MuZero

1. 概述

一直以来，构建具有规划能力的智能体 (Agents With Planning Capabilities) 都是人工智能领域的主要挑战之一。在诸如国际象棋和围棋等具有挑战性的领域，基于树的规划方法 (Tree-based Search) 取得了巨大的成功，然而与这些领域不同的是，在现实世界中缺少完善的模拟器并且环境控制的动力学往往是复杂与未知的。在本篇文章中，作者将介绍 MuZero 算法，该算法**在不知道环境状态转移规则 (Environment’s Dynamics)** 的情况下，通过将**基于树的搜索**与**学习模型 (learned model)** 相结合，在一系列挑战性的输入为复杂视觉图像的领域中实现了超人的性能。 构造一个抽象的 MDP (Markov Decision Process) 模型，在这个 MDP 模型上，去预测与规划 (planning) 直接相关的未来信息（策略、价值函数以及奖励），并在此基础上预测数据进行规划。当在57个不同的Atari游戏（用于测试人工智能技术的经典视频游戏环境，并且基于模型的规划方法在此类环境上一直表现不佳）上进行评估时，达到了新的 SOTA。当在围棋、国际象棋和 shogi 游戏上进行评估时，在未获取任何游戏规则知识的情况下，MuZero 实现了与明确游戏规则的 AlphaZero 算法一样的超人性能。

2. 研究背景与相关工作

一般的规划算法都依赖于对环境动态转移规则，而在现实应用场景中智能体通常无法完整获取此类信息。而基于模型的强化学习（Model-based RL）通过学习一个环境动态模型来解决这个问题，根据学习到的环境动态模型做进一步的规划。经典的model-based RL 算法有：Dyna, MVE, PILCO 等，详细的整理可见 Awesome Model-Based Reinforcement Learning。而无模型强化学习 (Model-free RL) 恰好相反，无模型方法不需要先学习环境的模型，而是通过不断试错直接学习从状态到动作的映射关系。经典的算法有 Deep Q-Network (DQN)、Proximal Policy Optimization (PPO) 等。

环境模型 (Model of The Environment) 是强化学习系统的重要元素之一，它被用来模仿环境行为，或者更普遍的说，让人们可以推断环境的行为。传统上，该模型由马尔可夫决策过程 (MDP) 表示，给定一个状态和行动，环境模型可以预测由此产生的下一个状态和下一个奖励。一旦构建了模型，就可以直接应用 MDP 规划算法，例如值迭代或蒙特卡洛树搜索 (MCTS) ，来计算 MDP 的最优值或最优策略。

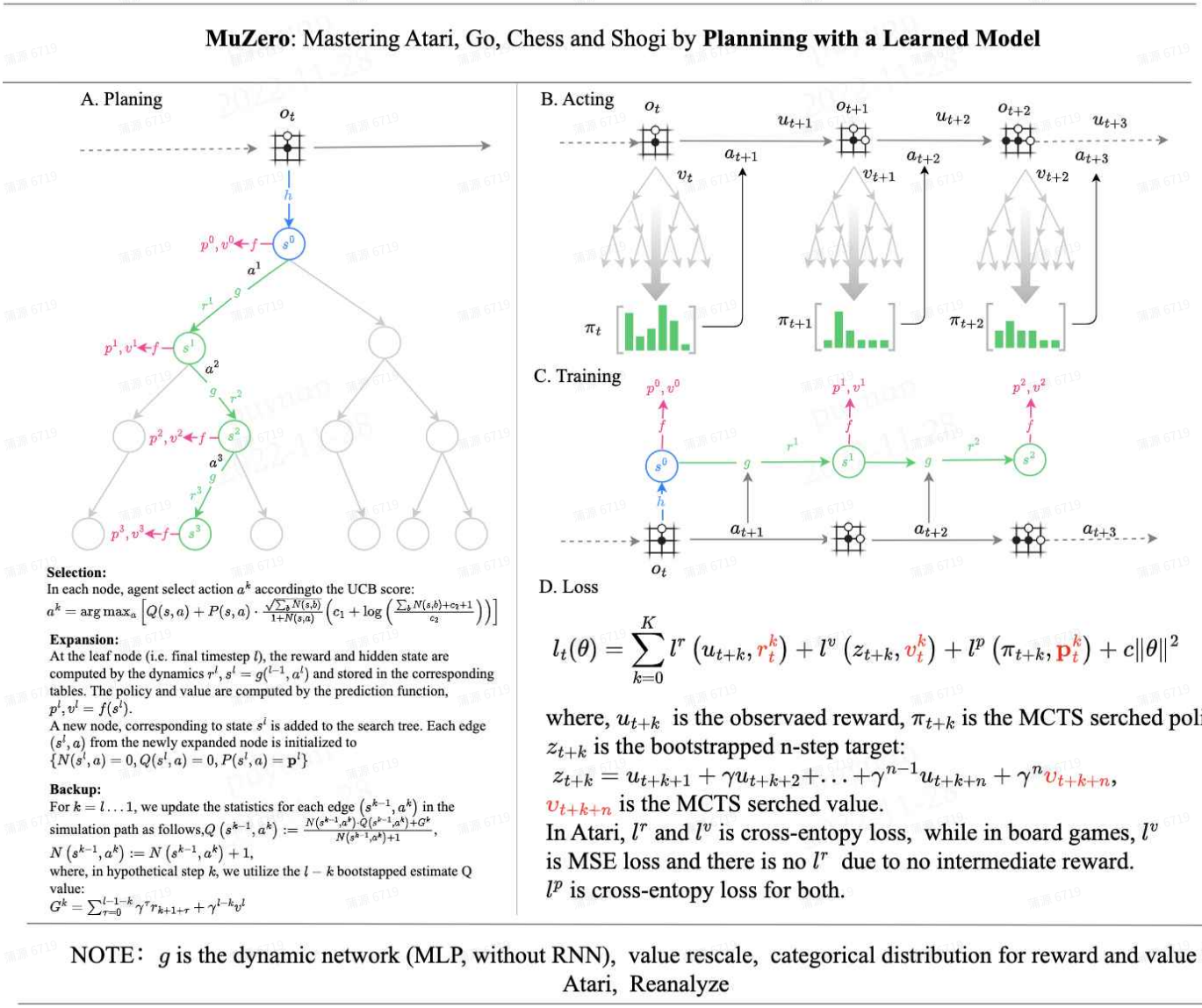
在大型或部分观察的环境中，算法必须首先构建模型将要预测的状态表示形式 (State Representation)。表示学习 (Representation Learning)、模型学习 (Model Learning) 和规划 (Planning) 之间的这种三方分离可能存在问题，因为智能体无法为了有效规划的目的优化其表示或模型，因此，建模错误的现象可能会在规划期间加重。

在 MuZero 之前，Model-based RL 在例如Atari类的视频游戏中表现远不如直接学习最优策略函数/价值函数并进行决策的 Model-free RL 。但是 Model-free RL 在需要较为精准复杂的前瞻的领域做的不好，比如说象棋和围棋。

MuZero 名字中的 **'Mu'**，其实是做梦的意思。MuZero 算法的主要思想，是在不知道环境动态转移规则的情况下，在梦中/想象的空间中进行搜索，即构造一个抽象的MDP模型，在这个 MDP 模型上，去预测与 Planning 直接相关的未来数据（策略、价值函数以及奖励），并在此基础上预测数据进行规划。简单来说，就是在虚拟状态空间中**先学出一个环境模型**，然后再**基于这个所学到的环境模型**，在**无法与真实环境交互**过多的情况下**进行规划。**

3. MuZero 算法

3.1 概览图



(图1：MuZero 算法概览图。**A**：MuZero agent 利用 MCTS 进行规划: representation network h 将连续 t 帧观测 (observation) $\{o_1, ..., o_t\}$ 编码为 latent state s^0 ；dynamics network g 预测在 s^{k-1} 下执行 action a^k 会转移到的 latent state s^k 和 reward r^k ；prediction network f 在 s^{k-1} 下输出 policy p^k (选每个 action 的概率)和 value v^k 。**B**：MuZero 如何与环境交互：对于图中 o_t 而言，使用蒙特卡洛树搜索得到一个策略，根据此策略进行采样，选出可执行动作，动作被选择的概率与 MCTS 根结点的每个动作的 visit count 成正比。在执行动作之后，环境产生真实奖励 u_{t+1} ，得到

下一时刻的观测 o_{t+1}，就这样不断往下与环境交互，在 episode 结束时，轨迹数据被存储到回放缓冲区（replay buffer）中。**C**：MuZero 在训练时需要用 dynamics network 展开 K 步，通过预测每一步的 reward, value, policy 来学习模型。**D**: 具体的损失函数定义。符号的含义参考【算法概览图符号表.pdf】。)

3.2 MuZero 模型

AlphaZero 依靠 MCTS 做决策，MCTS 是在真实的环境模拟器中进行的，搜索的过程中需要策略网络和价值网络的辅助。

MuZero 同样依靠 MCTS 做决策， 是在学习到的抽象状态空间中进行的，搜索的过程中需要如下3个网络的辅助：

- 表征网络 (Representation Network)
  - $s^0 = h_\theta(o_1, ..., o_t)$。将根节点（时刻t）的局面表示为隐藏状态 (或称为抽象状态)，后面的预测都在隐藏空间中进行搜索
- 动力学网络 (或称为机制网络) (Dynamics Network)
  - $r^k, s^k = g_\theta(s^{k-1}, a^k)$。也就是模拟环境机制，给定状态和所选动作，dynamics network 给出转移到的 state 和相应的 reward
- 预测网络 (Prediction Network)
  - $\mathbf{p}^k, v^k = f_\theta(s^k)$。给定隐藏状态，预测此隐藏状态的策略和价值，和 AlphaZero 类似。
  - $v \rightarrow z$，为数值，用 $l = (v - z)^2$
  - $p \rightarrow \pi$，为概率分布，用交叉熵 $-\pi^T log(p)$

其中 **prediction function** 即为 AlphaZero 中用到的策略网络和价值网络，也就是说，MuZero 在 AlphaZero 的基础上额外增加了 **representation network** 和 **dynamics network**。而增加的这两个网络即是为了让 MuZero 在学习到的抽象状态空间中进行搜索。

在下面作者会依次探讨如下问题：

- MuZero 所说的“在**不知道规则**的情况下，在想象的状态空间中进行搜索”的具体含义，即为什么要增加 **representation network** 和 **dynamics network?**
- MuZero 如何与环境进行交互和收集数据?
- MuZero 如何训练模型?

3.3 学习抽象状态空间

3.3.1 棋类环境的规则

- 所谓 MuZero 在不知道规则的情况下，进行学习和搜索，是因为 MuZero 在深度神经网络和MCTS运行的过程中，没有使用游戏的如下规则
  - 规则1：终止状态和最终奖励的判断。
  - 规则2：给出状态，动作，如何转移到下一个状态的规则（即环境动力学规则，Dynamics）。
    - 例如围棋游戏中的提子（在围棋中被对方围住的棋子会被提出棋盘）。
  - 规则3：给定一个状态，由谁落子，即需要采取动作的 agent是哪个玩家；给定一个状态，允许的落子，即合法动作集合。

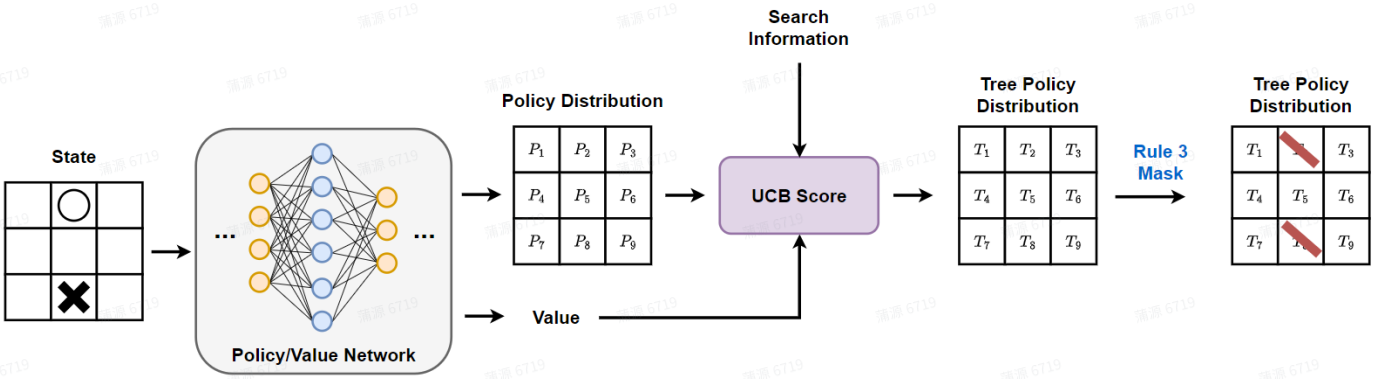
3.3.2 AlphaZero 是如何使用规则的：以井字棋为例

- 假设当前局面为 s_0 ，也即 MCTS 搜索的根节点为 s_0 ，在MCTS每一步搜索的每一次模拟时，会把当前局面输入给策略网络 π_θ ，得到9个概率值，然后通过如下公式得到最终9个动作的 score（Q 是之前 MCTS 模拟估计的动作价值）：

$$score(a) = Q(s,a) + P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}[c_1 + \log(\frac{\sum_b N(s,b) + c_2 + 1}{c_2})]$$

- 其中 N 为访问次数、 Q 为平均价值、 p 为策略、 s 为状态，最后部分，分子是该节点总的访问次数的开方，分母是子节点的访问次数，这样对于访问比较少的子节点的值就会比较大。

但是此时棋盘上本来就有两个棋子，所以需要屏蔽这两个动作及其概率值后再对当前 MCTS 得到的策略进行argmax，这里AlphaZero 就用到了上面的规则3。



(图2：AlphaZero 利用规则进行决策。)

- 选择action后，棋盘进入新的状态， $(s_t, a_t) \rightarrow s_{t+1}$ ，这里就用到了规则2
- 在搜索的时候如果遇到游戏结束，到达最终状态，此时不会用神经网络来判断此节点的价值，而是直接用输赢 $v = \pm 1$ （实际也融合了价值网络的输出），所以这里用到了规则1

3.3.3 MuZero 在抽象状态空间的学习

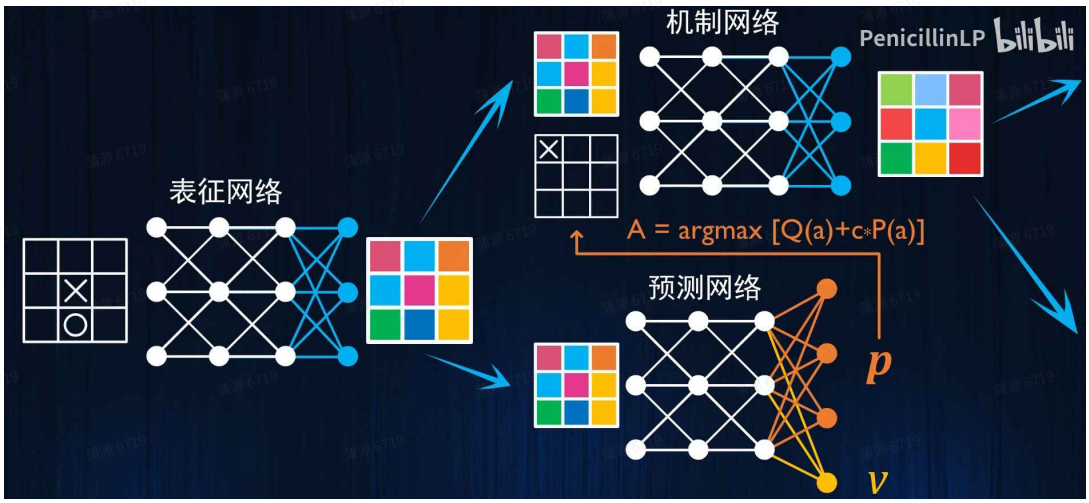
- 用 representation network: $s^0 = h_\theta(o_1, ..., o_t)$ ，将真实的棋盘状态转换为一个抽象的隐状态（Hidden State） s_0

- 以 s_0 作为搜索树的根节点，将这个隐藏状态而不是真实局面输入到 prediction network: $\mathbf{p}^k, v^k = f_\theta(s^k)$，但是在根节点处 MuZero 是知道规则3的，也就是哪些地方不能下子（已经有子），先 mask 掉非法动作概率，再根据 ucb_score 选择动作。
- MuZero 不知道规规则2，但是使用机制网络 dynamics network: $r^k, s^k = g_\theta(s^{k-1}, a^k)$ 进行模拟，进入下一个隐藏状态。
- 隐藏状态空间中不存在最终状态，且隐藏状态下任何动作都是允许的，所以 MuZero 不知道规则1。
- 根节点往后的搜索中，因为隐藏状态和真实状态无关，所以不知道哪些动作合法或者游戏是否结束，可以一直搜索下去，此时 MuZero 是不知道规则3的。

3.4 数据收集

和 AlphaZero 类似，MuZero 也是通过从 MCTS 得到的 visit count 的分布进行采样得到实际与环境交互的动作。

只是在 MCTS 的时候，MuZero 首先需要将当前状态通过 representation network 映射到隐藏状态空间，然后利用dynamic网络与预测网络进行搜索，这个映射和搜索的过程由下图表示：



(图3：MuZero 在隐藏空间进行搜索示意图。)

假设此时MuZero的三个模型（表征网络，动力学网络，预测网络）已经训练好，初始节点 s_0 已经由原始棋盘状态 通过表征网络生成， $s_0 = h_\theta(o_1, ..., o_t)$。

每条边保存的信息：$N(s,a), P(s,a), Q(s,a), R(s,a), S(s,a)$，MCTS在抽象的状态空间中搜索的过程可以分为二以下3个阶段：

❤️ 1. 选择 (selection)

- UCB，结合价值函数+策略函数，选择score最高的action

$$a^k = \arg\max_a \{Q(s,a) + P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}[c_1 + \log(\frac{\sum_b N(s,b)+c_2+1}{c_2})]\}$$

- 访问次数 N、平均价值 Q、策略 P、奖励 R 和状态转换 S。
- $s^k = S(s^{k-1}, a^k)$, $r^k = R(s^{k-1}, a^k)$ 得到 state transition and reward table

2. 扩展 (expansion)
   - 根据上面选择的action，拓展至新的状态节点，转移模型即为 dynamic function，会给出下一个状态和即时奖励。
   - 选择直到时刻 l 到达叶子节点，得到最终的 r^l, s^l （保存到 $R(s^{l-1}, a^l)$, $S(s^{l-1}, a^l)$ 表里），还有prediction function预测的 p^l, v^l ，其中的 p^l 也会保存到 $R(s^l, a)$ 中作为初始化， $R(s^l, a) = 0$, $S(s^l, a) = 0$
3. 回溯 (backup)
   - 主要要更新一路的 Q 和 N：
     - $$Q(s^{k-1}, a^k) := \frac{N(s^{k-1}, a^k) \times Q(s^{k-1}, a^k) + G^k}{N(s^{k-1}, a^k) + 1},$$
     - $$N(s^{k-1}, a^k) := N(s^{k-1}, a^k) + 1.$$
   - 其中 G^k 为 $l-k$ 步对累计奖励的估计，基于 v^l
     - $$G^k = \sum_{\tau=0}^{l-1-k} \gamma^\tau r_{k+1+\tau} + \gamma^{l-k} v^l$$
   - 由于在一些环境中，奖励是无界的，因此可以将奖励以及价值估计标准化到 [0,1] 区间，然后再与先验知识相结合：
     - $$\bar{Q}(s^{k-1}, a^k) = \frac{Q(s^{k-1}, a^k) - \min\limits_{s,a \in \text{Tree}} Q(s,a)}{\max\limits_{s,a \in \text{Tree}} Q(s,a) - \min\limits_{s,a \in \text{Tree}} Q(s,a)}.$$

在 MCTS 搜索完成后，在根节点 s_0 返回 visit counts 集合 $\{N(s,a)\}$，这些 visit counts 经过归一化后便得到了相对上一次训练有提升的策略 (improved policy) $\mathcal{I}_\pi(a|s) = N(s,a) / \sum_b N(s,b)$。然后从这个分布中采样得到实际与环境交互的动作。

3.5 模型训练

3.5.1 MuZero 模型训练的损失函数

该模型的所有参数都是联合训练的，以准确匹配每一个假设步骤 k 的 policy 、value 和 reward，以及 k 个实际时间步后观察到的相应目标值。但是，与传统的基于模型的强化学习方法不同，MuZero 中的隐藏状态 s_k 没有附加环境状态的语义，它只是整体模型的隐藏状态，其唯一目的是准确预测相关的、未来的信息：policy、value、reward。由于 MuZero 是在抽象 MDP 空间中搜索，为保证抽象 MDP 中的 planning 与真实环境中的 planning 等价，通过确保**价值等价 (Value Equivalence)** 来实现。即从相同的真实状态开始，通过抽象MDP的轨迹的**累积报酬**与真实环境中轨迹的**累积报酬**相匹配。于是对于 MuZero 作者有以下优化目标。

- 目标一：与 AlphaZero 类似，目标改进策略是通过 MCTS 搜索产生的，为了进行策略提升，作者最小化预测策略 p_t^k 和 MCTS得到的搜索策略 π_{t+k} 之间的误差 $l^{\mathrm{p}}(\pi_{t+k}, p_t^k)$。
- 目标二：在AlphaZero 中目标 value 使用完整序列的平均动作价值不同。而在 MuZero 中使用带衰减因子的中间奖励和 MCTS 搜索得到的估计价值来更新目标 value，相当于使用了 $TD(n)$，速度更快、方差更小。
  - n-step bootstrapped 目标 value 定义为：$z_t = u_{t+1} + \gamma\, u_{t+2} + \ldots + \gamma^{n-1} u_{t+n} + \gamma^n \mathrm{v}_{\mathrm{t+n}}$。其中 u_{t+i} 表示观测奖励、v_{t+n} 表示 MCTS 搜索得到的估计价值。然后最小化预测网络的预测价值 z_{t+k} 和 目标 value 的误差：$l^{\mathrm{v}}(z_t, v_t^k)$
- 目标三：最小化预测的奖励和实际观察到的奖励之间的误差：$l^{\mathrm{r}}(u_{t+k}, r_t^k)$。

$$l^{\mathrm{p}}(\pi, p) = \pi^T \log p \tag{1}$$

$$l^{\mathrm{v}}(z, v_t) = \begin{cases} (z-v)^2 \text{ for games} \\ \phi(z)^T \log v \text{ for general MDPs} \end{cases} \tag{2}$$

$$l^{\mathrm{r}}(u, r) = \begin{cases} 0 \text{ for games} \\ \phi(u)^T \log r \text{ for general MDPs} \end{cases} \tag{3}$$

- 综上所述，用于训练 MuZero 的3个网络的总损失函数如下，具体梯度回传路径分析见 3.4.2 的分析:

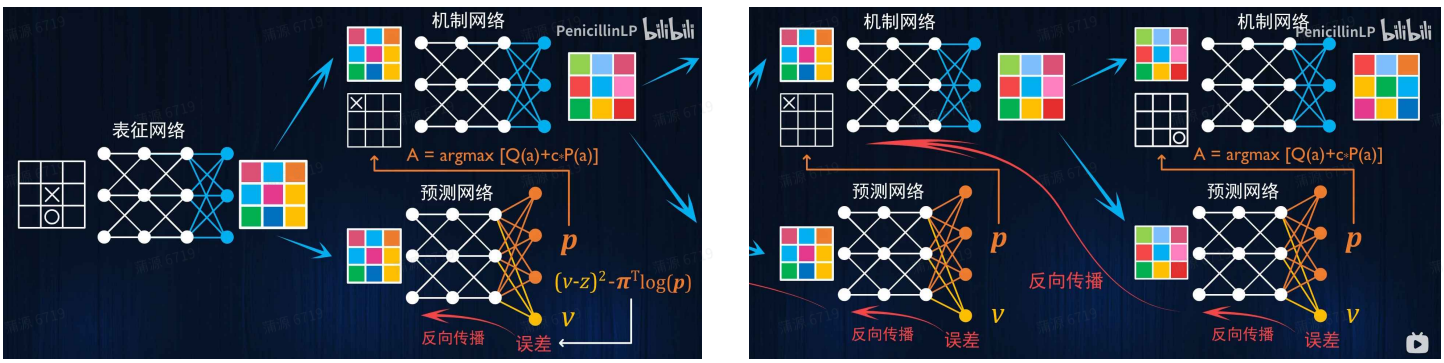
$$l_t(\theta) = \sum_{k=0}^{K} l^{\mathrm{p}}(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^{\mathrm{v}}(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^{\mathrm{r}}(u_{t+k}, r_t^k) + c||\theta||^2$$

3.5.2 重要实践细节

训练所需序列最小长度

根据前面3.3所描述的隐藏空间中的搜索过程，可以看到在时刻 t

- 对于 representation network $s^0 = h_\theta(o_1, \ldots, o_t)$，它的输出 s^0 是 prediction network的输入，所以 representation network 可以直接由 prediction function 反向传播的梯度进行更新。

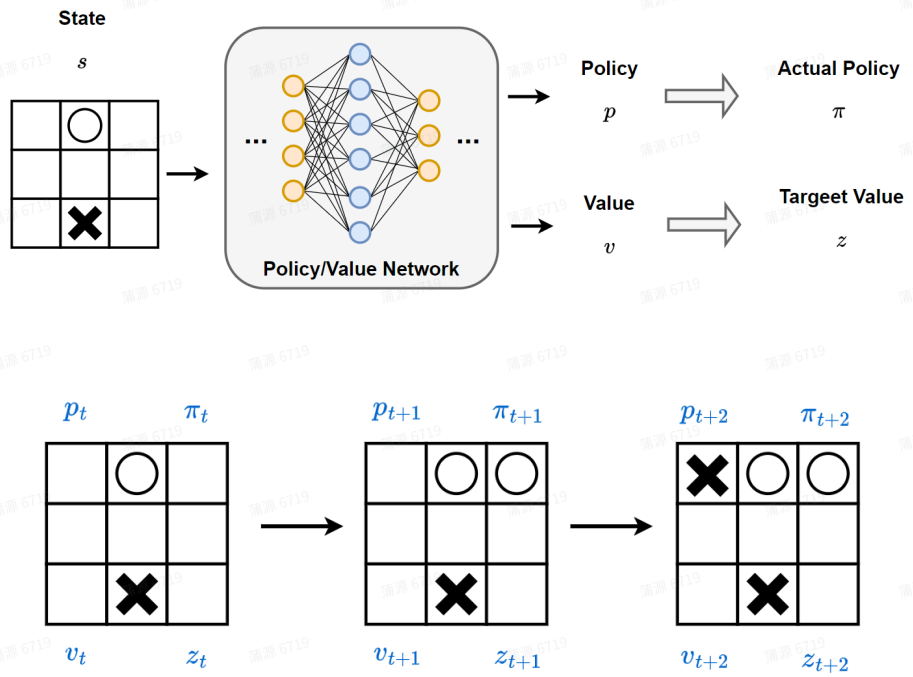


(图4：（左）MuZero 中误差由 prediction network 反向传播到 representation network 示意图。（右）MuZero 中误差由 prediction network 反向传播到 dynamics network示意图。)

- 而对于 dynamics network r^k，$s^k = g_\theta\left(s^{k-1}, a^k\right)$，它的输出 s^k 将会是下一个 prediction network $\mathbf{p}^k$，$v^k = f_\theta\left(s^k\right)$ 的输入，所以下一个 prediction fnetwork 的梯度可以反向传播到 dynamics network
- 所以至少要有两个隐藏状态及其信息，即 $(<o_t, a_t, z_t, \pi_t, s_t, p_t, v_t>, <o_{t+1}, a_{t+1}, z_{t+1}, \pi_{t+1}, s_{t+1}, p_{t+1}, v_{t+1}>)$ 才可以进行训练

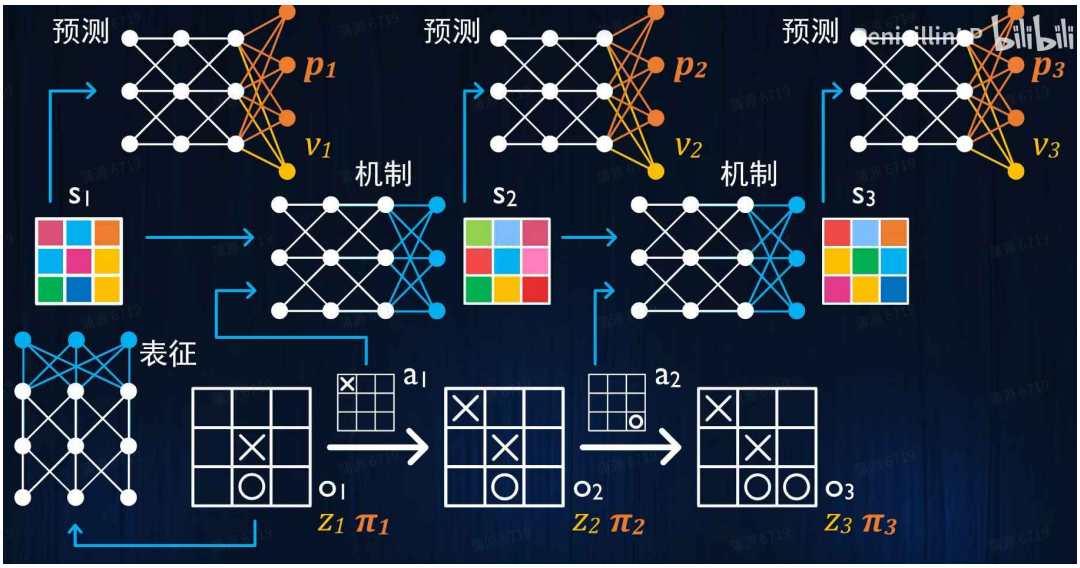
Replay buffer

- AlphaZero 存储的是每一个真实状态 o_t 及其真实策略 π_t，策略网络产生的策略 p_t，价值网络预测的价值 v_t，和真实累计奖励 z_t 信息，即 $(o_t, p_t, \pi_t, v_t, z_t)$，不用存储实际选择的动作



（图5：（上）AlphaZero 中每个状态 s 输入策略价值网络的输出以及优化目标。（下）AlphaZero 的 replay buffer 存储的每个状态 s 的信息）

- 和 AlphaZero 相比， MuZero 还存储了连续状态之间采取的实际动作、在隐藏空间的状态表示，即 $<o_t, a_t, z_t, \pi_t, s_t, p_t, v_t>$，且如上面提及的 MuZero 训练所需序列最小长度是2，而实际 MuZero 用的序列长度是6，即 $<(o_t, a_t, z_t, \pi_t, s_t, p_t, v_t), \ldots, (o_{t+5}, a_{t+5}, z_{t+5}, \pi_{t+5}, s_{t+5}, p_{t+5}, v_{t+5})>$ 作为一条训练数据存储在 replay buffer 中。



（图6： MuZero 的 replay buffer 存储的信息）

4. 实验

4.1 实验设置

4.1.1 实验环境

- 经典棋类游戏作为规划问题的基线：Go，chess，shogi。
- Atari 环境中的57个游戏作为视觉复杂RL领域的基线

4.1.2 网络结构与主要超参数

网络结构

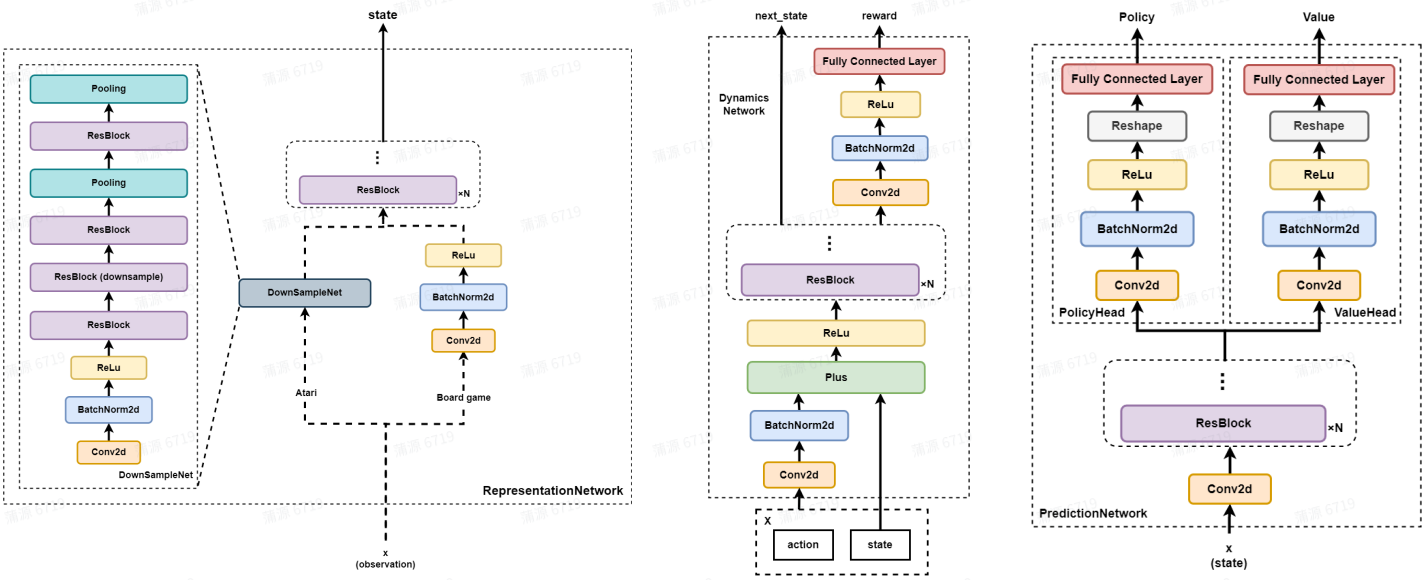
MuZero 用到的三个网络结构如下图1所示。

- representation network 使用了与 AlphaZero 相同的卷积和残差架构，但是有16个 residual block，而不是20个。由于 Atari 游戏环境的观察空间维度较大，所以在 representation network 中采取了 downsample ，从步长为2的卷积序列开始以降低观察空间维度。具体地说，从分辨率96x96和128个平面的输入观察开始（32个历史帧，每个历史帧具有3个颜色通道，与广播到平面的相应32个动作连接），MuZero 向下采样如下：
  - 1个 stride=2 和 128个输出平面的卷积层，输出分辨率48x48。
  - 2个具有128个平面的残差块。
  - 1个 stride=2 和 256个输出平面的卷积层，输出分辨率 24x24。
  - 3个具有256个平面的 residual blocks。
  - stride=2 的平均池，输出分辨率为12x12。
  - 3个具有256个平面的 residual blocks。

- stride=2 的平均池化，输出分辨率为6x6。

  所有操作的 kernel size 为3x3。

- dynamics network 使用与 representation network 相同的架构。
- prediction function 使用与 AlphaZero 相同的架构，一到两个卷积层，保留了分辨率 (preserve the resolution)，但减少了平面的数量，然后是一个输出大小的全连接层。所有的网络都使用 256 个隐藏平面。



(图7： 左：MuZero 的表征网络结构图。中：MuZero 的动力学网络结构图。右：MuZero 的预测网络结构图。注：本文档网络结构图均按照我们 LightZero 实际代码实现，与原论文有细微区别。)

主要超参数

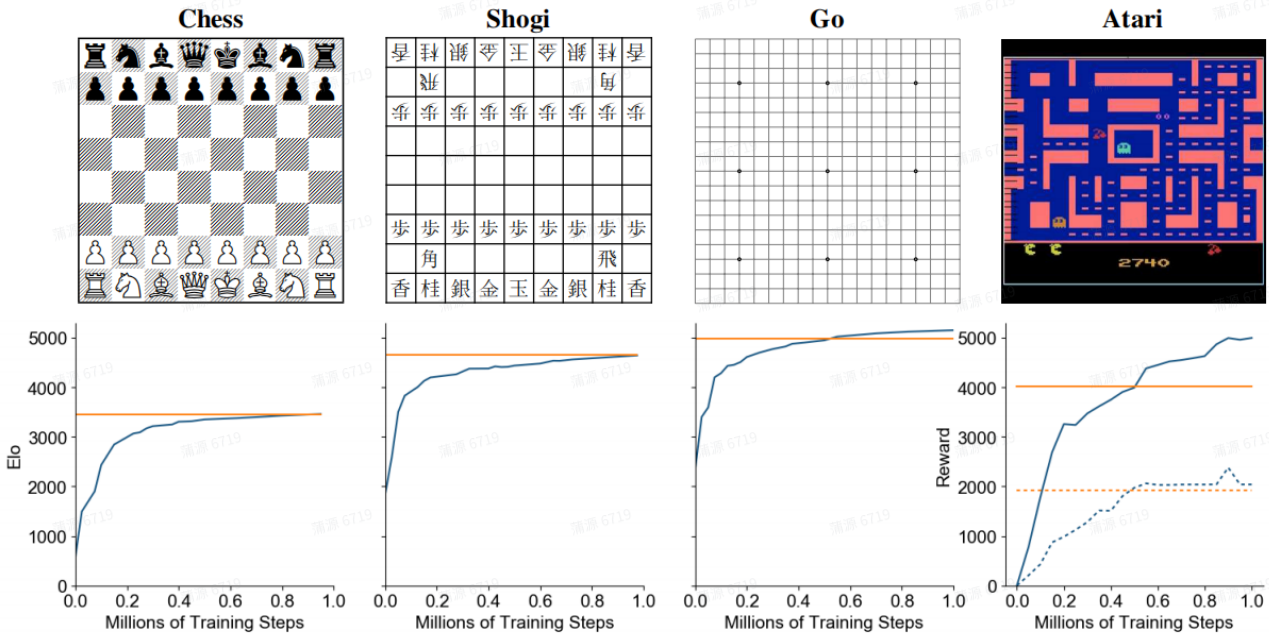
> - hypothetical steps：在每种环境下，作者训练 MuZero 的 hypothetical steps K = 5 ，即通过 dynamics network 从当前状态向下推导5步，在每一步给出转移到的 state 和相应 reward， 并根据 prediction function 得到预测的 policy 和 value，在这5步上计算相应的 loss 更新网络。

参数	hypothetical steps	discount	mini-batches size	simulations for each search	unroll steps num
棋类游戏	5	0.97	2048	800	5
Atari			1024	50	

(表1：MuZero 主要超参数)

其他详细参数见补充材料伪代码。

4.2 实验结果



(图8: 在国际象棋、Shogi、围棋和 Atari 上随着训练步数 MuZero 的学习曲线。)

4.2.1 经典棋类游戏

在图8中，展示了在国际象棋、Shogi、围棋和 Atari 上随着训练步数 MuZero 的学习曲线。 蓝色线为 MuZero 的 Elo 随着训练步数增加的变化。橘色线为 AlphaZero 的 Elo。由图8第3幅图可以看出，在 Go 中，MuZero 表现稍好于 AlphaZero ，且 MuZero 在搜索树的每个节点计算量更小。

4.2.2 Atari

图8第4幅图中，纵轴 reward 表示游戏中的得分，虚线表示57个游戏上得分的中位数，实线表示57个游戏上得分的平均数。蓝色线为 MuZero 的得分随着训练步数的变化，橙色水平线为当时的 SOTA--R2D2（model-free）的得。可以看到，Atari 中，MuZero 达到了当时 SOTA R2D2 (model-free) 的水平，并大大超出之前同类 model-based 方法中最好的 SimPLe

4.2.3 Reanalyze

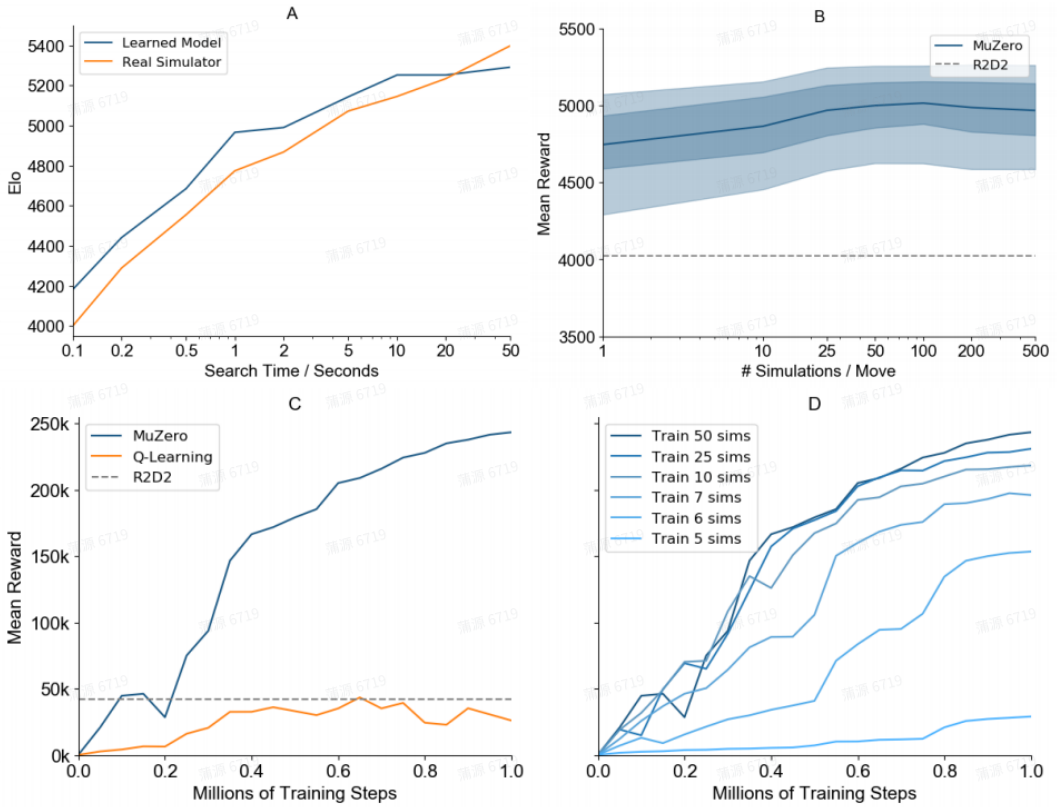
Reanalyze：简单来说，Reanalyze通过使用最新的网络参数重新运行MCTS来重新分析旧的轨迹，得到新的 target 以供训练。从表2可以看出，在 Atari 上，MuZero Reanalyze 优于之前的无模型方法。

Agent	Median	Mean	Env. Frames	Training Time	Training Steps
Ape-X [18]	434.1%	1695.6%	22.8B	5 days	8.64M
R2D2 [21]	1920.6%	4024.9%	37.5B	5 days	2.16M
MuZero	**2041.1%**	**4999.2%**	20.0B	12 hours	1M
IMPALA [9]	191.8%	957.6%	200M	–	–
Rainbow [17]	231.1%	–	200M	10 days	–
UNREAL[a] [19]	250%[a]	880%[a]	250M	–	–
LASER [36]	431%	–	200M	–	–
MuZero Reanalyze	**731.1%**	**2168.9%**	200M	12 hours	1M

(表2：MuZero 与之前算法在 Atari 上效果的比较。)

4.2.4 学习模型的作用

作者为了探究 Muzero 学习到的环境模型的作用，在棋类游戏 Go 以及 Atari 游戏中的 Ms. Pacman 游戏环境上做了如下系列实验。



（图9：MuZero 在 Go上的评估 (A)，在57个 Atari 游戏上的评估 (B)，和在 Ms. Pacman 的评估 (C-D)。）

围棋环境下规划的可扩展性 (Scalability)

- 如图3 (A) 所示，作者对比了已经训练好的使用完美环境模型的AlphaZero，和已经训练好的使用学习模型模拟环境动态机制的MuZero，在规定不同搜索时间下的性能。可以看到 MuZero 的性能和使用完美环境模型的 AlphaZero 差距不大。

Atari 环境下规划的可扩展性 (Scalability)

- 如图3 (B) 所示，使用训练好的 MuZero，但是 MCTS 时使用不同的 simulations 次数（每次 move），以 R2D2 为基线，观察平均奖励的变化。可以看到在 Atari 环境中效果与 R2D2 相近，但是性能优势没有GO上面那么明显。但是同时观察到，MuZero 只进行一次simulations（仅依靠策略网络）效果依然不错，说明策略网络在训练中已经内化了搜索的经验

Model-based 方法与 Model-free 比较

- 如图3 (C) 所示，作者将 MuZero 的 Loss 替换为 R2D2 中 Q-learning 的损失函数、策略价值双头网络替换为单头的Q值网络，训练中不进行 MCTS，得到一个 'Model-free' 版 MuZero，与原版性能进行比较。

- 可以看到 MuZero 的 Q-Learing 版本实现达到了与 R2D2 相同的最终分数，但与基于 MCTS 的训练相比，提高得更慢，最终性能也要差得多。作者认为 MuZero 基于搜索的策略改进步骤比 Q 学习所使用的高偏差、高方差目标提供了更强的学习信号。

获取训练数据的 simulation 次数

- 如图3 (D) 所示，作者探究了在 Ms. Pacman环境中，训练时采用不同 simulation 次数对性能带来的影响。
- 明显看到当训练时的 simulation 次数增加时，策略的改进速度更快。另外可以观察到 MuZero 在 simulation 次数小于动作数量时仍然可以有效地学习，比如 Ms. Pacman 环境中可选择的动作为 8 个，simulation 次数为 6 时算法性能依然很好。

5. 总结与展望

人工智能的许多突破都是基于**高性能规划或无模型强化学习**方法。在本文中，作者介绍了一种结合了两种方法优点的方法。作者的算法 MuZero 不仅在逻辑复杂的棋盘游戏，如国际象棋和围棋中与高性能规划算法的超人类水平的性能相匹配，而且在视觉复杂的 Atari 游戏中超过了最先进的无模型RL算法。至关重要的是，作者的方法不需要任何游戏规则或环境动力学的知识，这潜在地为将强大的学习和规划方法应用于不存在完美模拟器的大量现实世界的领域铺平了道路。

- MuZero 的优点
  a. 不需要先验知识：MuZero 可以从零开始学习如何玩一个游戏，而无需提前了解游戏规则和状态表示。相比之前的强化学习算法，MuZero 不需要手动设计游戏的特征或预先分析游戏的结构，这使得它更具有普适性。
  b. 适用于多种类型的游戏：MuZero 可以用于训练多种类型的游戏，包括棋类、桥牌、扑克等。而且不需要针对不同的游戏进行特定的修改，MuZero 是一个更为通用的强化学习算法。
  c. 更高效的学习：MuZero 相较于 AlphaZero 和 AlphaGo Zero，能够在更短的时间内学会一个游戏，且学习过程更加稳定和可靠。MuZero 的神经网络结构和训练方法对于价值和策略的估计更加准确和精细。
  d. 无需环境模拟器：MuZero 可以直接学习游戏的状态转移和奖励函数，而无需构建一个环境模拟器。这使得 MuZero 更加适用于实际应用场景，因为模拟环境的建立需要消耗大量的计算资源和时间。
- MuZero 的缺点：
  - 需要大量的计算资源和时间：与之前的强化学习算法相比，MuZero 需要较多的计算资源和时间来训练模型。尤其对于复杂的游戏，需要更长的训练时间和更强的计算能力。
  - 对状态表示的依赖性：虽然 MuZero 不需要先验知识来学习一个游戏，但它对于状态表示仍然具有一定的依赖性。不同的状态表示可能会影响 MuZero 的性能和表现，因此需要一定的领域知识来进行选择和优化。
- 未来的研究方向包括：

- 如何与 model-based RL 的最新进展结合以进一步提升算法的样本效率？
- 如何进行 MCTS 的加速?
- 如何让 MuZero 学习得到的表征空间更加具有语义和可解释性?

6. 参考文献

[1] Schrittwieser, J., Antonoglou, I., Hubert, T. *et al.* Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature* **588**, 604–609 (2020).

[2] MuZero算法过程详解， https://mp.weixin.qq.com/s/dTJ7NvVs7qjF2FRoAZ_WlQ , 2021.

[3] MuZero in Tensorflow，https://github.com/kaesve/muzero，2020.

[4] Visualizing MuZero Models, ICML 2021.

[5] 不知道规则也能下围棋！？【恐怖如斯 MuZero】【下】_哔哩哔哩_bilibili, https://www.bilibili.com/video/BV1Ey4y1s7zB/?spm_id_from=333.337.search-card.all.click&vd_source=35dd2c5bd9fbf55c9681c89ce63d38c1 , 2021.