中国科学院大学

University of Chinese Academy of Sciences

系统与计算神经科学 实验报告

融合脉冲神经网络与深度强化学习

学院：	未来技术学院	未来技术学院
专业：	神经生物学	神经生物学
学号：	202418020415010	202418020415005
姓名：	袁彧涵	刘科迪

2024 年 11 月 25 日

1 实验目的

随着人工智能技术的迅猛发展，强化学习（Reinforcement Learning,RL）和脉冲神经网络（Spiking Neural Network, SNN）正逐步成为模仿生物智能的核心研究方向。两者各具优势，在生物启发计算领域展现出广阔的应用潜力。

强化学习是一种通过试错机制和奖励信号驱动的算法，模拟了生物体行为学习的过程。其理论基础源自行为主义学习理论和生物学中的多巴胺信号模型，使得智能体能够在复杂环境中逐步优化行为策略。强化学习已在游戏 AI、机器人控制等领域取得了显著成就。

相比之下，脉冲神经网络以离散脉冲信号进行信息传递和处理，是当前最接近生物神经系统的计算模型。其稀疏激活和高能效计算特性，使其成为下一代神经网络的潜在方向，尤其在低功耗计算和硬件实现中具有巨大优势。

强化学习和脉冲神经网络分别从决策机制和神经计算两个角度模拟了生物智能的不同侧面。那么，当这两种方法结合在一起，会产生怎样的“化学反应”？这正是生物启发计算领域值得探索的重要问题。

因此，本实验目的在于探讨两者结合的潜力，采用深度 Q 脉冲网络解决经典强化学习问题——CartPole，尝试回答这一结合的表现和优势。

2 实验环境

- 操作系统：Windows 11
- 编程语言：Python 3.10
- 编程工具：Visual Studio Code
- 主要库及版本：Pytorch 2.5.1、 Gymnasium 1.0.0、SpikingJelly 0.0.0.0.14

3 实验原理

3.1 CartPole 任务

CartPole 是强化学习中的一个经典任务，常用于测试和验证各种强化学习算法的性能。它源于控制领域，具体任务是平衡一个倒立摆（Inverted Pendulum）系统。

CartPole 任务的环境是一个二维平面，上面有一个小车（Cart）和一个倒立摆（Pole）。小车可以在平面上左右移动，倒立摆可以在小车上方左右摆动。任务的目标是通过控制小车的移动，使得倒立摆保持垂直不倒下。

CartPole 任务的状态空间是连续的，包括小车的位置、速度、倒立摆的角度和角速度等。动作空间是离散的，包括向左移动、向右移动等。环境会根据智能体的动作和当前状态，返回奖励信号和下一个状态。

属性	内容
任务目标	控制小车移动，使杆子尽可能长时间保持直立。
终止条件	1. 杆子与垂直方向夹角超出 ±12°； 2. 小车超出轨道边界； 3. 达到最大时间步数。
状态空间	1. 小车的水平位置； 2. 小车的速度； 3. 杆子的角度； 4. 杆子的角速度。
动作空间	1. 向左施加固定力； 2. 向右施加固定力。
奖励函数	每个时间步获得 +1 奖励，直到任务终止。
难点	1. 连续状态空间的决策； 2. 动作后果有延迟； 3. 动态平衡特性对控制性能要求较高。

表 3.1: CartPole 任务描述

3.2 脉冲神经网络

脉冲神经网络是一种模拟生物神经元脉冲发放特性的计算模型。脉冲神经网络中的神经元以离散时间步长处理输入信号，并以脉冲形式传递信息，模拟了生物神经元的动作电位，与传统神经网络的连续激活方式有所不同。

3.2.1 IF（Integrate-and-Fire）模型

IF 模型是一种简单的脉冲神经元模型，描述了神经元在接受输入后累积膜电位，直到达到阈值时触发一个脉冲。模型过程如下：

1. 神经元接收输入信号，累积膜电位。
2. 当膜电位超过阈值时，神经元发放脉冲。
3. 神经元膜电位重置，开始新的计算。

IF 模型的动态变化由以下微分方程描述：

$$V(t) = V(t - \Delta t) + I(t)\Delta t$$

其中：

$V(t)$为神经元膜电位。

$I(t)$为输入电流。

Δt为时间步长。

3.2.2 LIF（Leaky Integrate-and-Fire）模型

LIF 模型是在 IF 模型的基础上加入了膜电位的漏失效应，更接近生物神经元的动态行为。

LIF 模型的动态变化由以下微分方程描述：

$$\tau_m \frac{dV(t)}{dt} = -(V(t) - V_{\text{rest}}) + R_m I(t)$$

其中：

$V(t)$为神经元膜电位。

V_{rest}为静息电位。

R_m为膜电阻。

$I(t)$为输入电流。

τ_m为膜时间常数。

LIF 模型具有以下特性：

1. 积分特性：当输入电流$I(t)$ 存在时，LIF 模型的膜电位会逐渐积累。电流强度越大，膜电位的积累速率越高，越快达到阈值电位。
2. 泄漏特性：没有输入电流时，膜电位会自动向静息电位衰减。泄漏过程模拟了生物神经元膜电阻带来的自然漏电现象，保证了模型在没有刺激时保持稳定。
3. 放电机制：膜电位达到阈值V_{th}时，模型产生动作电位并将膜电位重置，类似生物神经元的放电过程。
4. 稀疏放电：由于电位在衰减过程中需要足够的输入刺激才能达到阈值，因此 LIF 模型能够展现稀疏发放特性，使得神经网络能够高效地编码信息。

模型名称	IF 模型	LIF 模型
漏失效应	无	有
生物真实性	较低	较高
计算复杂度	低	中等

表 3.2: IF 和 LIF 模型比较

3.3 深度 Q 网络

深度 Q 网络（Deep Q-Network, DQN）是深度强化学习中的一种经典算法，它结合了深度学习和强化学习，用神经网络近似 Q 函数，从而能够在高维状态空间中进行决策。这一方法由 Google DeepMind 于 2015 年提出，并成功应用于 Atari 游戏中的强化学习任务中，展示了其强大的性能。

DQN 设计的目的在于解决传统 Q 学习算法无法处理高维状态空间的问题。传统 Q 学习使用一个表（Q 表）来存储每个状态-动作对的值，但当状态空间维度较高时，存储和更新这些值会变得不可行。DQN 使用深度神经网络作为函数逼近器来预测 Q 值，通过最大化 Q 值来选择最优动作。训练过程包括经验回放、固定 Q 目标和延迟更新等技术，使得算法更加稳定和高效。

Q 值是强化学习中的一个重要概念，表示在某个状态下采取某个动作的预期回报。其定义如下：

$$Q(s,a) = \mathbb{E}[R_t \mid s_t = s, a_t = a]$$

其中：

$Q(s,a)$为状态-动作对(s,a)的 Q 值。

R_t为奖励信号。

s_t为当前状态。

a_t为当前动作。

Q 值函数的更新公式如下：

$$Q(s,a) = Q(s,a) + \alpha\Big[r + \gamma \max_a Q(s',a) - Q(s,a)\Big]$$

其中：

$Q(s,a)$为状态-动作对(s,a)的 Q 值。

α为学习率。

r为奖励信号。

γ为折扣因子。

s'为下一个状态。

$\max_a Q(s',a)$为下一个状态的最大 Q 值。

可以看出，Q 值函数的更新是基于当前状态-动作对的奖励信号和下一个状态的最大 Q 值，通过不断迭代更新 Q 值函数，智能体能够逐步优化策略，实现最优决策。

同时，DQN 算法还引入了经验回放（Experience Replay）技术，DQN 将智能体经历的状态、动作、奖励和下一个状态存储在一个回放缓冲区中，训练时从中随机采样小批量样本。这种方法可以减少样本间的相关性，提高训练稳定性。目标网络（Target Network）技术则用于解决 DQN 中的估计偏差问题，通过固定目标网络的参数，减少目标 Q 值的波

动, 提高训练效果。动作选择策略通常采用 ε-贪心策略, 即以概率 ε 选择随机动作, 以概率 1-ε 选择当前最优动作。这种方法可以在探索和利用之间取得平衡，提高算法的收敛性。

DQN 算法的训练过程包括以下步骤：

1. 初始化深度神经网络和目标网络参数。
2. 从环境中获取初始状态。
3. 根据当前状态选择动作，更新状态和奖励。
4. 将状态、动作、奖励和下一个状态存储在经验回放缓冲区中。
5. 从经验回放缓冲区中随机采样小批量样本，更新神经网络参数。
6. 更新目标网络参数。
7. 重复步骤 2-6，直到达到终止条件。

训练过程中的损失函数基于 TD（Temporal Difference）误差定义如下：

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a} Q(s', a; \theta^{-}) - Q(s, a; \theta)\right)^2\right]$$

其中：

$L(\theta)$为损失函数。

r为奖励信号。

γ为折扣因子。

$\max_a Q(s', a; \theta^{-})$为目标网络的最大 Q 值。

$Q(s, a; \theta)$为当前网络的 Q 值。

θ为当前网络参数。

θ^{-}为目标网络参数。

通常采用梯度下降法更新神经网络参数，使得损失函数逐步减小，优化 Q 值函数。

4 实验步骤

4.1 环境搭建

考虑到实现方式的复杂性，本实验采用 Python 编程语言和 Pytorch 深度学习框架, 结合 SpikingJelly 脉冲神经网络框架和 Gymnasium 强化学习环境，搭建深度 Q 脉冲网络模型。

4.2 构建深度 Q 脉冲网络

首先，根据 CartPole 任务的状态空间和动作空间，构建深度 Q 脉冲网络模型。模型包括输入层、隐藏层和输出层，其中输入层接收环境状态信息，输出层输出动作值函数 Q 值，隐藏层采用脉冲神经元模拟神经网络的计算过程。

需要注意的是，在原本 ANN 网络中，我们使用 ReLU 激活函数，而在 SNN 网络中，我们使用 IF 或 LIF 模型的脉冲发放机制。因此，需要对网络结构和参数进行相应调整，以适应脉冲神经网络的特性。

在这里，我们简单的构建一个深度 Q 脉冲网络模型，包括输入层、隐藏层和输出层，其中隐藏层采用 IF 或 LIF 模型，输出层采用非脉冲神经元。模型结构如下：

```python
class DQSN(nn.Module):
 super().__init__()

 self.fc = nn.Sequential(
 layer.Linear(input_size, hidden_size),
 neuron.IFNode(),
 layer.Linear(hidden_size, output_size),
 NonSpikingLIFNode(tau=2.0),
 )

 self.T = T
```

由于 Q 值函数是连续的，而脉冲神经网络是离散的，因此需要对脉冲神经网络的输出进行适当处理，以获得连续的 Q 值函数。

一个简单的方法是将脉冲神经元的阈值设置为无穷大，使得神经元始不发放脉冲，而是输出神经元的电压作为连续的激活值。这样，我们可以得到连续的 Q 值函数，用于计算损失函数和优化器。

```python
class NonSpikingLIFNode(neuron.LIFNode):
    def __init__(self, *args, **kwargs):
        self.v_float_to_tensor(x)

        if self.training:
            self.neuronal_charge(x)
        else:
            if self.v_reset is None:
                if self.decay_input:
                    self.v = self.neuronal_charge_decay_input_reset0(
                        x, self.v, self.tau
                    )
                else:
                    self.v = self.neuronal_charge_no_decay_input_reset0(
                        x, self.v, self.tau
```

```
                    )
            else:
                if self.decay_input:
                    self.v = self.neuronal_charge_decay_input(
                        x, self.v, self.v_reset, self.tau
                    )
                else:
                    self.v = self.neuronal_charge_no_decay_input(
                        x, self.v, self.v_reset, self.tau
                    )
```

模型的前向传播过程则是单步的脉冲神经网络计算过程，根据输入状态信息，通过隐藏层编码信息，并在输出层以膜电位作为决策，最终输出 Q 值函数。

```python
def forward(self, x):
        for t in range(self.T):
            self.fc(x)

        return self.fc[-1].v
```

记忆回放部分则是从经验回放缓冲区中随机采样小批量样本，不需要额外的处理。

```python
class ReplayMemory(object):
    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0
        self.transition = namedtuple(
            "Transition", ("state", "action", "next_state", "reward")
        )
    def push(self, *args):
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = self.transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

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

至此，我们完成了深度 Q 脉冲网络模型的构建，包括网络结构、前向传播过程和记忆回放部分。接下来，我们将进行模型的训练和测试，验证深度 Q 脉冲网络在 CartPole 任务上的性能。

4.3 训练模型

在训练模型过程中，我们需要定义损失函数、优化器和训练参数，以及训练过程的具体步骤。损失函数采用 TD 误差，优化器采用 Adam 优化器，训练参数包括学习率、折扣因子、批量大小等。

属性	内容
损失函数	TD 误差
优化器	Adam 优化器
学习率	0.001
折扣因子	0.999
探索率起始值	0.9
探索率终止值	0.05
批量大小	128
训练轮数	500
目标网络更新频率	10
经验回放缓冲区大小	10000

表 4.1: 训练参数设置

接下来我们使用一个 DQNTrainer 类来实现模型的训练过程，包括初始化环境和智能体、训练过程的迭代更新等。

```python
class DQNTrainer:
    def __init__(
        self,
        env_name: str,
        model_class,
        hidden_size: int,
        t: int,
        use_cuda: bool,
        config: dict,
    ):
```

在类的初始化函数中，我们定义了环境名称、模型类、隐藏层大小、时间步长、是否使用 CUDA 等参数。

DQNTraier 类根据初始化参数，构建了环境、深度 Q 脉冲网络模型、目标网络、记忆回放缓冲区等组件，并定义了损失函数、优化器等训练过程中的关键组件。

同时，DQNTrainer 类还实现了训练过程的迭代更新，包括从环境中获取状态、选择动作、更新网络参数、更新目标网络参数等步骤。

我们先说明训练过程中选择动作的策略。在训练过程中，采用 ε-贪心策略，以一定的概率 ε 选择随机动作，以概率 1-ε 选择当前最优动作。随着训练的进行，逐渐减小 ε 值，早期更多探索，后期更注重利用，这样可以在探索和利用之间取得平衡，提高算法的收敛性。 算法伪代码如下：

算法 1： ε-贪心策略动作选择

计算 $\varepsilon_{\text{threshold}}$，公式如下：$\varepsilon_{\text{end}} + (\varepsilon_{\text{start}} - \varepsilon_{\text{end}}) * e^{\{-\frac{\text{steps}}{\varepsilon_{\text{decay}}}\}}$
将步数计数器加 1
如果 随机值 > $\varepsilon_{\text{threshold}}$ **那么**
    使用策略网络选择具有最高 Q 值的动作
    重置策略网络状态
    返回所选动作
否则
    从可用动作空间中随机选择一个动作
    返回随机动作
结束

其次，我们定义了更新网络参数的过程。在每一步中，我们从记忆回放缓冲区中随机采样小批量样本，计算 TD 误差，更新网络参数。同时，定期更新目标网络参数，以减少估计偏差，提高训练效果。

算法 2： 模型优化

如果 记忆库中的样本数量 < 批量大小 **则** 返回
从记忆库中采样 BATCH_SIZE 个转换样本，构成批量数据
将批量数据解包为以下变量：
    状态批次 $\text{state}_{\text{batch}}$
    动作批次 $\text{action}_{\text{batch}}$
    奖励批次 $\text{reward}_{\text{batch}}$
    非终止的下一个状态 $\text{non final}_{\text{next states}}$
计算当前策略网络的状态-动作值
初始化下一个状态值为零
如果 存在非终止状态：
    计算目标网络的最大状态值并分离梯度
    重置目标网络状态
计算期望的状态-动作值
使用 Huber 损失计算当前值与期望值的误差
清空优化器梯度，反向传播误差并裁剪梯度（范围为 $[-1, 1]$）
更新策略网络参数，重置策略网络状态

最后，我们定义了训练过程的迭代更新，包括初始化环境和智能体、进行训练过程、更新网络参数等步骤。

需要注意的是，在很多 DQN 的实现过程中，为了获取当前的状态，会加入 CNN 网络结构，利用图片两帧之间的信息作为特征，提取图像信息，这里我们简化了模型结构，将动作空间直接输入模型，只使用全连接层，以便更好地展示脉冲神经网络的特性。

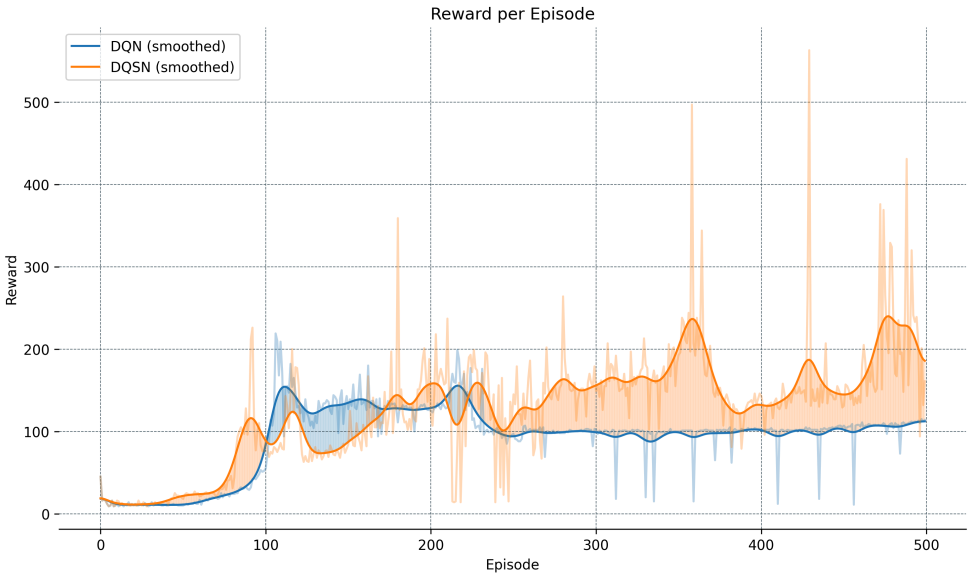
算法 3： 训练 DQN 智能体

```
初始化用于日志记录的 SummaryWriter
设置最大奖励为零
对于 每个回合在总回合数中
    重置环境，将状态初始化为全零张量，设置总奖励为零
    当 回合未终止且未被截断 时
        根据当前状态选择动作
        在环境中执行动作并获得：
            下一个状态、奖励、终止标志或截断标志
        累加奖励
        将下一个状态转换为张量并移动到设备
        如果 已终止或已截断 则
            将下一个状态设为空
        将转换（状态，动作，下一个状态，奖励）存储到记忆中
        更新状态为下一个状态并优化模型
        如果 已终止或已截断 则
            记录总奖励、损失到日志
            如果 总奖励 > 最大奖励 则
                更新最大奖励
                保存当前策略网络为最佳模型
            结束
            跳出循环
        结束
    结束
    如果 当前回合数是目标网络更新的倍数 则
        更新目标网络
    结束
关闭 SummaryWriter
保存最终的策略网络
```

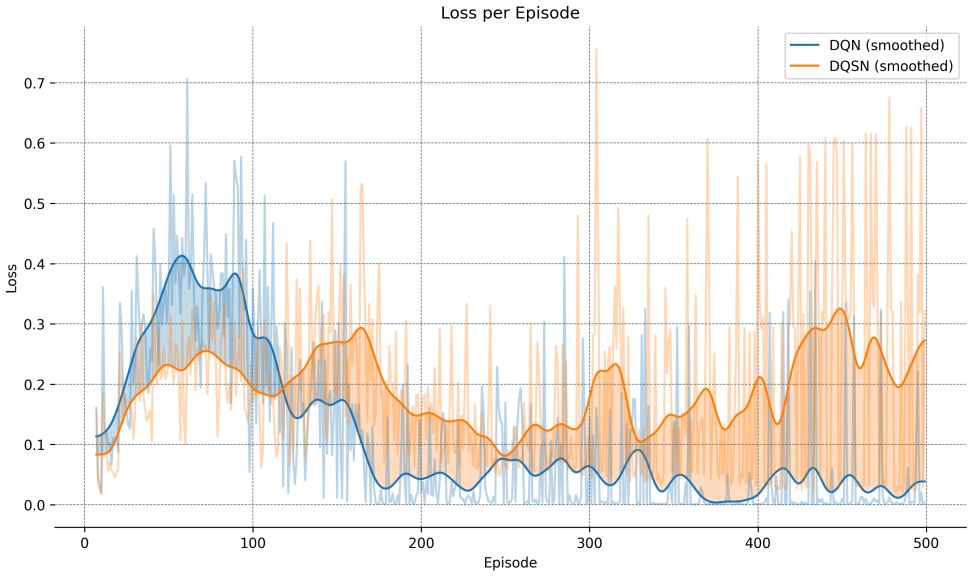
通过以上训练过程，我们可以逐步优化深度 Q 脉冲网络模型，提高在 CartPole 任务上的性能表现。

5 实验结果

在实验中，我们尝试使用深度 Q 脉冲网络模型解决 CartPole 任务，通过训练和测试，评估模型在任务中的性能表现。我们记录了训练过程中的总奖励和损失，以验证模型的收敛性和泛化能力。同时，与传统的深度 Q 网络模型进行对比，评估深度 Q 脉冲网络在 CartPole 任务上的性能优势。



(a)训练过程中的总奖励对比



(b)训练过程中的损失对比

图 5.1: 训练过程中的总奖励和损失

为确保公平比较，我们对比了深度 Q 脉冲网络模型和传统深度 Q 网络模型在训练过程中的总奖励和损失表现，使用了相同的训练参数和环境设置：隐藏层大小为 256，脉冲模拟步数为 16。

从图中可以看出，深度 Q 脉冲网络在训练后期的总奖励显著高于传统深度 Q 网络，表现出更优的学习能力。然而，其损失函数相对较高，这可能归因于脉冲神经网络的离散性特征和对训练噪声的敏感性。相比之下，传统深度 Q 网络的训练过程更加稳定，损失函数较低，但在后期表现略逊于深度 Q 脉冲网络模型。总体来看，深度 Q 脉冲网络表现出较大的训练波动性，可能需要更长的训练时间或进一步优化参数。

因此，为了评估模型的泛化能力，我们进一步测试了深度 Q 脉冲网络在不同参数设置下的性能表现。测试中，我们选择了以下参数组合进行对比：

- 隐藏层大小：256 和 512
- 脉冲模拟步数：8 和 16

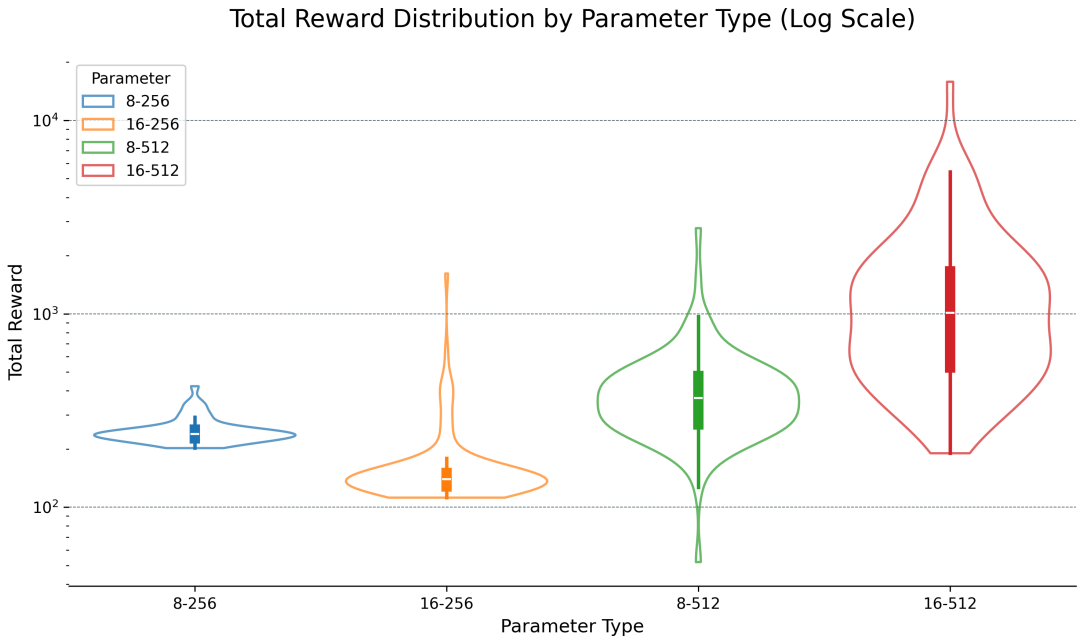
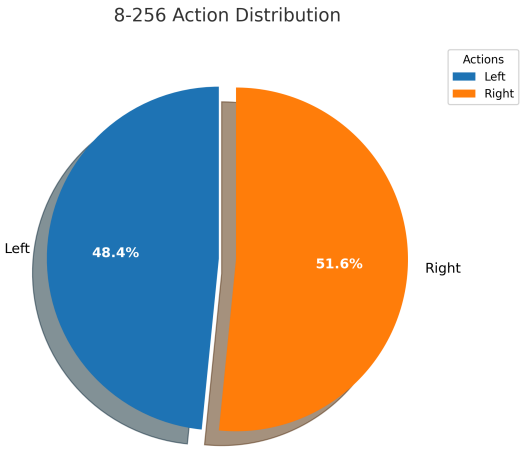


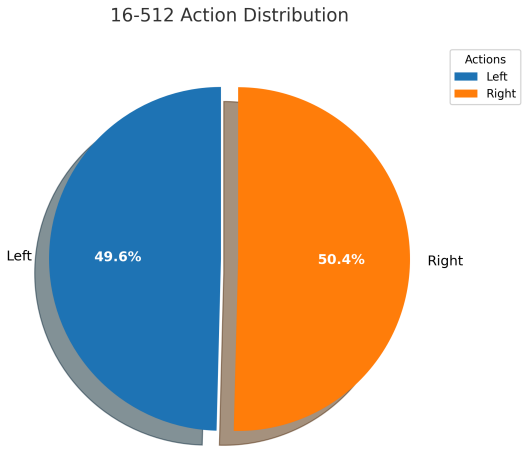
图 5.2: 不同参数设置下的性能对比

从测试结果可以看出，不同参数设置对深度 Q 脉冲网络模型的性能产生了显著影响。对于初始参数组合（隐藏层大小为 256，脉冲模拟步数为 16），模型的总奖励值处于较低水平，且损失函数较高。这可能是由于隐藏层规模较小且脉冲模拟步数较大，导致模型存在过拟合风险。当保持隐藏层大小不变，将脉冲模拟步数减少至 8 时，模型的总奖励值显著提升，且表现更加稳定，展现出更好的泛化能力。这一结果也验证了我们之前的假设。此外，随着隐藏层大小和脉冲模拟步数的进一步增加，模型的性能持续提升，表现出更强的学习能力和泛化能力。当隐藏层大小为 512，脉冲模拟步数为 16 时，模型的总奖励值达到最高，损失函数降至最低，展现了最优的性能表现。当模型参数量足够大时，适当增加脉冲模拟步数能够显著提升模型的性能，为设计更高效的深度 Q 脉冲网络提供了重要参考。

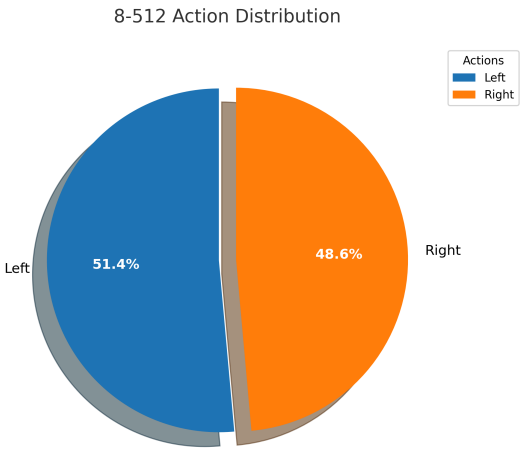
最后，我们分析了不同参数设置下，模型在测试过程中的动作选择分布，以评估模型的探索性和偏向性。



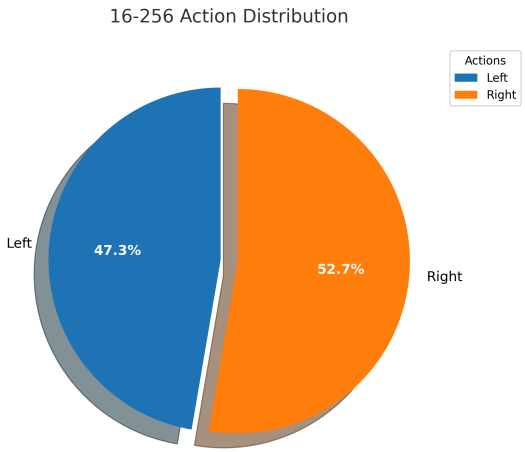
(a)隐藏层大小为 256，脉冲模拟步数为 8



(b)隐藏层大小为 512，脉冲模拟步数为 16



(c)隐藏层大小为 512，脉冲模拟步数为 8



(d)隐藏层大小为 256，脉冲模拟步数为 16

图 5.3: 不同参数设置下的动作选择分布

从图中可以看出，隐藏层大小为 512、脉冲模拟步数为 16 的模型在测试过程中动作选择分布较为均匀，表现出良好的探索能力。而其他参数设置下的模型在测试过程中动作选择均存在一定偏向，可能是由于模型参数量不足或训练不充分所致。这种偏向性可能限制了模型对环境的充分探索，影响其泛化性能。

实验结果表明，深度 Q 脉冲网络在 CartPole 任务中表现出一定的性能优势，但其波动性和参数敏感性较高，可能需要更细致的参数调优。未来研究可以聚焦于优化脉冲网络的训练策略（如调整脉冲编码方式或损失函数），并结合高效的计算框架进一步提升模型性能。

6 实验结论

通过本实验，我们成功构建并训练了深度 Q 脉冲网络，并将其应用于经典强化学习任务 CartPole 的求解，验证了脉冲神经网络与强化学习结合的可行性与潜力。

深度 Q 脉冲网络结合了强化学习的决策能力和脉冲神经网络的计算特性，实现了对复杂连续状态空间的处理。结果表明，深度 Q 脉冲网络在 CartPole 任务中表现出一定的性能优势，展示了脉冲神经网络在强化学习任务中的应用潜力。然而，实验也揭示了一些值得进一步研究的问题。脉冲神经网络的性能对神经元模型参数（如 LIF 模型参数）高度敏感，训练过程对参数优化提出了更高要求。同时，由于脉冲信号的离散化特性，经验回放在样本采样与训练效果上需要进一步适配。此外，虽然脉冲神经网络具有理论上的稀疏性优势，但在实际模拟中，时间步长的增加带来了额外的计算开销，这提示未来研究中应探索更加高效的实现方式，如网络结构的优化或硬件加速技术。

本实验还从生物智能的角度为强化学习任务的研究提供了新的启发。通过结合强化学习的决策能力与脉冲神经网络的计算机制，深度 Q 脉冲网络模拟了生物体行为学习的过程，为类脑计算研究提供了理论依据。这种结合展现了脉冲神经网络在类脑智能中的潜力，不仅在强化学习任务中表现优异，还可能为多智能体协作和动态环境适应等复杂任务的求解提供新的方向。

未来的研究可以进一步扩展深度 Q 脉冲网络的应用场景，例如结合类脑芯片进行硬件实现，探索其在嵌入式设备和边缘计算中的潜力。同时，还可将其应用于更复杂的强化学习任务，如多智能体系统中的协同行为学习或动态环境中的策略优化。综上所述，本实验证明了脉冲神经网络与强化学习结合的优势，为研究高能效类脑计算和生物启发人工智能提供了重要的参考价值。