强化学习算法实践：题目2

邱靖越 M202421061

December 24, 2024

1 作业内容

1.1 实验 1

实现 Dyna-Q 算法，并通过调节参数找到算法可提升的性能极限。

实验要求:

- 完成 Dyna-Q 代码，从 $n = 0$ 开始（即纯 model-free 方法），尝试调试不同的参数，记录不同 n 值的实验结果。
- 记录算法的收敛时间和所消耗的样本量，给出收敛时间随 n 的变化趋势。
- 寻找经验性估计：n 的取值达到某个范围后，算法收敛所消耗的样本量不再有明显的下降，并记录样本消耗量随 n 的变化趋势。

1.2 实验 2

用神经网络来预测环境 Model， 实现简单的 Model-based 算法。

要求 1：算法调试，调节算法的参数，寻找能达到的最好效果的参数组合。
要求 2： 改进算法

- 改进 1：尝试改进 Model 的学习流程，强化对稀疏/奖励变化相关的数据的学习。
- 改进 2：对策略的学习过程做额外的约束

分别尝试两个改进，重新调节该实验的参数组合，最优的参数和对应的性能是否发生变化？若有变化，发生了什么变化。

2 实现过程

2.1 实验 1

本节将介绍实验 1 的实现过程，包括 QAgent 类和 DynaModel 类的实现，以及重要的训练过程。

2.1.1 QAgent 类的实现

QAgent 类实现了强化学习中的 Q-learning 算法，用于 Q 表的更新并选择动作。具体包括以下关键功能:

- **Q 表的初始化**：Q 表是一个多维数组，其形状由状态空间的维度（8，8，2）和动作空间（4）的维度共同决定，用于存储每个状态-动作对的Q值。初始的Q值全部设置为0。

- **动作选择策略**：使用 ϵ-greedy 策略选择动作，若多个动作的Q值相同，则随机选择其中一个。
- **Q 值更新**：Q 值更新遵循 Q-learning 的更新规则：

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

其中，α 是学习率，γ 是折扣因子，r 是即时奖励。如果当前状态是终止状态（done 为 True），则无需考虑未来的 Q 值。

2.1.2 DynaModel 类的实现

DynaModel 类实现了 Dyna-Q 算法中的模型部分，用于存储和模拟环境的动态行为。

- **状态转移存储**：利用字典结构存储状态转移信息 $(s,a) \rightarrow (r,s')$：
- **随机采样机制**：从已记录的状态中随机采样一个状态（sample_state）。利用采样状态选择动作。
- **模拟环境预测**：根据存储的模型预测给定状态和动作的奖励以及转移后的新状态（predict）。若当前状态和动作尚未被记录，则返回默认值。

Dyna-Q算法如表 Algorithm 1 所示。

Algorithm 1 Dyna-Q

Require: Initialized $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A$. A given reward function R and terminal function D.

$s = \text{env.reset}()$
while True **do**
  while not done **do**
    $a = \epsilon\text{-greedy}(s,Q)$
    $s', r, \text{done} = \text{env.step}(a)$
    Update Q: $Q(s,a) \leftarrow \alpha\left[r + (1-\text{done}) \cdot \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    Update $Model$: $Model(s,a) \leftarrow s'$
    $s = s'$
    if done **then**
      $s = \text{env.reset}()$
    end if
  end while
  for n times **do**
    $s_m \leftarrow$ random previously observed state
    $a_m \leftarrow$ random action previously taken in s_m
    $s'_m \leftarrow Model(s_m, a_m)$
    $r_m \leftarrow R(s_m, a_m)$
    Update Q: $Q(s_m,a_m) \leftarrow \alpha\left[r_m + (1-\text{done}) \cdot \gamma \max_{a'} Q(s'_m,a'_m) - Q(s_m,a_m)\right]$
  end for
end while

2.1.3 实验设置

- $start\_planning = 10$，从第 10 次更新开始，模型利用已经收集的环境交互数据进行模拟更新。在此之前，只使用真实环境中的交互数据更新 Q 表。
- $h = 2$，单个状态下最多模拟的步数。模拟更新中，从一个随机采样的状态开始最多执行 10 步模拟。
- $n = [0, 1, 3, 5, 10, 20, 50, 100, 200, 300, 400, 500]$，表示每次模拟更新中采样的状态数量。在实验中，通过调整 n 的值来观察其对算法收敛速度和样本效率的影响。

- **终止条件**：连续 20 updates 结果大于 80 且最大差异不超过5，训练终止。

其他相关参数设置如Table 1所示。

Table 1: Experiment 1 setup

Parameter	value
epsilon	0.2
gamma	0.99
alpha	0.2

2.2 实验 2

实验 2 的整体训练流程与实验 1 相似，分别在三种实验条件：神经网络模型、改进的神经网络模型、加入 clip 的神经网络模型下进行较优参数的寻找和结果分析。n 的实验范围为 (0, 1000)，m 的实验范围为（1，1000），h 的实验范围为（1，10），sp 的实验范围为（10，120）。实验 2 的算法如表 Algorithm 2 所示。

Algorithm 2 Q-learning with Neutral Network

Require: Initialized $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A$. A given reward function R and terminal function D.

$s = \text{env.reset}()$
for iter $= 1$ to T **do**
  while not done **do**
    $a = \epsilon\text{-greedy}(s, Q)$
    $s', r = \text{env.step}(a)$
    Update Q: $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + (1 - \text{done}) \cdot \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s = s'$
    if done **then**
      $s = \text{env.reset}()$
    end if
  end while
  for m times **do**
    $Model$.train_transition()
  end for
  if iter $>$ start_planning **then**
    for n times **do**
      $s_m \leftarrow$ random previously observed state
      for h times **do**
        $a_m = \epsilon\text{-greedy}(s_m, Q)$
        $s'_m \leftarrow Model(s_m, a_m)$
        $r_m \leftarrow R(s_m, a_m, s'_m)$
        done $\leftarrow D(s_m, a_m, s'_m)$
        Update Q: $Q(s_m,a_m) \leftarrow Q(s_m,a_m) + (1 - \text{done}) \cdot \alpha\left[r_m + \gamma \max_{a'} Q(s'_m, a'_m) - Q(s_m,a_m)\right]$
        if done **then**
          break
        end if
      end for
    end for
  end if
end for

3 复现方式

在主文件夹下运行 python main.py –model=dyna，model 参数可以选择 dyna、network 和 optnetwork。通过改变 n、m、h、sp 参数来进行实验。

4 实验效果

4.1 实验 1

本节为实验 1 的结果分析。

观察 n 从 0 到 500 的变化，当 $n = 0, 10, 50, 100, 200, 500$，模型分数随训练样本量的变化情况如 Fig. 1 所示。结果分析如下：

1. $n = 0$ 的情况：当没有模拟更新时，算法完全依赖真实环境的交互数据进行学习。此时，模型的收敛较慢，训练过程起伏较大，这是因为纯 Q-learning 方法未能充分利用历史经验数据，导致样本利用率低。

2. n 的增加带来的提升：从图 (b)-(f) 可以看出，随着 n 的增加，模型的收敛速度显著提高。例如，当 $n = 10$ 时，模型在大约 2000 步后达到了初步的性能提升，而 $n = 200$ 则在不到 1000 步内迅速收敛至接近最优性能。此外，训练过程中曲线的波动幅度显著减小，表明模型在更新过程中的稳定性也有所提升。

3. 收敛分析：如图 (e)-(f) 所示，当 n 达到 200 或更高时，模型的收敛性能趋于稳定，进一步增大 n 对收敛时间和最终性能的提升均变得有限。

Fig. 2 描绘了训练时间和更新次数随 n 的变化趋势，进一步验证了 n 的影响规律：

1. 训练时间的变化（蓝色曲线）：随着 n 的增加，训练时间显著减少，尤其是在 $n = 10$ 至 $n = 200$ 的范围内，训练时间从约 16 秒快速下降至 8 秒，减少了 50%。在 $n \geq 200$ 时，训练时间的变化趋于平稳，进一步增大 n 的收益有限。

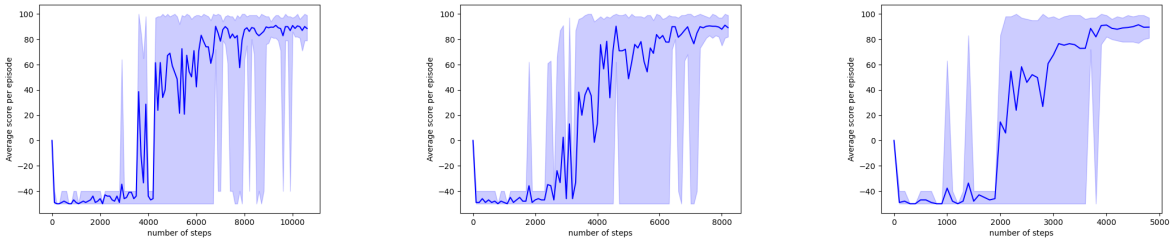
2. 采样次数的变化（绿色曲线）：随着 n 的增加，采样次数逐步减少。例如，从 $n = 10$ 到 $n = 200$，采样次数从约 9000 次下降至 4000 次，减少了 55%。这表明模拟采样显著降低了对真实环境交互的依赖。当 $n = 500$ 时，采样次数趋于平稳，仅比 $n = 200$ 略微减少。

综合两张图可以得出经验性结论：当 $n = 200$ 时，算法在收敛性能与计算效率之间达到了较好的平衡点。此时，模型在不到 1000 步内完成了快速收敛，训练时间减少至约 8 秒，同时采样次数减少至 4000 次左右。相比之下，进一步增大 n 的优化收益已趋于饱和，无法显著提升性能。

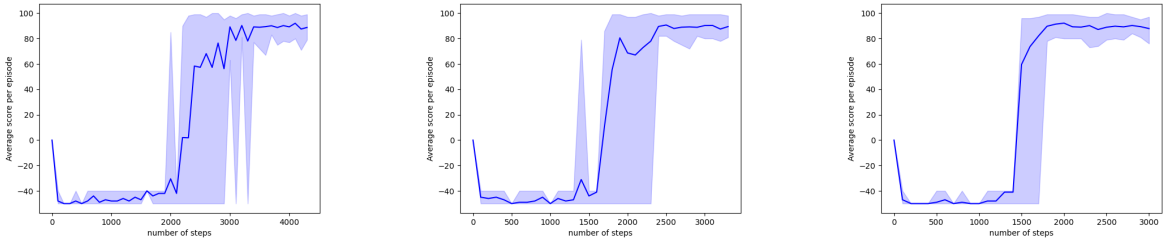
4.2 实验 2

本节为实验 2 的结果分析。

在神经网络的实验过程中，本文尝试了多种参数组合，但初始模型均出现了难以收敛的问题，其结果如 Fig. 3 所示。通过观察训练过程并打印梯度，发现模型中存在梯度爆炸的问题。这一问题导致模型在训练分数呈现较大波动，无法稳定提升。为解决梯度爆炸问题，本文引入了梯度裁剪技术，将梯度的一范数限制在 1 内，同时将学习率设置为 $1e{-}5$。改进后的模型在缓解梯度爆炸问题后表现明显改善，其结果如 Fig. 4 所示。改进模型的训练过程更为稳定，最终收敛至较高的分数。在调试过程中，我进一步探索了参数对模型性能的影响，重点分析了 m（训练步数）、n（回放次数）、h（模拟深度）和 sp（采样间隔）等参数。实验发现，这些参数的设置对训练效率和最终性能有显著影响：



(a) $n = 0$ (b) $n = 10$ (c) $n = 50$



(d) $n = 100$ (e) $n = 200$ (f) $n = 500$

Figure 1: Performance of different n values.

- m: 如果 m 设置过大，模型会过度依赖环境模型的预测，从而导致偏差累积；反之，m 设置过小会导致环境模型的学习不足，影响 Q 值更新的效果。
- n: 较大的 n 能够提高基于模型的规划次数，但会显著延长训练时间并增加收敛时间。
- h: 当 h 超过 3 时，模型的学习效果出现大幅下降，这可能是由于模型预测的误差随着模拟步数增加而累积，导致学习目标偏离。
- sp: 采样间隔 sp 如果太小，会导致模型在训练时过于依赖少量重复样本，限制探索；如果 sp 太大，模型的更新频率则不足，影响环境动态变化的学习，或者还没开始模拟就已提前收敛。

通过多次实验调试，本文确定了一组较优的参数配置，如 Table 2 所示。

Table 2: Experiment 2 setup

Parameter	value
sp	90
n	20
h	2
m	10

实验改进：

1. **强化对稀疏/奖励变化相关的数据的学习**：通过修改模型的训练流程，引入对稀疏奖励变化数据的强化学习方法。具体改进如下：在数据存储阶段，对奖励变化显著的数据进行标记，并在训练阶段增加对这些数据的采样频率。经过实验调试，使用改进后的训练流程，我找到的最佳参数组合为 $n = 20$, $m = 10$, $h = 2$, $sp = 50$。在训练了 11600 轮次后，模型相比于改进前所需的 12400 轮次，训练轮次减少了约 6.5%，但训练时间增加了 1 秒。这是由于额外的采样步骤增加了训练过程中对稀疏奖励数据的计算负担，但从最终模型性能来看，整体效果更优。训练后的性能变化趋势如 Fig. 5 所示。
2. **对策略的学习过程做额外的约束**：在 Q 值更新公式中增加了额外的约束，即对 $Q(s, a)$ 的值进行裁剪：

$$Q(s, a) = \text{clip}(Q(s, a), -100, 100)$$

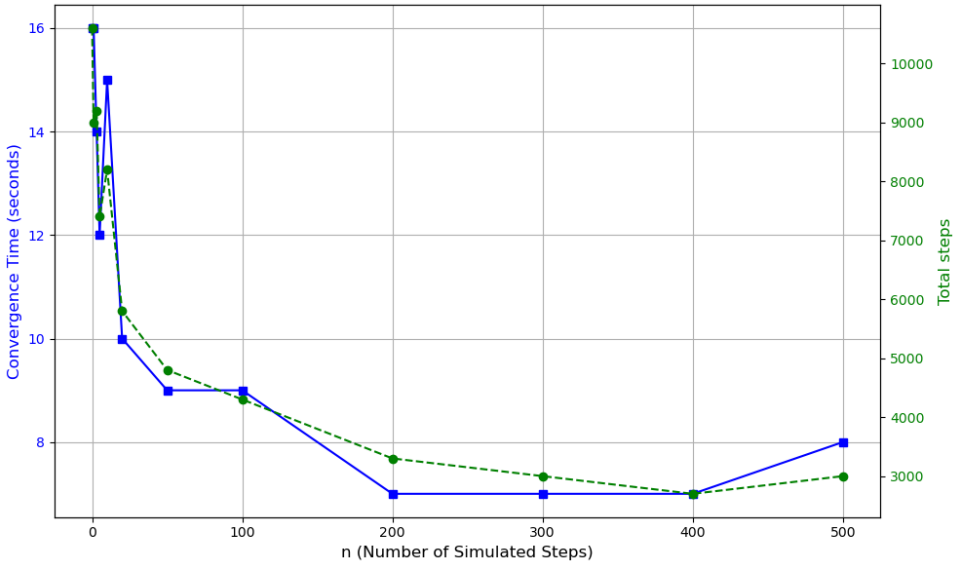


Figure 2: Convergence analysis: time and update times vs. n

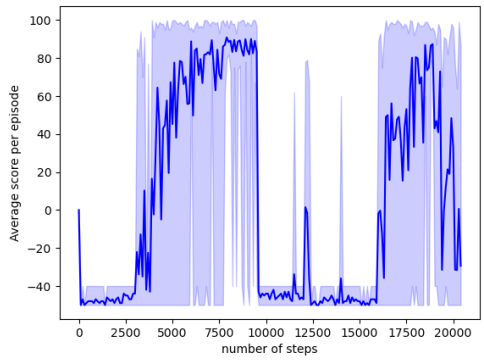


Figure 3: Performance of original model ($n = 20, m = 10, h = 2, sp = 90$)

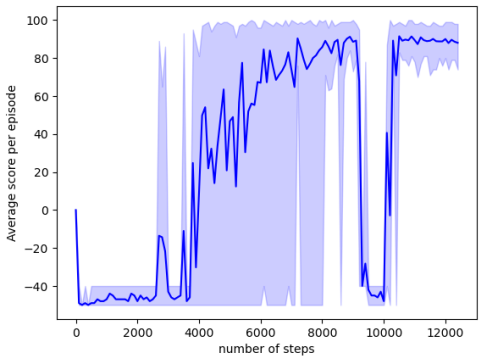


Figure 4: Performance of upgraded model ($n = 20, m = 10, h = 2, sp = 90$)

这一改进旨在限制 Q 值的极端波动，从而避免模型在学习过程中由于异常值导致的不稳定现象。实验表明，这一改进对实验的整体分数影响较小，模型的分数大多维持在 $-50 \sim 100$ 之间。虽然模型性能未显著提升，但约束的引入在部分训练阶段有效抑制了分数的大幅波动。

5 小结

本节为实验总结。

本次实验通过对 Dyna-Q 算法以及基于神经网络的 Model-based 强化学习方法的研究与改进，主要围绕稀疏奖励学习问题和模型性能优化进行了探索性实验。对于实验 1，调整了 Dyna-Q 算法中的关键参数 n（模拟更新次数），通过观察其对收敛速度和样本效率的影响，验证了模拟更新在提高算法效率和性能上的重要作用。实验结果表明，当 n 值从 0 增加到 200 时，算法的收敛速度显著提高，同时所需的真实交互数据量显著减少，展示了模拟更新对样本效率的提升作用。然而，进一步增大 n（如 $n \geq 200$）时，其优化收益逐渐减小，表明模拟更新存在收益递减效应。通过实验，我得到了一个经验性结论：在本实验环境下，$n = 200$ 是性能与效率之间的较优平衡点。

在实验 2 中，针对神经网络模型在稀疏奖励环境下学习效率低的问题，应用了强化对奖励变化显著

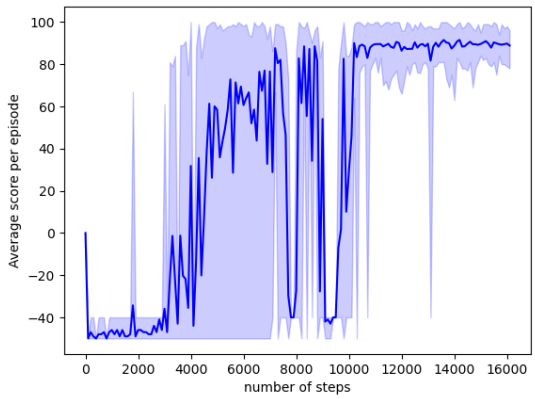


Figure 5: Optimal method

数据的学习方法，并通过额外的采样和训练步骤有效提高了模型的性能。实验表明，这一改进能够减少训练轮次约 6.5%，但引入了额外的计算成本。此外，针对模型在训练过程中梯度爆炸的问题，引入梯度裁剪技术以及对 Q 值的裁剪约束，有效缓解了训练过程中分数波动大的问题，提升了训练过程的稳定性。尽管这些改进未能显著提升最终分数，但它们通过抑制极端值对模型的影响，也许在更复杂的场景中会呈现出更优的效果。

在简单场景中，例如状态空间较小、动态变化较弱且奖励信号明确的问题，本次实验中的 Dyna-Q 算法及其模拟更新方法表现出显著优势。这类场景的特点是问题结构相对简单，模型可以通过有限的真实环境交互和少量的模拟更新快速收敛。利用 Dyna-Q 会更高效。在复杂场景中，例如状态空间大、环境动态复杂或奖励稀疏的任务，也许更适合用神经网络模型。基于神经网络的 Model-based 方法能够更好地对高维状态和非线性动态进行拟合，从而提供更精确的环境预测。