机器学习化学大作业：
基于强化学习的 Breakout-v4 游戏模型评估

黄柏喻[1], 杨雪巍[2]

[1] 数学科学学院，数据科学与大数据技术，
2100010860@stu.pku.edu.cn
[2] 数学科学学院，数据科学与大数据技术，
2100010864@stu.pku.edu.cn

2025.1

摘要

本项目基于 OpenAI Gym 开发，使用强化学习算法在经典 Atari 游戏环境 Breakout-v4 中进行实验。该游戏要求智能体控制挡板反弹球击碎砖块，得分通过击碎砖块实现。实验首先使用 Q-learning 作为基线模型，通过离散化游戏状态并利用 Q 表更新策略来选择动作。接着，采用 DQN 模型引入卷积神经网络（CNN），以处理高维原始像素图像并近似 Q 值函数。进一步地，我们在 DQN 模型的基础上使用预训练的 ResNet-18 模型提取图像特征，以加速学习过程。实验结果显示，虽然 DQN 模型优于基线模型，且预训练模型表现稍好，但由于硬件限制和训练时间不足，模型未能充分学习环境中的复杂性。此外，模型在训练过程中学习到了一些误区，特别是对于奖励机制和惩罚设计的反应，影响了其学习效果。本项目的实验展示了传统 Q-learning、DQN 以及结合预训练模型的算法在强化学习任务中的表现差异，并为进一步优化强化学习模型提供了方向。我们的项目可通过Github仓库获取。

目录

1 介绍

1.1 OpenAI Gym

我们的项目基于OpenAI Gym[1] 开发，它是一个用于开发和比较强化学习算法的开源工具包，由 OpenAI 开发。其中提供了多样化的预定义环境，包括经典控制问题、Atari 游戏和机器人模拟等。其优势在于提供了简单一致的 API，支持与主流深度学习框架集成。此外，它可以直接通过 pip 安装，并根据需要扩展特定环境的依赖，例如有针对性地下载我们本项目所需的 Atari 游戏部分。

1.2 Breakout-v4

我们选择的环境是 **Breakout-v4**，它是 Gym 提供的一个经典 Atari 游戏环境，基于 Atari 2600 游戏机上的 Breakout 游戏[1]。玩家控制挡板反弹球，击碎屏幕顶部的砖块以得分。游戏内截图如图1所示。游戏目标是控制一个挡板反弹球，击碎屏幕顶部的砖块。每击碎一个砖块会得分，当所有砖块被击碎时，游戏通关；若挡板没能反弹球，本回合结束。若干回合后计算总得分即为最终得分。

图 1: 游戏截图，上方所示为“砖块”，下方为挡板，中间红色为小球

[1] Atari 2600

1.3 强化学习要素

从游戏描述中我们可以看出，模型只能操控挡板的移动，动作是离散的，包括：

0 ： 无操作

1 ： 开火

2 ： 向左移动挡板

3 ： 向右移动挡板

在上方我们看到有一个奇怪的动作“开火”，这其实是源于 Atari 游戏的统一定义，在ALE 相关文档中可以发现，这是一套统一的动作空间，可以自动适应到所有 Atari 游戏中，而不必为每个游戏重新设计一组动作列表。我们在运行程序时输出动作序列也可以发现模型并不会调用动作 1，主要使用动作 2、3 来控制挡板位置。

而游戏的环境即为原始像素的图像，形状为 $(210, 160, 3)$，是三通道的 RGB 图像。

游戏的奖励依照 Atari 游戏的设计，就是模型的得分。每当模型成功接球并将其反弹，每打碎一个砖块模型便可获得一分。此外，出于一些特定的目的，例如惩罚模型学习到的“混分”行为，我们引入了一些惩罚措施，这将在后续展开讨论。

1.4 预处理

为了简化问题，通常对 `Breakout-v4` 的观察空间进行预处理：

- **灰度化**：将 RGB 图像转换为单通道灰度图像。
- **裁剪**：移除无关区域（如分数栏）。
- **缩放**：将图像缩小到更小的尺寸（如 84×84）。
- **帧堆叠**：将连续 4 帧图像堆叠在一起，以捕捉时间信息。

预处理中的一些具体数值，比如缩放图像的尺寸，主要是参考了 DQN 原文 [2] 的设计，并且也符合直观：游戏本身只需要关注球和挡板以及砖块，不需要颜色信息、分数栏等无用内容，且前后几帧内的图像变化相对小，可以堆叠起来，不影响判断球的位置。因此这样的简化是合理且非常有必要的。

2 强化学习模型

2.1 基线模型

本实验的 Baseline 模型基于经典的 Q-learning 算法 [3]，旨在通过与 Atari Breakout 环境的交互，学习一个 Q 表（Q-table）来指导智能体的动作选择。Q-learning 是一种无模型的强化学习算法，通过不断更新状态-动作对的 Q 值，逐步优化策略。Q 表是一个三维数组，形状为 $84 \times 84 \times n$，其中 84×84 由游戏图像 reshape 所得，n 为动作空间的大小，初始值通过随机采样生成。智能体采用 ϵ-greedy 策略选择动作，即以概率 ϵ 随机选择动作，以概率 $1-\epsilon$ 选择当前状态下 Q 值最大的动作。Q 值的更新基于 Bellman 方程：

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

其中，α 为学习率，γ 为折扣因子，r 为即时奖励，s' 为下一状态。初始探索率 $\epsilon = 1.0$，每轮训练后按

$$\epsilon \leftarrow max(\epsilon_{min}, \epsilon \times \epsilon_{decay})$$

衰减，以逐步减少随机探索。

训练过程中，模型共进行 1,000,000 轮训练，每轮最多执行 1,000 步。每轮训练结束后，记录总奖励并保存表现最佳的 Q 表。为了评估模型性能，每 1,000 轮计算一次平均奖励，并输出当前训练进度。训练过程中，若某轮的总奖励超过历史最佳值，则保存当前 Q 表为 `best_q_table.npy`，以便后续测试和分析。这确保了模型在训练过程中能够保留最佳表现，从而为后续实验提供了可靠的基础。

2.2 DQN

在 2.1的基础上，我们引入了深度学习算法，设计了基于 CNN 的 DQN 模型 [2]。DQN 模型的优点在于通过使用神经网络替代 Q 表来近似 Q 值函数，从而能够处理高维状态空间（如原始像素图像）。模型以经过预处理后的游戏画面作为输入，通过由三层卷积层和一层全连接层组成的卷积神经网络从输入中提取特征，最终输出每个动作的 Q 值。借助于 CNN 在图像识别上的良好性能，模型能够自动学习游戏画面中的空间特征，从而更好地理解游戏状态。

为了提升训练的稳定性和样本利用率，DQN 模型引入了经验回放机制。经验包括状态、动作、奖励、下一状态和是否结束等智能体与环境交互的信息，由一个回放缓冲区存储。每次训练时，模型从回放缓冲区中随机抽取大小为 $BatchSize$ 的经验，用于更新模型参数。这种机制打破了数据之间的相关性，使得训练过程更加稳定。

此外，为了进一步稳定训练过程，DQN 模型还使用了目标网络（Target Network）。目标网络与主网络结构相同，但参数更新频率较低。每 10 轮训练后，主网络的参数会被复制到目标网络。目标网络用于计算目标 Q 值，减少了训练过程中的波动。

DQN 模型还引入了原地不动的惩罚机制，这是因为在实际模型训练过程中，有时会观察到智能体会采取静止在角落的错误策略，这可能是由环境发球随机性有限带来的偶然因素所导致的。为了避免智能体学习到静止在角落的错误动作，当它被观察到当前状态与下一状态相同时，会受到惩罚，从而鼓励智能体采取更有意义的动作。

2.3 预训练基座模型

在 2.2的基础上，我们直接使用预训练的图像模型基座作为图像特征提取器，用来表征当前状态所见到的图像具有的特征。我们选择的基座模型是 ResNet-18[4]，这主要是考虑到 ResNet 本身性能较强，其次它可以轻松通过 torch API 取得，以及相对较简单、小型 GPU 上有能力微调。我们的项目中首先去掉最终的分类层，然后只取 ResNet 的最后一层全连接模块可以微调，其余参数均冻结。由于 ResNet 本身训练在 ImageNet 数据集上，输入的图像是三通道的，我们需要将灰度图重新升为三通道以供模型使用。此外，我们还根据 ImageNet 的均值和方差作了归一化。

在预训练模型的基础上，我们叠加了一层带 ReLU 激活函数的全连接层，用于学习，最后用一层全连接层输出各个动作的概率。其余部分与章节2.2中的基本设置均相同。

2.4 实现细节

2.4.1 开发环境

由于Bohrium平台不支持长达数天的强化学习训练[2]，所以我们使用自己的电脑完成训练。本地设备具有 NVIDIA GeForce RTX 3060 Laptop GPU，具有一定的并行加速能力。2.1中的模型较为简单，可以直接使用 CPU 完成训练，而2.2和2.3中的模型则使用 GPU 加速。尽管如此，受限于显存大小和 GPU 本身的性能，训练依然相对较慢，迭代 24 小时约能执行 20000 次，这其实是数量非常小的。此外由于显存较小，本地的运行需要通过一个批处理文件来完成自动化地在显存占用过多时中止训练并保存权重。

2.4.2 软件版本

我们使用的 Python 版本为 3.9，Gym 版本为 0.26.2，torch 版本为 2.1.2+cu118。其余使用到的软件包括 Pandas, NumPy 等等，可参考 Github 项目中的 requirements.txt。除了 Gym 版本以外，我们并没有使用具有各个软件中具有明显版本特性的功能，可以预见对不同版本的模块也有一定的支持能力。

2.4.3 超参数设计

模型中的大部分超参数参考了 [2] 的设计，例如学习率 $lr = 1e-4, \gamma = 0.99, min\ \epsilon = 0.02$。部分超参数例如 $BatchSize = 64$ 在试验中有相应的调整，这则是结合了本地硬件的实际情况，选取了适应本地设备的大小。

3 实验结果对比

3.1 数据图表

我们整理训练过程的数据展示如下。其中各项指标均使用最后 3000 次迭代内的数值，避免在 $\epsilon - greedy$ 策略中的贪心概率较大时出现的随机异常值。这确实发生过，DQN 在迭代数百次后突然获得一个 12 分的高分，具体解释将在下一节中尝试给出。

[2]指直接挂在后台一直跑的形式。在长时间不操作后网页容易自动断开连接，并且校园网稳定性有待提升……

此外我们同样引用了 DQN 所在论文中的数据，包括人类平均水准和充分训练的 DQN 模型的表现。表1展示了各个模型之间的对比数据。

模型	平均得分	最高得分	方差
基线	0.75	**11**	1.50
DQN	0.988	8	1.637
预训练 + DQN	**1.37**	8	1.67
人类平均	31	-	-
DQN(论文)	168	225	-

表 1: 各个模型得分对比

在此基础上，我们将迭代过程中的得分可视化为下方图2, 3, 4的散点图，和前面展示的数值一样，只采用了最后 3000 次的数据。

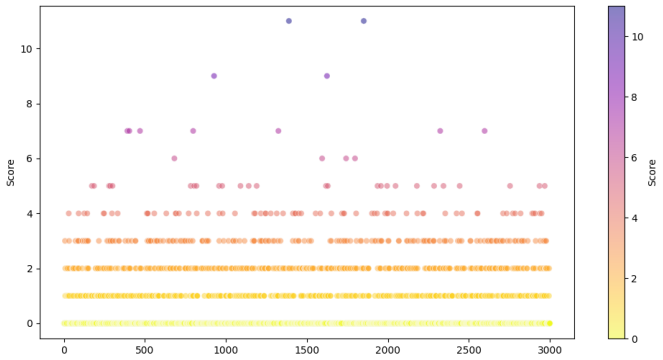

图 2: Q learning 得分散点图

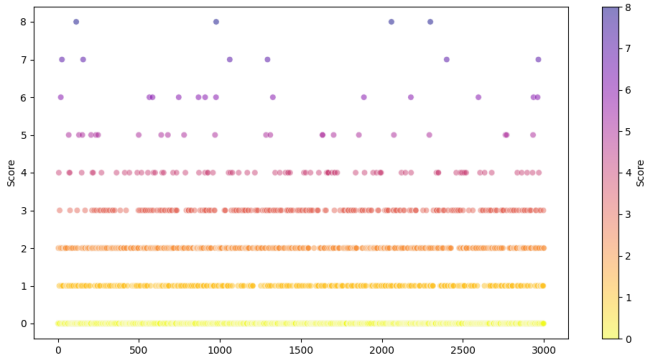

图 3: DQN 得分散点图

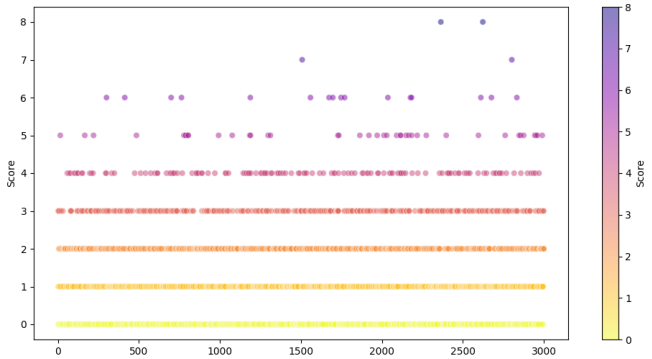

图 4: 预训练 +DQN 得分散点图

从中可以看出，尽管基线模型取得了最高分 11，但从散点图中可以看出，很可能只是随机地获得的，并非充分学习的结果，平均分反而是最低的，其大部分得分集中在 5 分以下，这也导致了方差较小，但这实际上是稳定的差的表现；DQN 的平均分有所提升，预训练 DQN 的平均分则比 DQN 稍高，且是三个模型中最高的，这符合我们的预期。

3.2 人类实际观感

我们通过 `render_mode='human'` 观察模型的实际操作表现。观测到基线模型的表现无疑是最不理想的，而 DQN 模型表现稍好，预训练模型比

DQN 模型观感略好但并不显著。归根结底还是因为学习不够充分，有待继续学习。

此外，我们认为模型学习到了错误的经验。由于游戏本身发球的随机性不够强，容易把球发到靠近右下角的位置，并且经过物理碰撞后，会恰好反弹到稍微偏左一点点的地方，如果挡板能一直固定在右下角，将会获得至少 2 分。我们曾在非常早期的时候观测到一个总得分 10 分，但那其实都是因为模型的好几次发球都获得了这样的保底。

为此我们在前面提到，引入了对于始终使用相同动作的惩罚，以期待能避免模型持续停在原地，但模型从中学会了插入序号为 0 的原地不动动作来打断连续的右移，用“点刹”回避了持续卡在右下角的惩罚。总而言之，通过观看模型实际表现视频，我们发现模型容易学习到误区，特别是在训练轮数远远不足以让它充分学习环境的情况下，这种现象尤为明显。

4 分析解读

4.1 游戏本身内容分析

游戏本身可以被认为是纯视觉的，只需根据当前图像输出四分类结果，即决策动作。针对图像的深度学习算法正是近十年来深度学习的最为成功的代表，用 CNN 等计算机视觉算法解决强化学习问题无疑是非常有前景的。

但同时我们需要考虑到的是强化学习本身的特性。与深度学习不同，深度学习里一张图的计算可以通过并行快速缩减，但强化学习模型需要非常长时间的训练，而且具有明显的循环依赖性，不容易使用硬件加速。

此外，正如3.2中所提到的，对惩罚项的设计也有值得改进的地方。对于惩罚过高的设计，模型会学习到宁可胡乱移动也比持续选择一个可能得分的动作要好，这反而影响了正常的学习；对于惩罚不够的情形，模型就会陷入到系统本身存在的一些小特性中，利用这些特性来获得局部最优解，但是这并非我们希望看到的。

4.2 算法比较

我们采用的三种算法，包含了最基本的 Q Learning，以及引入深度学习的 DQN，并在此基础上尝试使用预训练模型的知识来加速学习。DQN 和预训练模型理论上应该有更好的表现，但受限于硬件和时间，模型的学习程度十分有限，尚不能充分体现出这几个算法之间的优劣。

另外，我们的模型设计上仍有改进之处。一是卷积网络本身优越在于能提取丰富的图像特征，但该游戏的图像前后变化相对较小，有可能经过卷积、池化等操作后不同状态下提取到的特征反而相差不大。同时 CNN 等模型预训练的任务是图像分类，对于前后两帧高度相似的游戏场景，模型可能输出相同的类别，这也导致了我们观测到的持续使用一个动作的现象。如果引入更多特征工程，比如对小球部分作边缘检测后将边缘信息交给深度学习模型处理，有可能助于模型判断小球位置并移动挡板到合适位置。

二是我们的模型仍有可能过于简单不足以完成这一游戏，**Breakout** 游戏本身已经称得上相对复杂，人类在这一游戏上的表现也不见得有多高分，特别是对于不熟悉游戏操作的人而言。未来我们可以尝试更复杂的模型、换用当下流行的 VLM 等预训练大模型，以获得更好的游戏表现。

最后从数据本身我们可以知道，模型并没有完全训练完成——这需要非常长的时间，在 DQN 原文图表中展示的结果是训练了 100 轮，每轮有 50000 次运行，而我们总共尚且迭代了约 5-6 万次，这显然是远远不够的。DQN 作者还提出 "*This suggests that, despite lacking any theoretical convergence guarantees, our method is able to train large neural networks using a reinforcement learning signal and stochastic gradient descent in a stable manner.*" [2] 这也暗示着复现算法潜在的收敛速度不够快、收敛性不好等现实问题。更何况深度学习本身有 "炼丹" 的美名，训练过程的随机性有目共睹，想在一两次实验之内就成功训练是有困难的。

5 结论

我们的项目通过应用强化学习算法对 **Breakout-v4** 进行性能优化与模型评估。尽管训练有待继续充分开展，但实验结果表明，DQN 和策略梯度方法在提高游戏得分和智能体表现方面具有一定的效果，尤其是在面对相对复杂的游戏界面作为环境时。通过对比不同算法和超参数设置，我们发现，合适的奖励、惩罚设计与训练超参数设计是优化强化学习模型性能的关键因素，但无论如何要想获得足够强大的强化学习模型，足够的训练时间是必不可少的。

总的来说，我们通过项目学习了如何设计基于强化学习的游戏智能体，获得了宝贵的经验，并通过学习课程和阅读相关文献，了解了这些设计背后的理论依据，未来可以进一步探索更复杂的游戏环境以及更加高效的算法，以期在其他应用领域实现更广泛的应用。同时，随着算法与硬件的发展，期

望强化学习在实际场景中的应用能够突破现有的瓶颈，发挥更大的潜力。

组内分工

黄柏喻：负责前期规划、模型设计、代码编写、报告撰写工作。
杨雪巍：负责资料收集整理、报告撰写工作

参考文献

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[3] Christopher J. C. H. Watkins and Peter Dayan. Q-learning, may 1992.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.