基于深度强化学习 (DQN) 的贪吃蛇 AI

算法实现与结果分析

2351050 杨瑞晨

同济大学 软件学院

2025 年 6 月 2 日

目录

项目简介

项目目标

- 利用深度强化学习方法训练一个能够自主玩贪吃蛇游戏的智能体。
- 探索 DQN 算法在像素级输入环境中的学习能力。

项目背景与意义

- 验证 DQN 算法有效性: 通过训练 AI 验证其在处理复杂策略游戏中效果。
- 验证从原始像素输入进行端到端学习的可行性。
- 了解深度强化学习在实时决策和复杂环境中的应用潜力。

主要方法概述

- 采用**深度 Q 网络 (Deep Q-Network, DQN)** 算法。
- 结合**经验回放 (Experience Replay)** 和**目标网络 (Target Network)** 机制优化训练过程。

Q-Learning 简介

Q-Learning

Q-Learning 是一种无模型、离策略的强化学习算法，旨在学习最优动作价值函数 $Q^*(s, a)$，表示在状态 s 下执行动作 a 的预期未来总回报。

核心公式

更新规则：

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

优点与缺点

- 优点：简单易实现，适用于离散状态空间。
- 缺点：在高维状态空间中难以存储和更新 Q 值表，学习效率低下，对超参数 $(\alpha, \gamma, \epsilon)$ 的选择比较敏感

DQN 核心思想

DQN 的创新点

1. **神经网络逼近 Q 函数：** DQN 引入了深度神经网络 (DCNN) 来逼近 Q 值函数 $Q(s, a; \theta)$。
   - 输入: 状态 s (游戏画面像素)。
   - 输出: 每个可能动作的 Q 值。
   - 参数 θ : 网络的权重和偏置, 通过梯度下降更新。

   $\Longrightarrow$ 解决了高维状态空间问题 实现泛化 (从学习到的状态中推断出未见过但相似的状态的 Q 值)

2. **经验回放：** 存储经验 (s, a, r, s') 到记忆库缓冲区。训练时从中随机采样小批量的经验来更新 Q-Network
   随机采样训练，打破数据相关性, 提高数据利用率。

3. **目标网络：** 使用独立的、周期性更新的目标网络 $Q'(s, a; \theta')$ 计算目标 Q 值: $y = r + \gamma \max_{a'} Q'(s', a'; \theta')$。以稳定训练过程、减少 Q 值估计的震荡、提高训练稳定性。

DQN 算法实现

神经网络结构

- 输入层：接收 num_last_frames 帧堆叠的游戏画面
- 卷积层：两层卷积用于提取空间特征
- 全连接层：将特征图展平后连接到全连接层，进行更高级的抽象
- 输出层：输出每个动作的 Q 值

```
model = Sequential()

# Convolutions.
model.add(Conv2D(
    16,
    kernel_size=(3, 3),
    strides=(1, 1),
    data_format='channels_first',
    input_shape=(num_last_frames, ) + env.
    observation_shape
))
model.add(Activation('relu'))
model.add(Conv2D(
    32,
    kernel_size=(3, 3),
    strides=(1, 1),
    data_format='channels_first'
))
model.add(Activation('relu'))

# Dense layers.
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(env.num_actions))

model.summary()
model.compile(optimizer=RMSprop(),
loss=MeanSquaredError()) # Use explicit classes
```

探索与利用

- **ϵ-greedy 策略：** 以概率 ϵ 选择随机动作，否则选择当前 Q 值最高的动作。ϵ 随训练轮数从 1.0 衰减到 0.1

```
if np.random.random() < exploration_rate:
    # Explore: take a random action.
    action: Any = np.random.randint(env.
    num_actions)
else:
    # Exploit: take the best known action
    for this state.
    q: Any = self.model.predict(state)
    action: intp = np.argmax(q[0])
```

经验回放与目标 Q 值计算

- **经验回放：** 存储每个时间步的经验 (s, a, r, s') 到经验池，随机采样小批量进行训练

```
experience_item: list = [state, action,
reward, state_next, game_over]
self.memory.remember(*experience_item)
```

- **目标 Q 值计算：** 使用目标网络 Q' 计算目标 Q 值 $y = r + \gamma \max_{a'} Q'(s', a')$

```
# Predict future state-action values.
X: NDArray = np.concatenate([states, states_next], axis=0)
y: Any = model.predict(X)
Q_next: Any = np.max(y[batch_size:], axis=1).repeat(self.num_actions).
reshape((batch_size, self.num_actions))

delta: NDArray[float64] = np.zeros((batch_size, self.num_actions))
delta[np.arange(batch_size), actions] = 1

targets: Any = (1 - delta) * y[:batch_size] + delta * (rewards +
discount_factor * (1 - episode_ends) * Q_next)
```

实验设置

环境参数

- **地图：** 10x10-blank.json
- **初始蛇长：** 3
- **奖励：**
  - 吃果子: $+1 \times$ 蛇长
  - 每步: 0
  - 死亡: -1

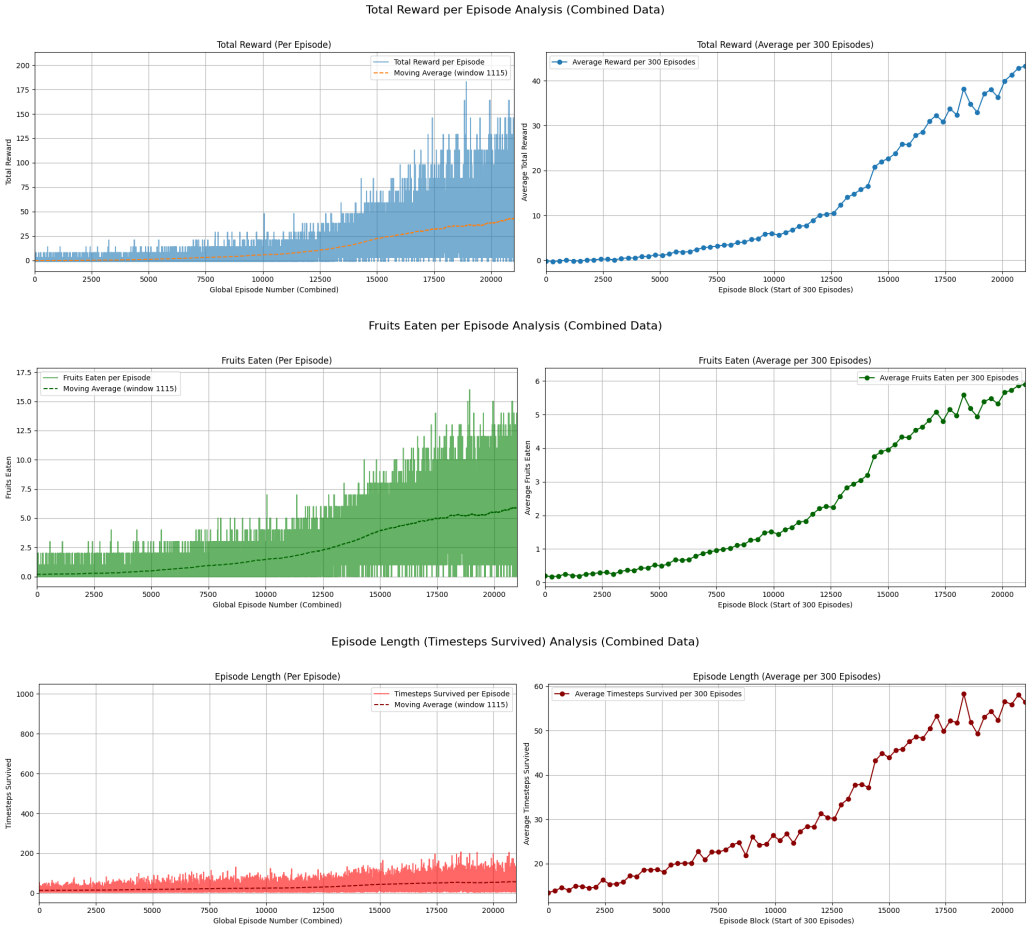
```
"rewards": {
  "timestep": 0,      // 每存活一步的奖励
  "ate_fruit": 1,     // 吃到果子的基础奖励因子
  "died": -1          // 死亡的惩罚
}
```

训练参数

- **优化器：** RMSprop (Keras 默认参数)
- **损失函数：** Mean Squared Error (MSE)
- **学习率：** RMSprop 默认学习率 $lr = 0.001$
- **折扣因子 (γ)：** 0.95 (鼓励远期回报)
- **经验回放缓冲区大小 (memory_size)：** -1 (无限制，实际受限于物理内存)
- **批处理大小 (batch_size)：** 64 (每次从经验池中采样 64 个样本进行训练)

```
agent = DeepQNetworkAgent(
    model=model,
    memory_size=-1,
    num_last_frames=model.input_shape[1]
)
agent.train(
    env,
    batch_size=64,
    num_episodes=parsed_args.num_episodes,
    checkpoint_freq=parsed_args.num_episodes // 10,
    discount_factor=0.95,
    start_episode=parsed_args.initial_episode
)
```

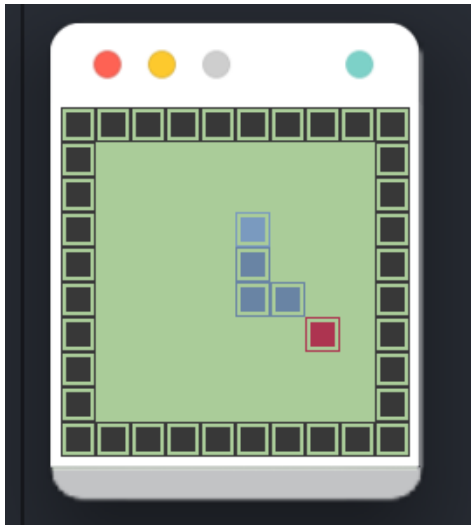
训练曲线分析



- 观察到随着训练轮数增加，平均总奖励、吃掉的果子数、存活步数均呈现 **明显上升趋势**。
- 训练后期，各项增长趋于 **平缓**，但仍存在波动。
- 表明智能体不仅学会了吃果子，也学会了更好地规避死亡，智能体逐渐掌握了游戏策略，但仍有改进空间。

训练结果展示

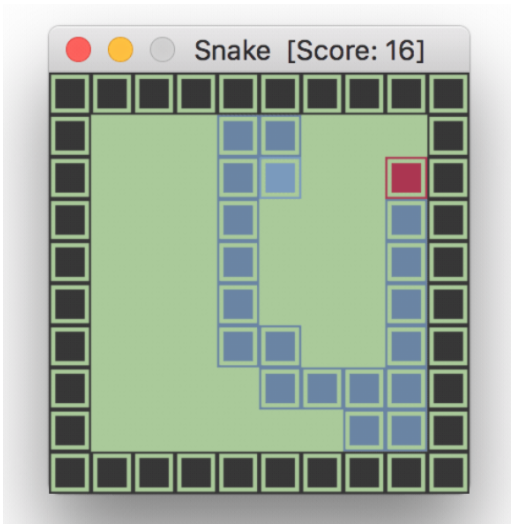
训练初期-Epi 9000



PLAY

- 智能体训练效果一般，表现出随机行为。
- 平均得分较低，吃果子数量有限。
- 有时会转圈“摆烂”，以获得更大的生存时间。

训练后期-Epi 21000



PLAY

- 智能体在这一阶段表现出更高的策略水平。
- 能够吃掉更多的果子，生存时间显著延长。
- 但当得分到达一定程度后，智能体出现不稳定行为。

结论与展望

主要结论

- 成功实现了基于 DQN 算法的贪吃蛇 AI，并验证了其学习能力。
- 通过分析关键指标，证明了 DQN 在像素级输入下的有效性。
- 智能体从随机行为逐步学习到更优策略。

未来工作展望

- **算法改进：**尝试 Double DQN, Dueling DQN 等。
- **网络结构优化：**探索更有效的 CNN 架构。
- **奖励函数探索：**设计更精细的奖励函数。
- **更复杂环境：**在包含更多动态障碍物的关卡中测试。

THANKS!