Gumbel MuZero

1. 概述

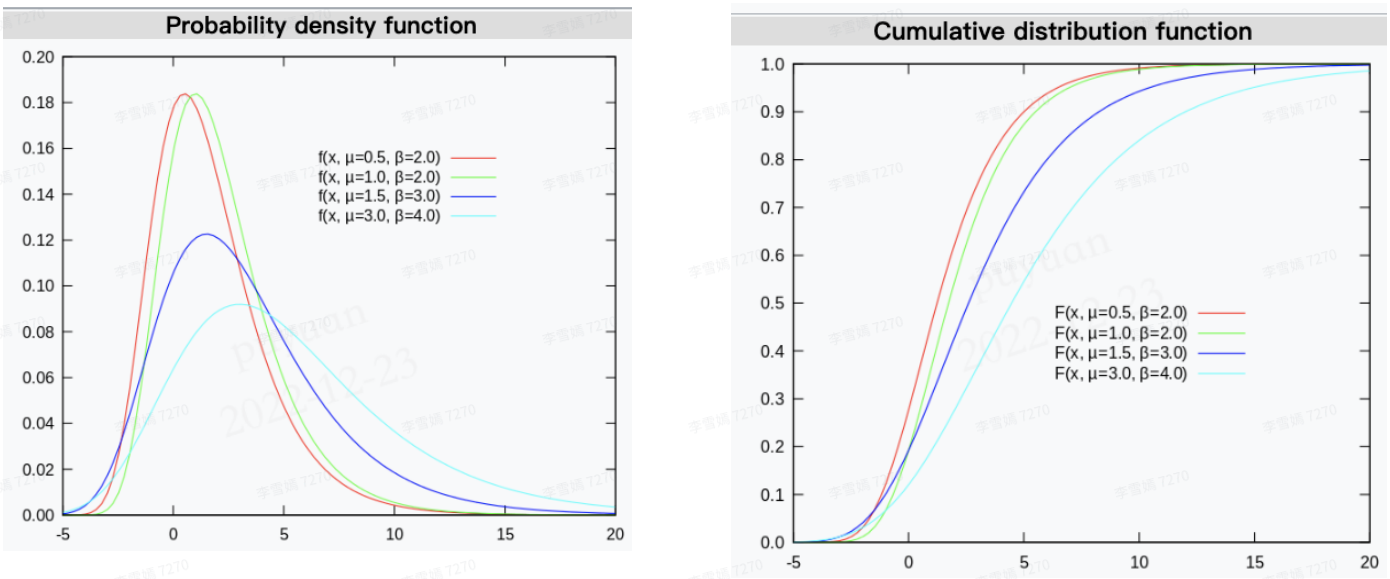
AlphaZero 通过策略迭代和树搜索的结合，在 2018 年于棋类游戏上获得了 SOTA 效果。然而，AlphaZero 需要访问到搜索树根节点的所有合法动作才能更新策略网络，导致了较大的计算开销。为了解决这一问题，Gumbel AlphaZero 采用 Gumbel Top-k trick 无重复地采样动作从而实现策略提升，而非 AlphaZero 中启发式的动作选择方法。同时，作者也基于 MuZero 学习环境模型来进行树搜索的思路设计了 Gumbel MuZero [1]，当模拟次数极少时，Gumbel MuZero 在棋类游戏和 Atari 等图像输入环境上取得了明显的效果提升。

2. 研究背景与相关工作

2.1 Gumbel distribution

Gumbel 分布 [2]（也称为 **I 型广义极值分布**）用于对各种分布的多个样本的最大值（或最小值）的分布进行建模。例如每天各个小时的降水量服从正态分布 $\mathcal{N}(\mu,\ \sigma)$，则必有一个小时的降水量是最大值，这个最大值的分布就服从 Gumbel 分布。

Gumbel 分布的累积分布函数为 $F(x;\mu,\beta)=e^{-e^{-(x-\mu)/\beta}}$，概率密度函数 $f(x)=\frac{1}{\beta}e^{-(z+e^{-z})}$，这里$z=\frac{x-\mu}{\beta}$，其示意图如图1所示。



（图 1：Gumbel 分布的概率密度函数和累计分布函数示意图。）

- **标准 Gumbel 分布**
  - 标准的 Gumbel 分布是这样的情况 $\mu=0,\ \beta=1$
  - 具有累积分布函数： $F(x)=e^{-e^{(-x)}}$

- 和概率密度函数： $f(x) = e^{-(x+e^{-x})}$
- 在这种情况下，众数为 0，中位数为 $-\ln(\ln(2)) \approx 0.3665$， 均值为 $\gamma \approx 0.5772$ （Euler–Mascheroni 常数），标准差为 $\pi/\sqrt{6} \approx 1.2825$

2.2 Gumbel-max trick

Gumbel-max trick 一般应用在 Gumbel-softmax [3] 中进行梯度近似。本文中，作者用 Gumbel-top-k trick 执行无重复抽样（sample without replacement）。假设 π 是 categorical distribution，$logits(a)$ 表示动作 a 的 logits， $logits \in \mathbb{R}^k$ 。从分布 π 采样一个样本 A，相当于首先从标准 Gumbel 分布中采样 k 个样本 $g(a)$，然后取 $\arg\max$：

$$(g \in \mathbb{R}^k) \sim Gumbel(0) \tag{1}$$
$$A = argmax_a(g(a) + logits(a)) \tag{2}$$

2.3 Gumbel-top-k trick

Gumbel-max trick 则是采样 k 个样本后，如公式（3）所示，通过取 n-top 样本，扩展到无重复采样 n 个动作，如公式（4）-（6）所示：

$$(g \in \mathbb{R}^k) \sim Gumbel(0) \tag{3}$$
$$A_1 = \arg\max_a(g(a) + logits(a)) \tag{4}$$
$$\dots \tag{5}$$
$$A_n = \arg\max_{a \notin \{A_1, \dots, A_{n-1}\}} (g(a) + logits(a)) \tag{6}$$

此时的 n-top actions 可以记为 $\text{argtop}\ (g + logits, n) = A_1, A_2, ..., A_n$ 。

2.4 AlphaZero

AlphaZero 在 search 时/规划时遵循以下两种规则选择动作：

- 在 root 节点，首先对 policy 添加 dirichelet 噪声，然后根据 PUCB 公式选择动作：

$$a = \arg\max_a [Q(s,a) + P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} (c_1 + \log(\frac{\sum_b N(s,b) + c_2 + 1}{c_2}))] \tag{7}$$

  然而，在simulation次数较小的情况下，对动作的Q值估计存在较大偏差，此时根据PUCB公式的策略选择不能保证提升策略 π 。

- 在非 root 节点， 不对 policy 添加噪声，直接根据 PUCB 公式选择动作。

3. Gumbel MuZero 算法

3.1 Root 节点的动作选择策略

在搜索树的根节点上，AlphaZero 和 MuZero 在与真实的环境交互前，可以探索 n 次模拟。这一问题可以建模为 一个带有预测器的确定性赌博机（a deterministic bandit with a predictor），且这一建模还可以扩展到随机性赌博机和 MCTS 中。

Bandit 建模

一个 k-armed deterministic bandit 可以表示为一个 Q 值向量 $q \in \mathbb{R}^k$， 其中 $q(a)$ 是动作 a 对应的 Q 值。智能体与 bandit 交互 n 轮（n 次模拟）， 在每次模拟 $t \in 1, 2..., n$ 时，智能体从 k 个动作中选择一个 $A_t \in 0, 1..., k-1$，并访问这个动作以观测到 Q 值 $q(A_t)$。

目标

最大化最后一个动作 A_{n+1} 获得的 Q 值，即最大化 $\mathbb{E}[q(A_{n+1})]$。该目标等价于最小化 simple regret，即在这一节点上最佳决策的回报期望和时机选择的决策的回报期望之间的差异。先前工作 [4] 已经证明，在搜索树的根节点上应考虑的是 simple regret 来选择动作。k-armed deterministic bandit 问题的影响在 $k > n$ 时变得显著，例如 19x19 围棋有 392 个可能的动作，但模拟次数仅为 $n = 2$ 时，如何选择动作进行模拟变得难以判断。幸运的是，策略网络可以帮助我们实现这一目标。

预测器

在带有预测器的赌博机（bandit-with-predictor）的设定中， 智能体可以借助预测器（即 Gumbel MuZero 中的策略网络）进行预测。在与 bandit 进行交互之前，策略网络通过产生一个概率分布 π 来预测最好的动作。智能体利用策略网络的预测可以做出更明智的选择。

策略提升

自然地，我们希望智能体能够表现得不差于策略网络，即取得策略提升，此时智能体选择的动作应满足 $\mathbb{E}[q(A_{n+1})] >= \sum_a \pi(a)q(a)$， 这里 $\pi(a)$ 是策略网络对于动作 a 的预测。通过这一保证，策略网络可以通过建模更优的策略以保持策略提升。

planning with gumbel

Gumbel MuZero 在策略网络 π 的帮助下，通过 Gumbel-Top-k trick 实现了上述确定性赌博机场景下的策略提升。具体来说，给定原始动作个数 K 和模拟次数 n，首先通过 Gumbel-Top-k trick 采样 n 个动作。获取这 n 个动作的 $q(a)$ 后，选出具有最大的 $g(a) + logits(a) + \sigma(\hat{q}(a))$ 的动作，其中 σ 为任意单调递增的函数。注意在采样 n 个动作和选择动作的过程中，使用的是同一个 $g(a)$，这避免了双重计数带来的偏差：

Algorithm 1 Policy Improvement by Planning with Gumbel

Require: k: number of actions.
Require: $n \leq k$: number of simulations.
Require: $\text{logits} \in \mathbb{R}^k$: predictor logits from a policy network π.
  Sample k Gumbel variables:
    $(g \in \mathbb{R}^k) \sim \text{Gumbel}(0)$
  Find n actions with the highest $g(a) + \text{logits}(a)$:
    $\mathcal{A}_{\text{topn}} = \texttt{argtop}(g + \text{logits}, n)$
  Get $q(a)$ for each $a \in \mathcal{A}_{\text{topn}}$ by visiting the actions.
  From the $\mathcal{A}_{\text{topn}}$ actions, find the action with the highest $g(a) + \text{logits}(a) + \sigma(q(a))$:
    $A_{n+1} = \arg\max_{a \in \mathcal{A}_{\text{topn}}} (g(a) + \text{logits}(a) + \sigma(q(a)))$
  return A_{n+1}

（图 2：Planning with Gumbel 算法伪代码。）

以上算法产生了策略提升，因为如下不等式对于任意 Gumbel 分布 g 都满足：

$$q(\arg\max_{a \in \mathcal{A}_{\text{topn}}}(g(a) + \text{logits}(a) + \sigma(q(a)))) \geq q(\arg\max_{a \in \mathcal{A}_{\text{topn}}}(g(a) + \text{logits}(a))) \tag{8}$$

因此Q值期望也满足：

$$\mathbb{E}[q(A_{n+1})] \;\geq\; \mathbb{E}_{A \sim \pi}\left[q(A)\right] \;=\; \sum_{A} \pi(A) q(A) \tag{9}$$

其中 $\arg\max_{a \in \mathcal{A}\,\text{topn}}(g(a) + \text{logits}(a))$ 等价于从策略网络 π 中采样，包含在 logits 中的先验知识可以在部分可观测环境中或 Q 值为近似的或随机的情况下产生帮助。

Planning on a stochastic bandit

接下来考虑随机性的情况，随机性赌博机（stochastic bandit）只提供了期望 Q 值 $q(a)$ 的随机估计，因此使用经验均值（empirical mean） $\hat{q}(a)$ 表示 Q 值而不是确定性 Q 值 $q(a)$。显然，如果一个动作的访问次数越多，他对应的经验均值 $\hat{q}(a)$ 也就越准确。因此，重要的是 **访问什么动作** 以及 **每个动作访问多少次**，这可以从 2 个方面进行控制：

- 第一，控制不重复采样的动作数量 （the number of actions sampled without replacement）
- 第二，使用 bandit 算法来高效地探索采样动作集合 （the set of sampled actions）

有许多 bandit 算法可以实现 simple regret 最小化，Gumbel MuZero 中采用的是 **sequential halving**。这是因为 sequential halving 没有与问题相关的 （problem-dependent） 超参数，相对于 UCB-E [5] 和 UCB $\sqrt{\cdot}$ [6] 算法更容易调参。

Sequential Halving with Gumbel

以下伪代码展示了采用 Gumbel-Top-k trick 的 sequential halving 算法：

Algorithm 2 Sequential Halving with Gumbel

Require: k: number of actions.
Require: $m \leq k$: number of actions sampled without replacement.
Require: n: number of simulations.
Require: $\text{logits} \in \mathbb{R}^k$: predictor logits from a policy network π.
Sample k Gumbel variables:
$(g \in \mathbb{R}^k) \sim \text{Gumbel}(0)$
Find m actions with the highest $g(a) + \text{logits}(a)$:
$\mathcal{A}_{\text{topm}} = \texttt{argtop}(g + \text{logits}, m)$
Use Sequential Halving with n simulations to identify the best action from the $\mathcal{A}_{\text{topm}}$ actions, by comparing $g(a) + \text{logits}(a) + \sigma(\hat{q}(a))$.
$A_{n+1} = \arg\max_{a \in Remaining}(g(a) + \text{logits}(a) + \sigma(\hat{q}(a)))$
return A_{n+1}

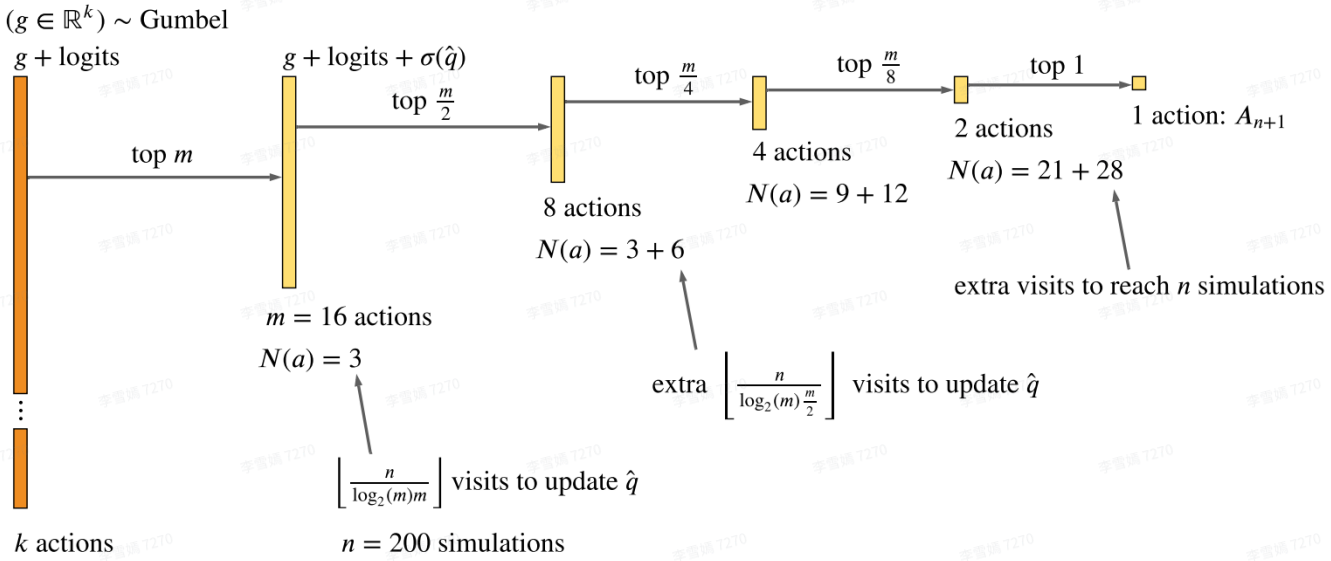
（图 3：Sequential Halving with Gumbel 算法伪代码。）

这一算法的目的是为了识别出使得 $g(a) + logits(a) + \sigma(\hat{q}(s))$ 最大的动作，具体步骤为：

- 在 root 节点，先根据 gumbel-top-k trick 从策略网络 π 中采样 m 个动作
- 通过 sequential halving 算法确定这 m 个动作中每个动作的访问次数，得到他们的估计 Q 值 $\hat{q}(a)$，通过比较 $g(a) + logits(a) + \sigma(\hat{q}(a))$ 找到这 m 个动作中最好的动作

图形化示例

图 4 中展示了 Sequential Halving with Gumbel 的一个示例。在 $k > n$ （例如围棋中 $k = 362, n = 200$）的情况下，首先根据 $g + logits$ 无重复地采样 top-m 个动作，例如下图中 $m = 16$。接下来，sequential halving 将 200 次模拟的开销分为 $log_2(m) = 4$ 个阶段进行分配。每个阶段，所有动作都被分配 $\frac{n}{log_2(m)m}$ 模拟次数。在每个阶段最后，有一半的动作被舍弃，另一半动作进入下一个阶段。最终选择的动作即为 $g + logits + \sigma(\hat{q})$ 最高的动作。



（图 4：Sequential Halving with Gumbel 算法在 $n = 200$ 时的示例。）

代码示例

以下为 sequential halving 的代码实现：

```python
def get_table_of_considered_visits(max_num_considered_actions, num_simulation
    """Returns a table of sequences of visit counts.

    Args:
        max_num_considered_actions: The maximum number of considered actions.
            The `max_num_considered_actions` can be smaller than the number o
            actions.
        num_simulations: The total simulation budget.

    Returns:
        A tuple of sequences of visit counts.
        Shape [max_num_considered_actions + 1, num_simulations].
    """
    return tuple(
        get_sequence_of_considered_visits(m, num_simulations)
        for m in range(max_num_considered_actions + 1))

max_num_considered_actions = 4
num_simulations = 8
table = get_table_of_considered_visits(max_num_considered_actions, num_simula
print('table: ', table)
```

- 此时得到的 table=((0, 1, 2, 3, 4, 5, 6, 7), (0, 1, 2, 3, 4, 5, 6, 7), (0, 0, 1, 1, 2, 2, 3, 3), (0, 0, 0, 1, 1, 2, 2, 3), (0, 0, 0, 0, 1, 1, 2, 2))，
- m=4 时，table[4]=(0, 0, 0, 0, 1, 1, 2, 2），
- 每个数字代表的是第几个阶段，0 是第 0 个阶段，1 是第 1 个阶段。所以四个 0 表示在第一个阶段中，top4 的动作进行一次模拟，两个 1 表示 top2 的动作进行一次模拟，两个 2 表示 top2 的动作进行一次模拟，所以最终 top4 的动作分别模拟了[3， 3， 1， 1]次。

3.2 Improved policy 的构造方式

上述过程中得到的 A_{n+1} 是从潜在的 improved policy 中得到的，为了将 improved policy 提取到策略网络中，一种朴素的想法是通过如下 policy loss，更新 policy π 以增大选择 A_{n+1} 的概率：

$$L_{\text{simple}}(\pi) = -\log \pi(A_{n+1}) \tag{10}$$

然而上述 policy loss 没有充分利用搜索中得到的信息。例如，搜索过程中为访问的动作提供的 $q(a)$ 实际上反映了策略的回报期望，可以用于指导策略选择回报期望更高的动作。因此 Gumbel MuZero 中利用了这一信息，构造如下 completedQ 值：

$$\text{completedQ}(a) = \begin{cases} q(a) & \text{if } N(a) > 0 \\ v_\pi & \text{otherwise,} \end{cases} \tag{11}$$

其中已被访问过的节点的 $q(a)$ 已知，而未被访问过的节点则可以通过 value network 得到其回报期望的估计 $\hat{v}_\pi$。此外，对于 off-policy data，作者提出了一种结合了 $\hat{v}_\pi$ 和其他已被访问节点的平均 Q 值的方法：

$$v_{\text{mix}} = \frac{1}{1+\sum_b N(b)}(\hat{v}_\pi) + \frac{\sum_b N(b)}{\sum_{b\in\{b:N(b)>0\}} \pi(b)} \sum_{a\in\{a:N(a)>0\}} \pi(a)q(a) \tag{12}$$

此时，可以构造 improved policy 如下：

$$\pi' = \text{softmax}(\text{logits} + \sigma(\text{completedQ})) \tag{13}$$

这里 σ 为单调递增函数。为了将 improved policy 提取到策略网络中，更新策略网络的损失函数如下：

$$L_{\text{completed}(\pi)} = \text{KL}(\pi', \pi) \tag{14}$$

> 在实验中，σ 函数设置为：
>
> $$\sigma(\hat{q}(a)) = \left(c_{\text{visit}} + \max_b N(b)\right) c_{\text{scale}} \hat{q}(a) \tag{15}$$
>
> 其中 $\max_b N(b)$ 是最频繁访问的动作的访问次数。这一函数保证了当动作的访问次数逐渐增加时，增大 $\hat{q}(a)$ 的 scale，即逐渐减小先验策略的影响。

3.3 非 root 节点的动作选择策略

在先前工作 [7] 中已经证明，AlphaZero 在non-root节点上选择动作的过程可以看作一种正则化的策略优化过程，因此作者考虑直接从 improved policy π' 中采样动作。然而，采样得到的动作是随机性的，会给估计的 Q 值带来不想要的高方差。作者提出一种**确定性的**动作选择方法，使得选择动作 a，能够最小化 improved policy π' 和选择动作 a 后导致的规范化的 visit counts 的距离：

$$\arg\min_a \sum_b (\pi'(b) - \underbrace{\frac{N(b)+\mathbb{I}(a=b)}{1+\sum_c N(c)}}_{\text{Normalized visit counts, if taking a.}})^2 \tag{16}$$

代数化简后得到：

$$\arg\max_a (\pi'(a) - \frac{N(a)}{1+\sum_b} N(b)) \tag{17}$$

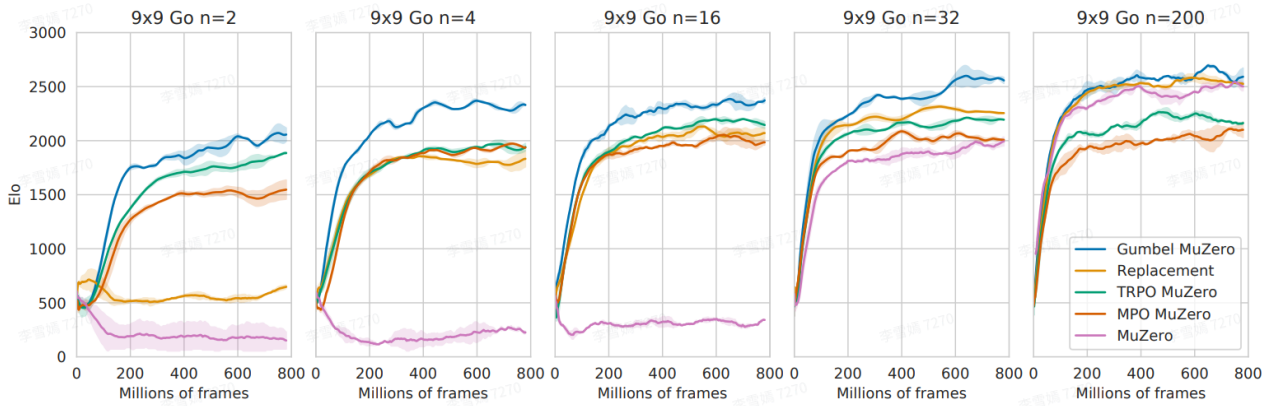
这种确定性动作选择方法，使得选出各个动作的概率是与 improved policy π' 的分布成正比，同时避免了高方差。但是作者推荐只是在非 root 节点使用，而在 root 节点，Gumbel noise 的加入有助于在学习的不同阶段探索不同动作，保证策略提升。

4. 实验结果

4.1 核心实验

4.1.1 Go

在围棋上，作者比较了 MuZero 及其变种在不同 simluation 次数下的效果，如图 5 所示。可以观察到当 simulation 次数降低到 16 次及以下时，MuZero 方法无法学习到有效策略。而 Gumbel MuZero 在 simulation 次数仅为 2 时也能够稳定学习到较好策略。

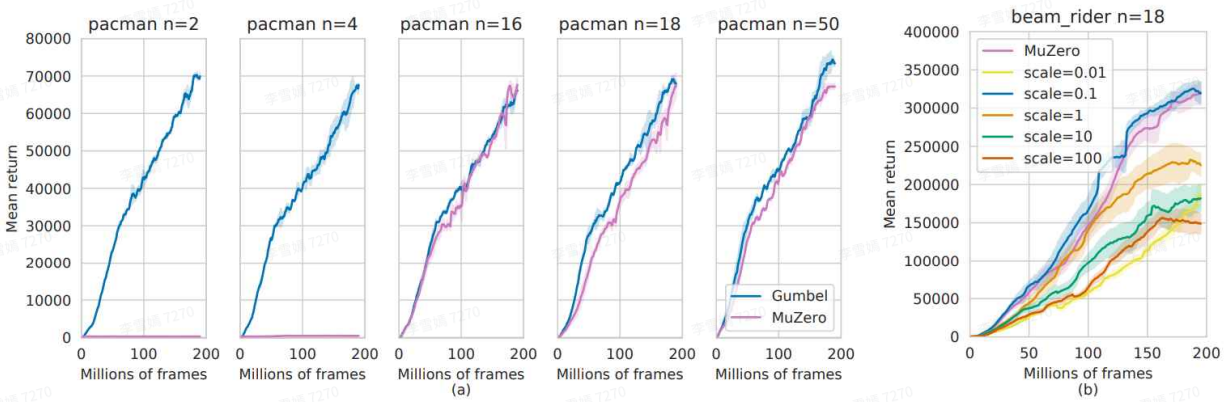


（图 5：在围棋中，MuZero 及其变种算法在不同模拟次数下的效果比较。其中 **Gumbel MuZero** 对 MuZero的改进为：在 root 节点上采用 Sequential Halving with Gumbel，且策略更新目标为采用completedQ 的 improved policy。**Replacement** 表示在动作采样过程中进行有重复采样的Gumbel MuZero。**TRPO MuZero** 表示采用TRPO正则项 $KL(\pi, \pi_{\text{new}})$ 的 Gumbel MuZero。**MPO MuZero**表示采用MPO正则项 $KL(\pi_{\text{new}}, \pi)$。**Full Gumbel MuZero** 表示在Gumbel MuZero的基础上，非root节点采用 3.3 中确定性的动作选择。）

4.1.2 Atari

Atari 存在不同游戏的奖励尺度不同的问题。作者将 Q value 进行 max-min 归一化，并设置相关超参数 $c_{\text{visit}} = 50$，$c_{\text{scale}=0.1}$。如图 6 左边所示，在 MsPacman 上，作者比较了 Gumbel MuZero 和 MuZero 在不同 simulation 次数下的效果。Gumbel MuZero 在 simulation 次数仅为 2 的情况下也可以稳定训练得到较好策略，同时，在 simulation 次数为 50 时可以达到 SOTA 效果。

另一方面，图 6 右边以 Beam Rider 环境为例，说明了 logits 中包含的信息的重要性。通过调节公式 15 中的系数 c_{scale} 可以控制 logits 和估计 Q 值的比例。从结果中可以观察到，当 $c_{\text{scale}} = 0.1$，即 logits 权重在合理的范围 时，Gumbel MuZero 的效果达到最优。这说明 logits 中包含了重要的先验知识。

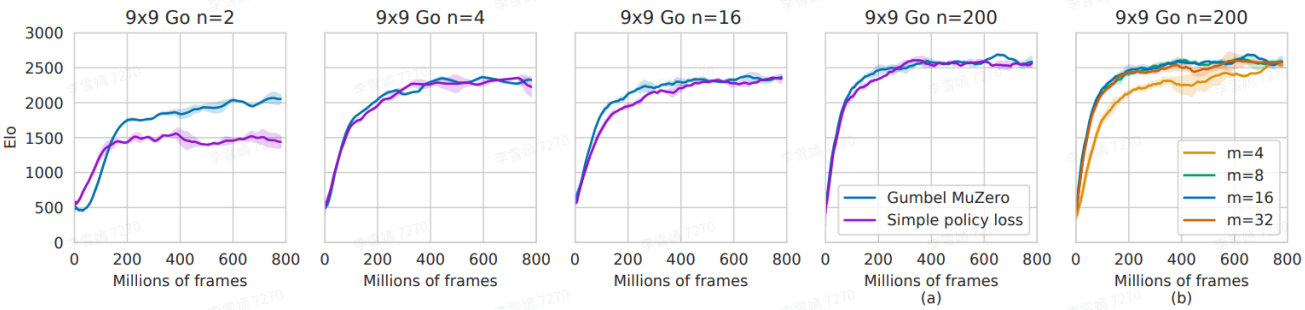


（图 6：（a）MuZero 和 Gumbel MuZero 在 MsPacman 上不同模拟次数的效果比较（b）Gumbel Muzero 在 Beam Rider 上设置不同 c_{scale} 的效果比较。）

4.2 Gumbel MuZero policy loss 与 原始 policy loss 的比较

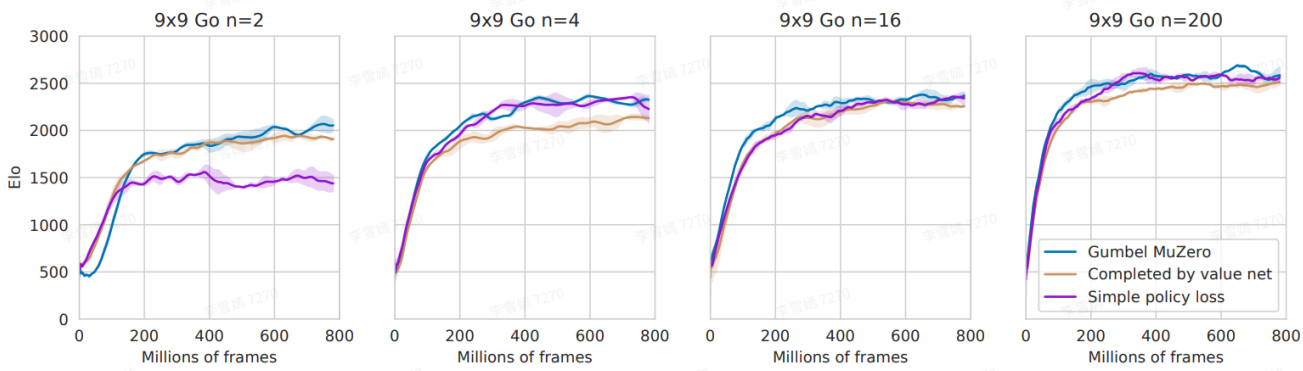
为了说明公式 14 中设计的策略损失的合理性，作者在围棋上比较了 simple loss 和 policy loss with completed Q-values。从图 7 左边可以观察到，在大多数场景下，simple loss 就足够达到较好效果了。但当模拟次数较小（$n = 2$）时，policy loss with completed Q-values 明显比 simple loss 取得了更好效果。

而在图 7 右边，作者比较了 Gumbel MuZero 对采样动作的敏感度。当无重复地采样 $m = 4$ 个动作时，由于所有的 simulation 次数都用于模拟仅有的这几个动作，导致学习速度相较于其他情况更慢。在围棋实验中，作者一般将 m 设置为 $\min(n, 16)$ 。



（图 7：（a）采用 simple policy loss 和 policy loss with completed Q value 的 Gumbel MuZero 在围棋上的效果比较（b）Gumbel MuZero 在不同采样次数下的效果比较。）

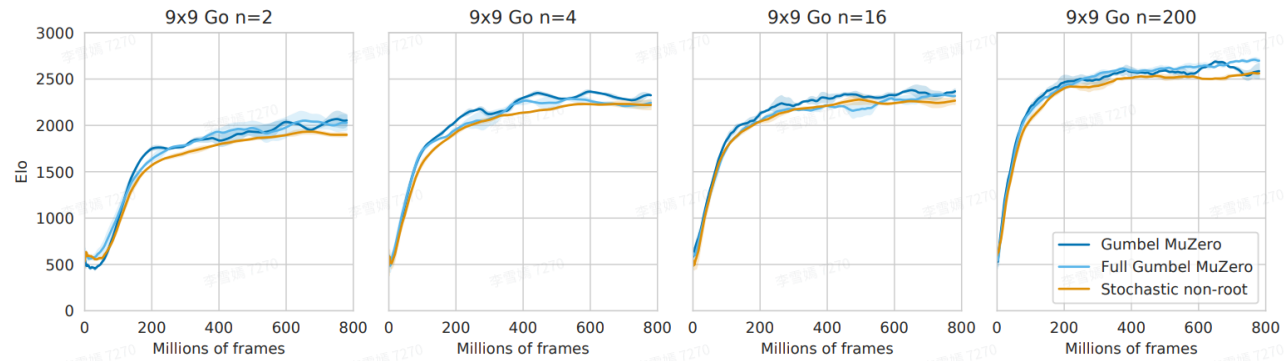
图 8 说明了作者所构造的 policy loss with completed Q value 的效果。当 simulation 次数较小时，以 v_{mix} 为 Q value 的 policy loss 相较于仅用 value network 估计得到的 $\hat{v}_\pi$ 效果更好。



（图 8：Gumbel Muzero 采用 simple policy loss，policy loss with $\hat{v}_\pi$ 和 policy loss with completed Q value 在围棋上的效果比较。）

4.3 Non-root 改进的影响

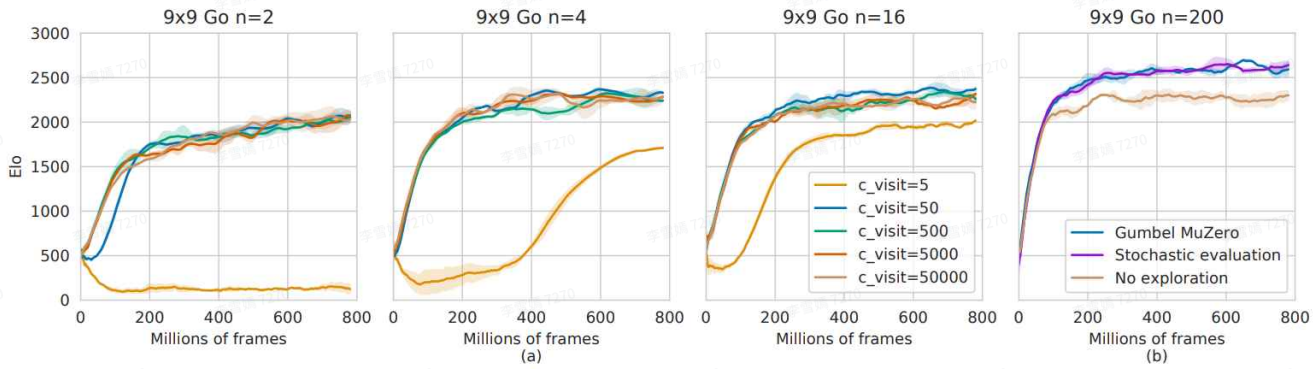
图 9 展示了在非 root 节点上以不同策略选择动作的比较。Gumbel Muzero 选择类似于 MuZero 的方法选择动作，而 Full Gumbel MuZero 选择了公式 17 中的方法选择动作。而 Stochastic non-root 则是根据 improved policy 随机采样动作。可以观察到，确定性选择动作的 Gumbel Muzero 和 Full Gumbel MuZero 相比于随机采样动作的 Stochastic non-root 更为稳定地达到较好效果，这说明了随机采样动作存在方差过大的问题。



（图 9：在9x9 Go上，Gumbel MuZero 在非 root 节点上采取不同的动作选择策略的性能比较。其中**Gumbel MuZero** 对MuZero在root节点的动作选择和策略更新目标上进行了改进。**Full Gumbel MuZero** 在 Gumbel MuZero的基础上，对non-root节点的动作选择也做了改进。**Stochastic non-root**则在non-root节点上根据improved policy π' 随机性选择动作。）

4.4 使用 Gumbel Noise 的影响

图 10 左边展示了在不同 simulation 次数下，Gumbel MuZero 在 Q 值放缩过程中对于 c_{vist} 的敏感度。而右边则展示了在完全信息游戏中，Gumbel noise 带来的影响。可以观察到，由于围棋中的信息是完全的，因此 MuZero 和 Gumbel MuZero 中的探索能够为效果带来明显提升，而 Gumbel noise 并没有呈现出明显影响。

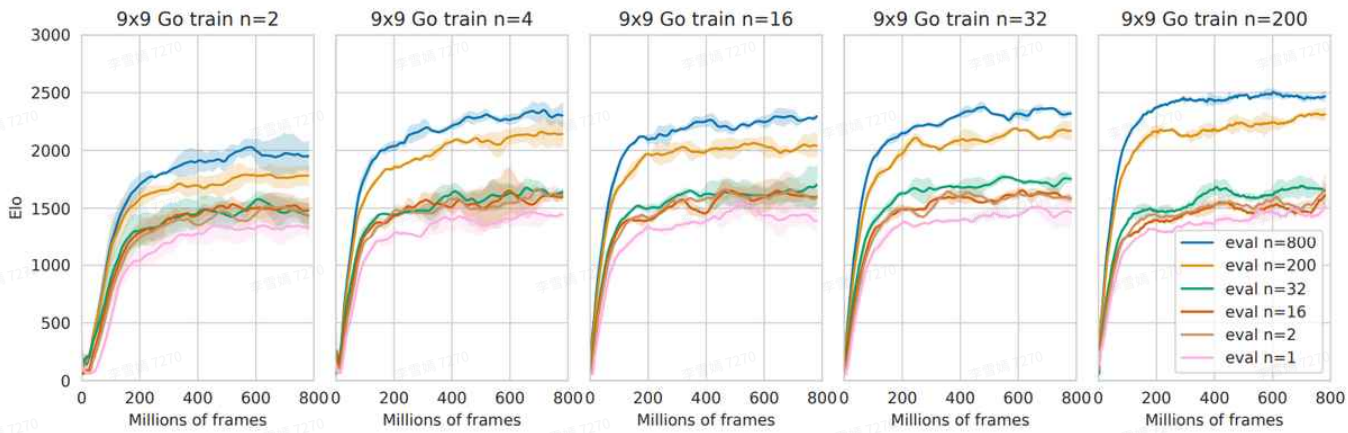


（图 10：（a）Gumbel Muzero 在 Q 值放缩过程中对于 c_{vist} 的敏感度（b）在完全信息游戏中，Gumbel noise 和探索带来的影响。）

4.5 不同 simulation 数量的影响

图 11 展示了训练时 simulation 次数的重要性。

- 例如第一张图中，Gumbel MuZero 在 simulation 次数为 2 的设置下进行训练，分别在 simulation 次数为 1，2，16，32，200，800 的设置下进行 evaluate。此时 evaluate 的 simulation 次数增长并没有为 evaluate 的效果带来明显提升。
- 而最后一张图中，Gumbel MuZero 在 simulation 次数为 200 的设置下进行训练，此时 evaluate 在 $n > 200$ 时效果显著优于 simulation 次数较低的情况。
- 这说明训练时的 simulation 次数决定了 evaluate 时不同 n 设置的性能上限。



（图 11：以不同 simulation 次数进行训练后得到的 Gumbel MuZero，在 evaluate 时，不同 simulation 次数的效果比较。）

5. 总结与展望

总的来说，Gumbel MuZero 为 MuZero 在极少模拟次数的场景下的改进进行了探索，将 MuZero 的应用扩展到了模拟消耗较大的场景中。其改进可以总结为：

- 在 root 节点上，通过 Gumbel-Top-k trick 采样动作以实现策略提升，再通过 Sequential Halving 算法高效访问采样动作集合。

- 在策略更新过程中，结合 logits 和 $q(a)$ 的先验知识，构造 improved policy 为目标策略。
- 在 non-root 节点上，以确定性的方式选择动作，这样的设计既避免了随机性选择动作带来的高方差，又让选择动作的概率与 improved policy 成正比，从而近似AlphaZero中的PUCT动作选择算法的效果。

6. 参考文献

[1] Danihelka I, Guez A, Schrittwieser J, et al. Policy improvement by planning with Gumbel[C]//International Conference on Learning Representations. 2022.

[2] https://zh.wikipedia.org/wiki/%E8%80%BF%E8%B4%9D%E5%B0%94%E5%88%86%E5%B8%83

[3] https://spaces.ac.cn/archives/6705

[4] Bubeck S, Munos R, Stoltz G. Pure exploration in finitely-armed and continuous-armed bandits[J]. Theoretical Computer Science, 2011, 412(19): 1832-1852.

[5] Audibert J Y, Bubeck S, Munos R. Best arm identification in multi-armed bandits[C]//COLT. 2010: 41-53.

[6] Tolpin D, Shimony S. MCTS based on simple regret[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2012, 26(1): 570-576.

[7] Grill J B, Altché F, Tang Y, et al. Monte-Carlo tree search as regularized policy optimization[C]//International Conference on Machine Learning. PMLR, 2020: 3769-3778.