后打印信息
    # 绘制奖励曲线
    plt.figure(figsize=(12, 5))
    plt.subplot(121)
    plt.plot(episode_rewards, label='Episode Reward')
    plt.xlabel('Episode')
    plt.ylabel('Reward')
    plt.legend()
    plt.subplot(122)
    plt.plot(average_rewards, label='Average Reward (last 100
episodes)')
    plt.xlabel('Episode')
    plt.ylabel('Average Reward')
```

```
    plt.legend()
    plt.tight_layout()
    plt.show()
```

- **超参数数据：**

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--env",        default="CartPole-v1",  type=str,
help="environment name")
    parser.add_argument("--lr",           default=1e-3,
type=float, help="learning rate")
    parser.add_argument("--hidden",        default=64,        type=int,
help="dimension of hidden layer")
    parser.add_argument("--n_episodes",    default=500,        type=int,
help="number of episodes")
    parser.add_argument("--gamma",         default=0.99,
type=float, help="discount factor")
    # parser.add_argument("--log_freq",      default=100,
type=int)
    parser.add_argument("--capacity",      default=10000,     type=int,
help="capacity of replay buffer")
    parser.add_argument("--eps",           default=0.2,
type=float, help="epsilon of ε-greedy")
    parser.add_argument("--eps_min",       default=0.001,
type=float)
    parser.add_argument("--batch_size",    default=128,        type=int)
    parser.add_argument("--eps_decay",     default=0.95,
type=float)
    parser.add_argument("--update_target",  default=100,       type=int,
help="frequency to update target network")
    args = parser.parse_args()
    #深度强化学习中用到的超参数
    main()
```

4. 创新点&优化（如果有）

首先解释探索率 ε 的概念：有 ε 的可能性随机探索动作，1- ε 可能性会计算 Q 值后选取最佳动作。我们一开始采用的是固定的探索率 ε 值为 0.01，发现学习速度较慢，虽然可以达到连续 10 局 500 分的要求，但是对于进阶要求，近 100 局平均要达到 475 的要求还是没

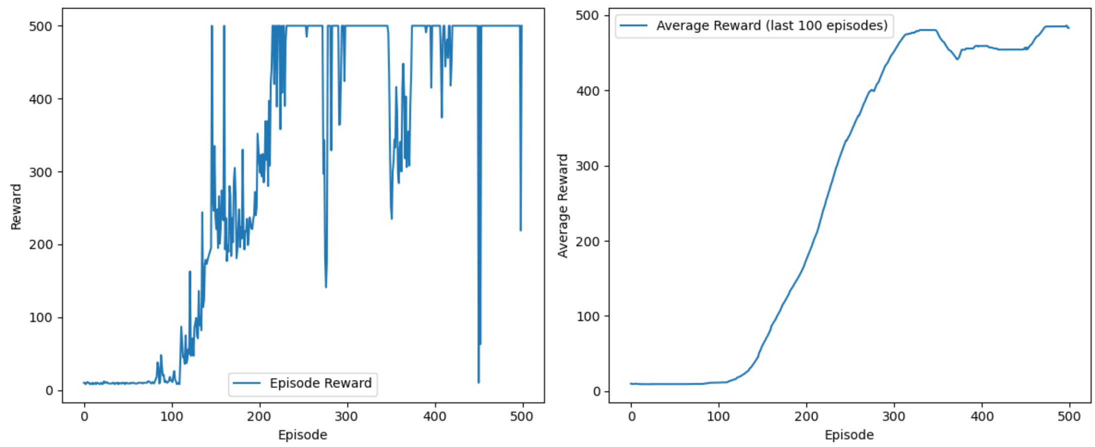
有实现（具体结果看实验结果展示和评测指标）。

考虑到在强化学习的初期，智能体需要探索各种可能的动作来提供经验回放的数据等，而随着智能体对环境的学习的增加，它可以更多的去依赖自己的知识，这时候探索率应该下降。而且固定的探索率太高会导致难以收敛，太低会导致陷入局部最优解。

所以我们采用定期衰减的探索率 ε 来对模型进行训练。再通过对超参数进行略微调整后实验，最终达成了实验要求，并且收敛的速度也比较理想（详见实验结果展示）

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）



最早出现连续 10 局 500 分的时间点：

```
Episode: 231, Reward: 500.0, Average Reward (last 100): 285.43
Episode: 232, Reward: 500.0, Average Reward (last 100): 289.54
Episode: 233, Reward: 500.0, Average Reward (last 100): 293.54
Episode: 234, Reward: 500.0, Average Reward (last 100): 297.72
Episode: 235, Reward: 500.0, Average Reward (last 100): 300.28
Episode: 236, Reward: 500.0, Average Reward (last 100): 304.14
Episode: 237, Reward: 500.0, Average Reward (last 100): 307.9
Episode: 238, Reward: 500.0, Average Reward (last 100): 311.17
Episode: 239, Reward: 500.0, Average Reward (last 100): 314.38
Episode: 240, Reward: 500.0, Average Reward (last 100): 317.65
Episode: 241, Reward: 500.0, Average Reward (last 100): 320.87
```

第一次达到近 100 局的平均 reward 超 475 的时间点：

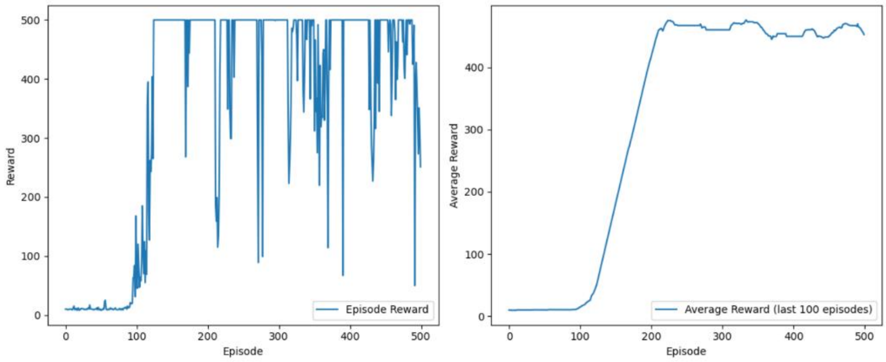
```
Episode: 316, Reward: 500.0, Average Reward (last 100): 474.75
Episode: 317, Reward: 500.0, Average Reward (last 100): 475.55
Episode: 318, Reward: 500.0, Average Reward (last 100): 475.55
Episode: 319, Reward: 500.0, Average Reward (last 100): 475.55
Episode: 320, Reward: 500.0, Average Reward (last 100): 476.66
Episode: 321, Reward: 500.0, Average Reward (last 100): 476.66
Episode: 322, Reward: 500.0, Average Reward (last 100): 476.66
Episode: 323, Reward: 500.0, Average Reward (last 100): 476.66
Episode: 324, Reward: 500.0, Average Reward (last 100): 478.08
Episode: 325, Reward: 500.0, Average Reward (last 100): 478.08
Episode: 326, Reward: 500.0, Average Reward (last 100): 479.0
Episode: 327, Reward: 500.0, Average Reward (last 100): 479.0
Episode: 328, Reward: 500.0, Average Reward (last 100): 479.0
Episode: 329, Reward: 500.0, Average Reward (last 100): 480.1
```

近 100 局的平均得分的峰值为：483.92

```
Episode: 496, Reward: 500.0, Average Reward (last 100): 485.92
Episode: 497, Reward: 500.0, Average Reward (last 100): 485.92
```

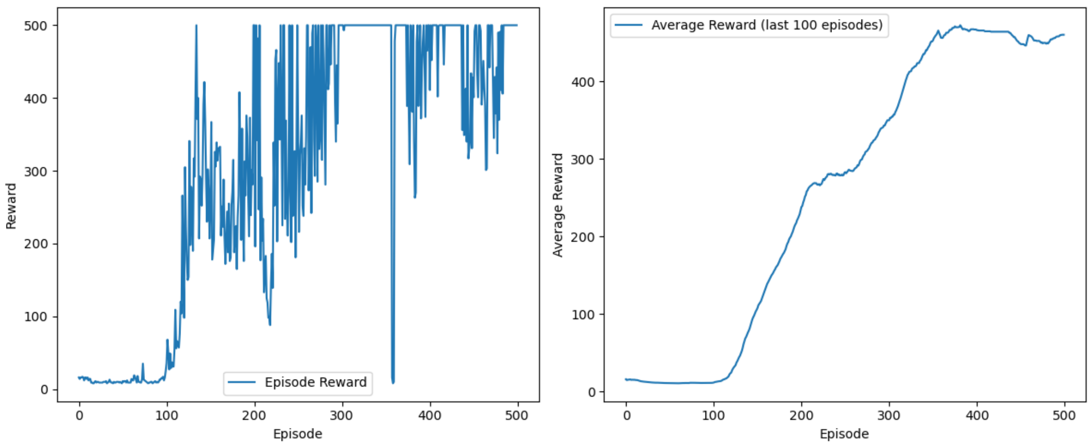
2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

首先给出探索率 ε 为 0 的结果作为对照(即每次都按照评价网络选取最优的动作)

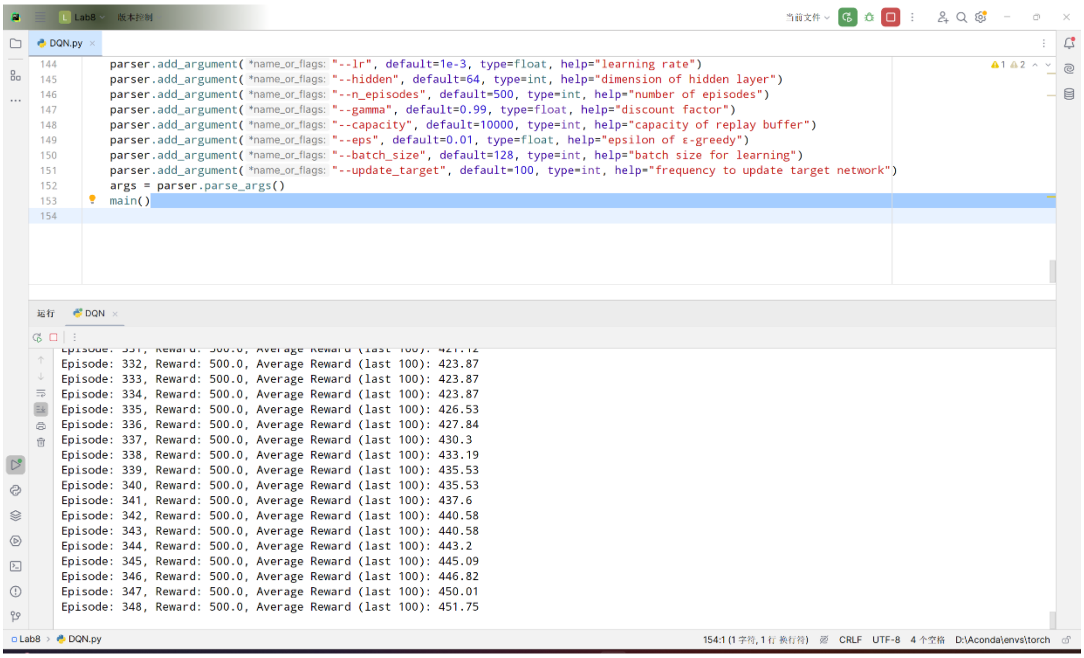


可以看到虽然网络可以完成基础要求（达成连续 10 局 500 分），但是 reward 的波动非常明显，有大量的从 500 到 10 多分这样的波动，这显然是不理想的。原因在于缺少对于环境的探索，经验回放的数据相对局限，所以我们给到一个固定的 ε 进行实验。

探索率 ε 固定为 0.01 结果如下：



可以看到加入了探索率后收敛到 500 分的速度变慢了，但是收敛后的波动少了很多，而且近 100 局的 reward 平均值也有所提升，不过最高值只达到了 450 左右，离 475 还存在差距。



最后我们采用定期衰减的探索率 ε 来进行强化学习。最初我们将探索率设置为 0.5，衰减率为 0.999，然后实验后发现效果不好并且收敛速度也很慢。通过不断尝试我们最终选定一组较好的参数设置如下：

将初始探索率定为 0.2，然后每次做一次动作选择就让探索率衰减到原来的 95%，但是不能超过给定的最小值，最后我们的最小值设置为 0.001，如下所示：

```
def decay_eps(self):
        # 衰减探索率 ε 的值
        if self.eps > self.eps_min:
            self.eps *= self.eps_decay
            self.eps = max(self.eps, self.eps_min)
parser.add_argument("--eps", default=0.2, type=float, help="epsilon of ε-greedy")
    parser.add_argument("--eps_min",default=0.001, type=float)
parser.add_argument("--eps_decay", default=0.95, type=float)
```

最终得到的结果在 1.实验结果展示示例中已经给出故不重复。可以看到后续的波动已经很少（由于探索率的存在还是难免会出现），并

且连续 10 局以上的 500 分还有近 100 局平均 reward 超过 475 也都已实现，最高可以达到 485。至此，实验圆满完成。

四、 参考资料

1） 动手学强化学习（七）：DQN 算法 - jasonzhangxianrong - 博客园 (cnblogs.com)

2） 【深度强化学习】(1) DQN 模型解析，附 Pytorch 完整代码_dqn 模型-CSDN 博客

3） 深度强化学习(5_5)：AlphaGo_哔哩哔哩_bilibili