哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2024 春季

课程名称：人工智能（实验）

实验名称：实验三强化学习

实验性质：综合设计型

实验学时：2 地点：

学生班级：3

授课教师：

实验与创新实践教育中心制

2024 年 5 月

一、实验环境

描述操作系统、开发环境（CPU\GPU）、使用的库等。

操作系统：windows

开发环境：GPU

使用库：gymnasium、matplotlib、pytorch

```
device: cuda
```

二、实验过程和结果分析

2.1 初始代码运行结果

运行 reinforcement_q_learning.ipynb 并将结果截图

```python
end_time = time.time()
print('Complete')
print('The training time is ', (end_time - start_time), 's')
```

```
Episode 596, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  497.0110557079315 s

<Figure size 640x480 with 0 Axes>
```

2.2 优化代码及运行结果

可从神经网络结构的优化、超参数调优、优化经验回放区、奖励函数的设计、探索策略的设计等方面着手，挑选 4 个方向进行优化，并分析对比结果。

1. 优化一

（1） 优化代码描述

代码截图粘贴于此，并简单描述优化内容。

```python
class DQN(nn.Module):

    def __init__(self, n_observations, n_actions):
        super(DQN, self).__init__()
        self.layer1 = nn.Linear(n_observations, 256)
        self.layer2 = nn.Linear(256, 256)
        self.layer3 = nn.Linear(256, n_actions)

    # Called with either one element to determine next action, or a batch
    # during optimization. Returns tensor([[left0exp,right0exp]...]).
    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        return self.layer3(x)
```

神经网络结构优化，当前的网络结构已经包含了两个隐藏层，每层 128 个神经元。

进一步优化网络结构，增加层数、改变每层的神经元数量为 256。

（2） 运行结果截图

```
Episode 343, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  216.2202968597412 s

<Figure size 640x480 with 0 Axes>
```

（3） 对比分析

与初始代码结果对比分析，从训练速度、收敛效果等方面进行分析。

```python
end_time = time.time()
print('Complete')
print('The training time is ', (end_time - start_time), 's')
```

```
Episode 596, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  497.0110557079315 s

<Figure size 640x480 with 0 Axes>
```

可以看到和初始代码结果相比训练速度有显著提升，同时所需次数也相应减少。

2. 优化二

（1） 优化代码描述

截图代码粘贴于此，并简单描述优化内容。

超参数对强化学习的效果有显著影响，可以通过网格搜索或随机搜索来找到最优的超参数组合。参数调整：

增加经验回放缓冲区的容量。

调整学习率和优化器参数。

调整批次大小（BATCH_SIZE），较小的批次大小可能会更稳定，但较大的批次大小可能会加快训练速度。

调整目标网络更新频率（TAU）。

```
# LR is the learning rate of the AdamW optimizer
BATCH_SIZE = 64
GAMMA = 0.99
EPS_START = 1.0
EPS_END = 0.01
EPS_DECAY = 500
TAU = 0.01
LR = 5e-4

# Get number of actions from gym action space
```

（2） 运行结果截图

```
Episode 248, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  154.45833659172058 s

<Figure size 640x480 with 0 Axes>
```

（3） 对比分析

与初始代码结果对比分析，从训练速度、收敛效果等方面进行分析。

如图所示，运行结果相比初始代码结果显著提升，训练速度和收敛效果都有增进。

学习率提高可以加速模型的学习，但过高的学习率可能导致不稳定。通过调整为 5e-4 可以在稳定性和速度之间找到更好的平衡。

批量大小减小可以增加更新的频率，从而加速学习。

更快的 ε 衰减可以加速从探索到利用的转换，提升训练效率。

目标网络更新率增加使得目标网络更新更频繁，有助于稳定训练。

```
end_time = time.time()
print('Complete')
print('The training time is ', (end_time - start_time), 's')
```

```
Episode 596, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  497.0110557079315 s

<Figure size 640x480 with 0 Axes>
```

```
Episode 248, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  154.45833659172058 s

<Figure size 640x480 with 0 Axes>
```

3. 优化三

（1） 优化代码描述

代码截图粘贴于此，并简单描述优化内容。

通过以下方法优化经验回放区：使用优先经验回放（Prioritized Experience Replay），优先选择高 TD 误差的样本进行训练。增加经验回放缓冲区的容量，以提供更多的训练数据。通过使用缓冲区中的样本进行多次梯度更新，提高数据利用率。

```python
class PrioritizedReplayMemory:
    def __init__(self, capacity, alpha=0.6):
        self.capacity = capacity
        self.alpha = alpha
        self.memory = deque([], maxlen=capacity)
        self.priorities = deque([], maxlen=capacity)

    def push(self, *args):
        max_priority = max(self.priorities, default=1.0)
        self.memory.append(Transition(*args))
        self.priorities.append(max_priority)

    def sample(self, batch_size, beta=0.4):
        if len(self.memory) == 0:
            return [], [], []

        probs = np.array(self.priorities) ** self.alpha
        probs /= probs.sum()
        indices = np.random.choice(len(self.memory), batch_size, p=probs)
        samples = [self.memory[idx] for idx in indices]

        weights = (len(self.memory) * probs[indices]) ** (-beta)
        weights /= weights.max()

        return samples, weights, indices

    def update_priorities(self, batch_indices, batch_priorities):
        for idx, priority in zip(batch_indices, batch_priorities):
            self.priorities[idx] = priority

    def __len__(self):
        return len(self.memory)
```

（2） 运行结果截图

```
Episode 132, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  103.97511005401611 s

<Figure size 640x480 with 0 Axes>
```

（3） 对比分析

与初始代码结果对比分析，从训练速度、收敛效果等方面进行分析。

```
end_time = time.time()
print('Complete')
print('The training time is ', (end_time - start_time), 's')
```

```
Episode 596, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  497.0110557079315 s

<Figure size 640x480 with 0 Axes>
```

```
Episode 132, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  103.97511005401611 s

<Figure size 640x480 with 0 Axes>
```

优先经验回放会优先选择高 TD 误差的样本进行训练，这样可以更快地减少 Q 值估计的误差，从而加速收敛。

通过优先经验回放，模型能更高效地学习到关键样本的信息，从而提升了训练效率。

4. 优化四

（1） 优化代码描述

代码截图粘贴于此，并简单描述优化内容。

现有的探索策略是简单的 ε -greedy 策略，改为使用 Boltzmann 策略，根据动作价值的软最大化选择动作。动态调整 ε 值，使其在训练初期快速衰减，以便更快地收敛。

```
def select_action(state):
    global steps_done
    sample = random.random()
    eps_threshold = EPS_END + (EPS_START - EPS_END) * math.exp(-1. * steps_done / EPS_DECAY)
    steps_done += 1
    if sample > eps_threshold:
        with torch.no_grad():
            return policy_net(state).max(1).indices.view(1, 1)
    else:
        q_values = policy_net(state)
        temperature = 1.0
        probabilities = F.softmax(q_values / temperature, dim=1)
        action = np.random.choice(n_actions, p=probabilities.cpu().numpy().ravel())
        return torch.tensor([[action]], device=device, dtype=torch.long)

episode_durations = []
```

（2） 运行结果截图

```
Episode 123, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  83.07986974716187 s

<Figure size 640x480 with 0 Axes>
```

（3） 对比分析

与初始代码结果对比分析，从训练速度、收敛效果等方面进行分析。

Boltzmann 策略根据动作的价值进行软选择，可以更平滑地从探索过渡到利用，更快找到最优策略，从而加速收敛。

```python
end_time = time.time()
print('Complete')
print('The training time is ', (end_time - start_time), 's')
```

```
Episode 596, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  497.0110557079315 s

<Figure size 640x480 with 0 Axes>
```

```
Episode 123, Cumulative Reward: 500.0, success_count: 10
Complete
The training time is  83.07986974716187 s

<Figure size 640x480 with 0 Axes>
```