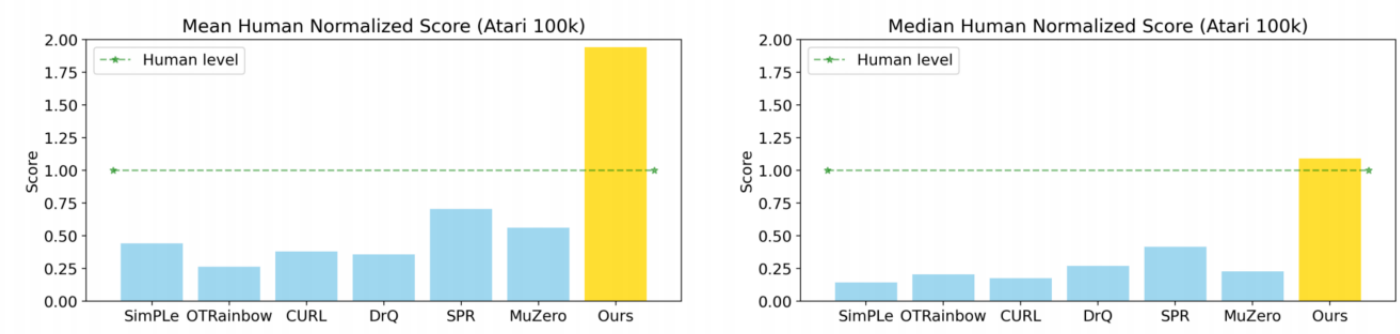
EfficientZero

1. 概述

样本效率（sample efficiency）是强化学习的一个关键挑战，一般来说需要高达数百万或者数十亿的环境步数（env steps）才能够取得相对来说理想的训练效果。 尤其是在 Atari 等基准游戏测试中，如何在有限的交互中达到人类的表现水平依旧是一个困难的目标。 在 MuZero 的基础上，作者们提出了一种高效的视觉 RL 算法 —— EfficientZero。 这个方法在 Atari 100k 的 benchmark 上，只需要大约 2 个小时的训练数据就能达到超越人类的水平。具体来说，如图 1 所示，EfficientZero 在 Atari 游戏上能达到人类平均水平的 194.3%和人类中位数水平的 109%。 与此同时，在 DMControl 100k benchmark 上的效果也超过了将 state（非图像）作为 observation 的 SAC 算法（state SAC）。这也是第一次在训练数量样本有限的情况下，达到超越人类的水平。相比于 DQN 需要 200 million frames 的交互样本数量， EfficientZero 只需要它的五百分之一。(code: https://github.com/YeWR/EfficientZero)



（图 1：EfficientZero 在 Atari 100k benchmark 上与其他方法和人类水平的比较。EfficientZero 在人类平均得分和中位数得分上达到了基线模型的 176%和 163%，是第一个在该环境上能够超越平均人类水平的算法。EfficientZero 的高采样效率和性能让 RL 更贴近真实应用。）

2. 研究背景与相关工作

2.1 样本效率高的强化学习（Sample Efficient Reinforcement Learning）

样本效率 （sample efficiency） 这一话题在过去有很多相关工作。无模型 RL 在有限数据量场景中的性能远优于基于模型的 RL，例如，时序一致性损失与数据增强的结合能够达到较高性能 [1]。而基于模型的 RL 同时用确定性模型（如 RNN）和随机性模型（如 SSM）建模 dynamics model，在图像输入的模拟机器人控制中取得了很好效果 [2]，但其中基于模型的方法被证明是不必要的 [3]。EfficientZero 提出了一种基于模型的 RL 方法，在数据量有限的 Atari 上能够超越人类水平。

2.2 使用 MCTS 的强化学习（Reinforcement Learning with MCTS）

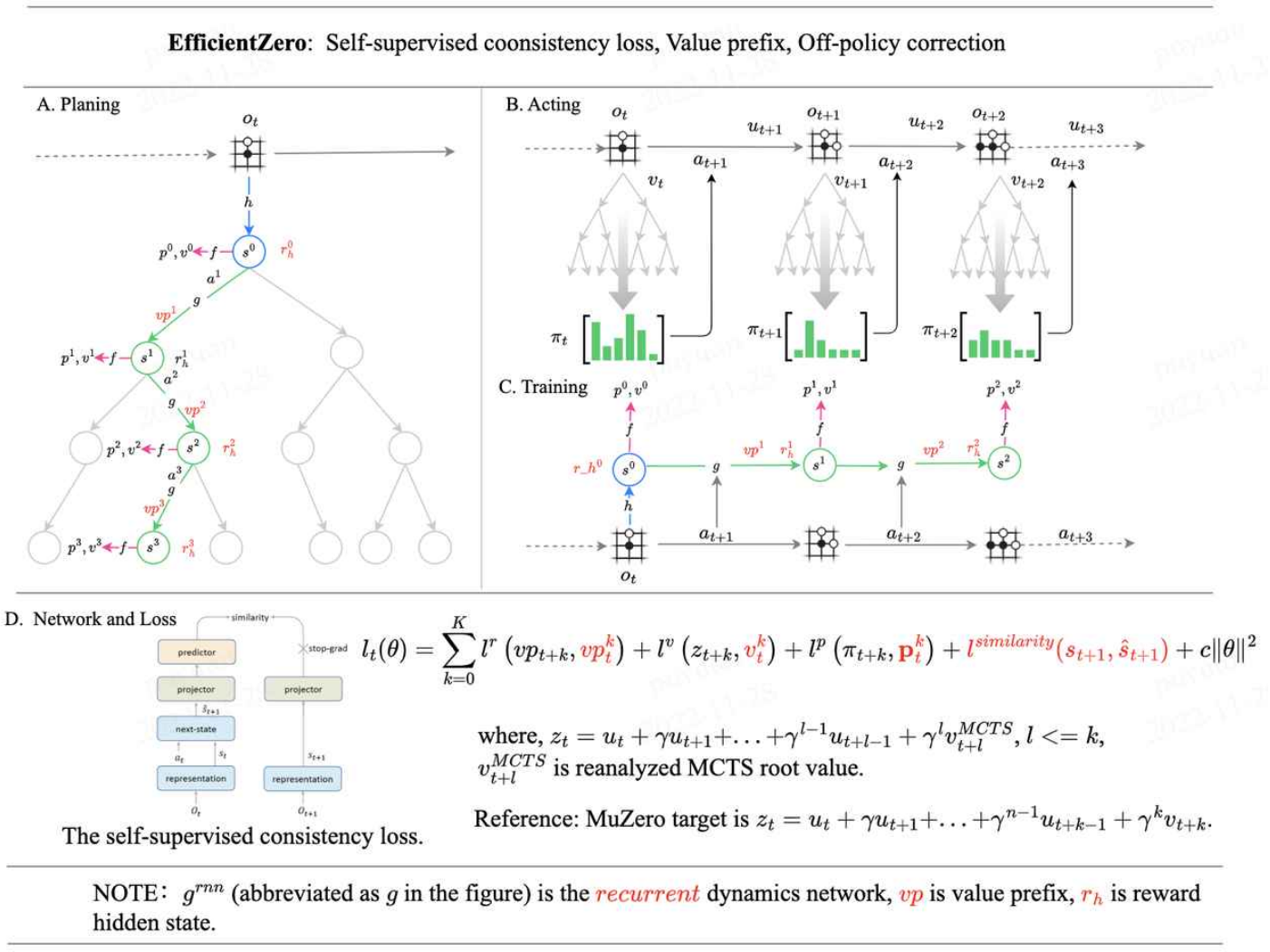
区别于 temporal difference learning 和 policy gradient based 等方法，MuZero 方法使用 MCTS 作为 policy improvement operator 并且在棋盘类游戏之中取得巨大的成功，例如在围棋 [4] ，国际象棋和将棋之中 [5]。在此之后，它又被拓展到了连续动作空间 [6] 和离线数据 [7] 中。 然而，这类基于 MCTS 的方法需要大量的数据，因此样本效率很低。

2.3 多步值估计方法（Multi-Step Value Estimation）

在 Q-learning 的时候，target Q value 往往通过 one step backup 来获得。 在实践中，人们发现把多个 step 的奖励纳入 value 的估计之中，可以得到更快的收敛 [8] [9]。但是多步价值会有 off-policy 的常规问题，即自举 （bootstrap） 过程中的 value 并不是最新的 policy 得到的。为了解决这个问题，EfficientZero 提出了一个 model-based policy correction 方法，它根据当前策略，重新收集真实经验以获得最新奖励，并使用基于模型的估计方法来进行自举，在解决 off-policy 问题和模型的经验利用上达到了平衡。

3. EfficientZero 算法

3.1 概览图



（图 2：EfficientZero 算法概览图。**A**：EfficientZero agent 利用 MCTS 进行规划，其中 representation network h 将连续 t 帧观测编码为 latent state s^0；prediction network f 在 s^k 下预测 policy p^k 和 value prefix vp^k；dynamics network g 预测在 s^{k-1} 下执行动作会转移到的 s^k 和奖励 r^k。**B**：EfficientZero 和环境的交互过程中，从某个观测 o_t 开始，根据 MCTS 中根节点下各个动作的 visit count 的分布 π_t 选择动作 a_{t+1} 与环境交互得到新的观察 o_{t+1} 和回报 u_{t+1}。这样不断互动后得到整轨迹，保存到 replay buffer 中。**C**：EfficientZero 在训练时根据 dynamics network g 展开 k 步，通过预测每一步的 reward， value prefix 和 policy 来学习模型。**D**：EfficientZero 的损失函数定义包含五个部分，第一个部分为 dynamics network 预测的 value prefix 和真实 value prefix 之间的距离，第二个部分和第三个部分为 prediction network 预测的 value 和 policy 与真实值之间的距离，第四个部分是自监督一致性损失，第五个部分为正则化项。符号的含义参考【算法概览图符号表.pdf】。）

当直接把 MuZero 直接应用到少样本的 benchmark 上面时， 它的表现并不好，通过消融实验，作者验证了它存在的三个问题：

- （1）在学习环境模型时缺少监督信号。在 MuZero 里损失函数项分别优化的是 reward，value 和 policy。reward 只是标量信号，且在很多环境中，reward 都是稀疏的。value 是通过自举训练得到的，因此较为 noisy。策略是随着搜索过程训练得到的，因此波动也很大。可以看到，reward，value 和 policy 三者都无法提供足够训练环境模型的监督信号。
- （2）难以处理随机不确定性。由于环境的随机不确定性，当环境较难建模时，随着 MCTS 深度加大，对奖励的预测误差也会随之变大。例如在卡牌游戏环境中，对手卡牌的不可观测性导致环境本身具有随机不确定性，因此对当前状态的奖励预测误差较大。
- （3）multi-step value 更新导致的 off-policy 问题。虽然 multi-step value 让奖励能够更快传播到 value function。但在数据有限的情况下，使用 off-policy 的旧策略收集的数据中，奖励是由旧策略得到的，这导致对 target-value 的更新不准确，导致 value function 不容易收敛。

针对以上问题，EfficientZero 做了如下的三点改进，即：

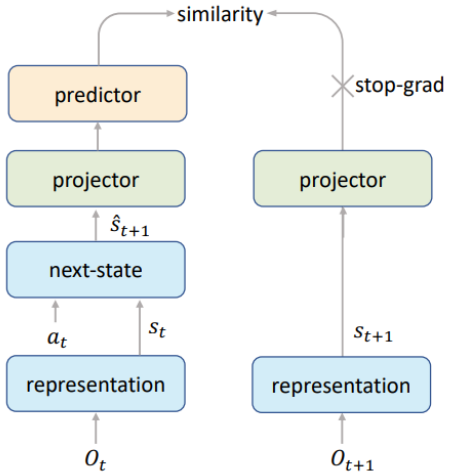
- （1） 设计自监督（时序）一致性损失。
- （2） 使用端到端的学习 value prefix，预测时间段内奖励值之和，降低预测 reward 不准导致的误差。
- （3） 改变 Multi-step reward 的算法，使用一个自适应的展开长度来纠正 off-policy target。

下面将依次详细介绍以上三点改进：

3.2 自监督（时序）一致性损失

在以前 MCTS 工作中，环境模型有些是直接已知的，有些则通过训练 rewards， values 和 policies 来得到。然而，这些变量往往只是标量，无法提供足够的训练信息。当环境的奖励是稀疏的，或者自举更新的 value 不准确时，这一问题尤为严重。 为了得到足够的模型训练信号，作者提出了**自监督一致性损失，即 dynamic model 预测得到的 $\hat{s}_{t+1}$ 应该和实际的 s_{t+1} 一致。** 这一方法可以让实际的 s_{t+1} 作为监督信号，对 dynamic model 的预测与训练进行监督，而 s_{t+1} 往往是几百维的张量，相对于标

量有更多的训练信号。值得注意的是，隐藏状态 s_{t+1} 和预测状态 $\hat{s}_{t+1}$ 的一致性可以直接通过 dynamics function 来构造，而不需要提供额外的模型。



（图 3：自监督一致性损失 （self-supervised consistency loss）。）

具体的说，是使用 Sim-Siam 自监督框架，它可以获取同一张图片的两个增强视图，并将第二个增强视图分支的输出向第一个视图的输出拉近。其中第一个分支是不反传梯度的编码器网络，第二个分支由相同但反传梯度编码器网络和 prediction head 组成，其中 prediction head 可以是简单的 MLP 网络。

原始的 Sim-Siam 自监督模型 [10] 仅能学习到单个图像的表征，但无法得知不同图像之间的关联，因此 Sim-Siam 对 transition 前后两帧的表征可能完全不同，不适用于学习 transition function。为了解决这一问题，作者改进了这个自监督模型，实现了端到端的训练，具体的逻辑可以参考图 3。由于训练的目标是找到相邻帧的转移关系，因此要将 $\hat{s}_{t+1}$ 拉近 s_{t+1} 。该模型首先从 o_t 中得到 s_t 的表征，再经过 transition function 转变为 $\hat{s}_{t+1}$ 。而另一个分支从 o_{t+1} 中得到 s_{t+1} 的表征，此时两个分支预测的为同一个实体。另外，两个分支预测的 s_{t+1} 都要经过一个共同的 projector 网络。由于 s_{t+1} 是从 o_{t+1} 中表示得到的，相对于 transition function 预测的 $\hat{s}_{t+1}$ 更为准确，因此以 o_{t+1} 的分支作为目标分支，该分支不回传梯度。

此外，作者也发现在 observation 上添加**数据增强**有助于提高学习到的表征的质量。除了让 $\hat{s}_{t+1}$ 和 s_{t+1} 靠近，作者也将 dynamic function 展开 5 步，让 $\hat{s}_{t+k}$ 和 s_{t+k} 接近 $(k = 1, \ldots, 5)$ 。

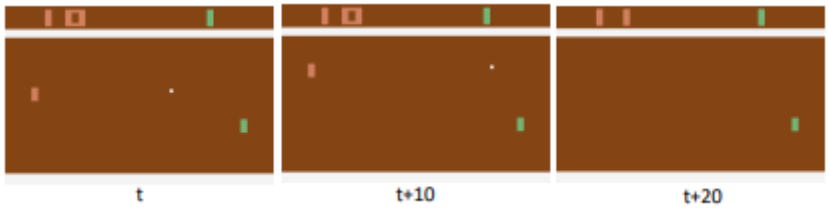
在实际实现中， $\hat{s}_{t+1}$ 和 s_{t+1} 之间的一致性损失定义为：左边分支 predictor 预测得到的 $\hat{z}_{t+1}$ 和右侧分支 stop-grad 的 projector 输出的 z_{t+1} 之间余弦相似度的负数。此时最小化一致性损失，即最大化两者的余弦相似度，其范围为[-1，1]：

$$cos\_\ similarity(z_{t+1}, \hat{z}_{t+1}) \ = \ \frac{z_{t+1} \cdot \hat{z}_{t+1}}{\| z_{t+1} \| \| \hat{z}_{t+1} \|}$$

3.3 端到端的 Value Prefix 预测

在基于模型的学习中，agent 只需要根据当前的状态，和想象执行的动作序列，就可以预测未来的状态。 这期间会遇到 state aliasing problem，即由于循环预测导致的累计误差，预测的时间越长，预测

的准确性就越低。这个问题可能导致次优的探索和次优动作选择 （sub-optimal exploration as well as sub-optimal action search）。



（图 4：Pong 中的轨迹样例。）

作者给出了一个例子： 如图 4 所示，右侧玩家未能接到这个球。但如果只看到 t 时刻的状态和未来的动作序列，我们很难预测出，是中间具体哪个时刻决策失误而导致了丢球这个结果。但是我们很容易发现，如果 t 时刻之后一段时间右边玩家都不动的话，就一定会丢球。因此，预测单一状态下的奖励是很困难的，但预测长期的奖励往往比较容易。根据这一发现，作者提出了**一种端到端的方式来预测 value prefix，即用 unrolled states 预测 value prefix。**

计算 UCT 中的 Q 值的时候，预测的 reward 总是通过如下方式被用到：

$$Q(s_t, a) \ = \ \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k \ v_{t+k}$$

因此我们可以把前面的 $\sum_{i=0}^{k-1} \gamma^i r_{t+i}$ 命名为 value prefix，由于它是在计算 Q 函数时候的前缀（prefix）。

例如，value prefix= $f(s_t, \hat{s}_{t+1}, \ldots \hat{s}_{t+k-1})$，其中 f 为神经网络，输入是当前状态和未来各个时刻的预测状态。在实验中，使用 LSTM 网络来预测时序信息。在实际训练时，因为每个新的状态输入时都需要重新计算 value prefix，因此 LSTM 网络在每个时间步都进行有监督训练。相比于简单的预测单步分数，这种 per-step rich supervision 预测 value prefix 的方法可以更好的避免 state aliasing problem ，让 MCTS 能够更好地探索，最终提高效果。

3.4 基于模型的 Off-Policy 修正

在传统的 MCTS 算法中，值函数估计是当前神经网络对应的 value 值。而在 MuZero Reanalyze 实际实现中，value target 是通过采样 replay buffer 中的轨迹得到的： $z_t \ = \ \sum_{i=0}^{k-1} \gamma^i u_{t+i} + \gamma^k \ v_{t+k}$ 。由于这一轨迹中 rollout 的 reward 是用旧的策略得到的，因此计算出的 value target 并不准确。当数据量有限时，计算 value target 还需要复用更久以前的 policy 采样得到的数据，这更加剧了 value target 不准确的问题。

因此作者提出了**基于模型的 Off-Policy 修正，基于已经学到的环境模型，进行“在线体验” （online experience），即从历史轨迹中使用奖励的视野是动态的（dynamic horizon）**。具体来说，当采用 k-step 的 value target 估计时，历史轨迹中 reward 的时间窗口 $l < k$，且采样的轨迹越古老，则 l 的长度越短。这通过减少 rollout 的步数减少 off-policy 带来的策略发散问题。

同时，作者还在最后一个状态 s_{t+l} 上用当前的策略重新进行一次 MCTS 搜索，并计算根节点的真实平均 value。这有效地修正了 off-policy 问题，也减少了设置 $l < k$ 带来的偏差。

> value target 由历史轨迹的真实奖励和在状态上 s_{t+l} 的 MCTS 估计值组成。其中历史轨迹是根据过去的策略得到的，导致了 off-policy 问题；而 MCTS 估计值和真实的奖励期望之间存在偏差过大的问题。因此用最新策略重新进行 MCTS 搜索能够减小偏差，而设置 $l < k$ 能够减少历史轨迹的影响，从而修正 off-policy 的问题。

最终， 目标的 value 可以表征为：

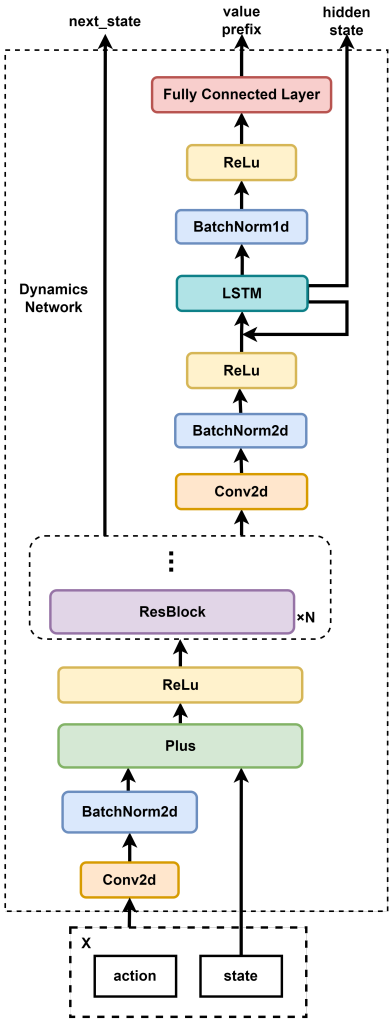
$$z_t = \sum_{i=0}^{l-1} \gamma^i u_{t+i} + \gamma^l v_{t+l}^{MCTS}$$

其中 v_{t+l}^{MCTS} 是用当前策略在 s_{t+l} 上扩展的 MCTS 树的 root value。事实上，这一修正的计算成本是 reanalyze 计算侧的两倍，但由于训练是并行的，因此并不会受到影响。

4. 实验

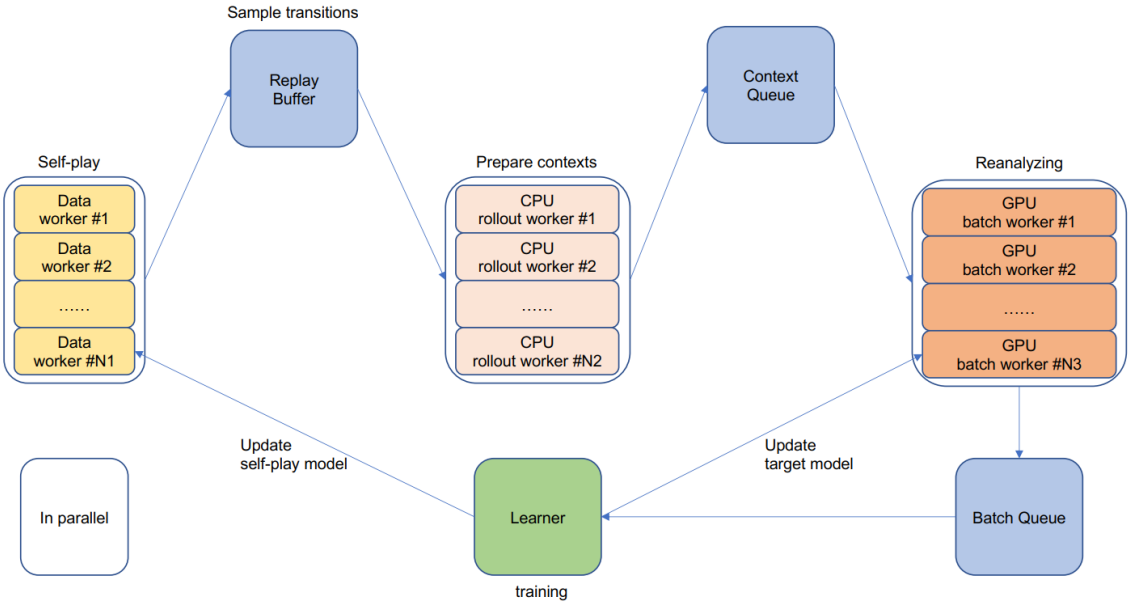
4.1 网络结构及具体实现

- representation network 使用和 MuZero 相同的架构。
- dynamics network 相对于 MuZero，其输出 reward 的 head 变为输出 value prefix。且网络中加入了 LSTM ，其隐藏状态作为 hidden state 输出。
- prediction network 使用和 MuZero 相同的架构。

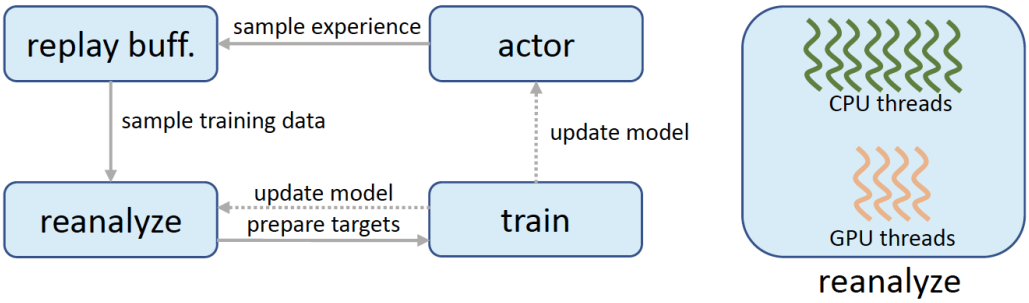


（图 5：EfficientZero 的动力学网络结构图。）

EfficientZero 的具体实现采用了并行的方式，其具体 pipeline 如图 6 和图 7 所示。首先，最左侧的 worker 会通过 600 步的 self-play 收集数据，并放到 replay buffer 中。中间的 rollout worker 从 replay buffer 中采集轨迹并拆分为 batch transition，将其中的 context 交给最右侧的 batch worker，这一步只需要 CPU 参与。而 batch worker 用 GPU 对 batch transition 进行 reanalyze 后，将数据交由 learner 学习。由于这三个工作流程分别使用到 self-play，CPU 和 GPU，因此完全可以并行进行。



（图 6：EfficientZero 具体实现的 pipeline。）



（图 7：EfficientZero 具体实现的概览图。）

4.2 Atari 100k Benchmark 上的基线比较结果

此处 human normalized score 定义为：

$$\frac{score_{agent} - score_{random}}{score_{human} - score_{random}}$$

EfficientZero 与 基线算法比较结果如下：

Game	Random	Human	SimPLe	OTRainbow	CURL	DrQ	SPR	MuZero	Ours
Alien	227.8	7127.7	616.9	**824.7**	558.2	771.2	801.5	530.0	808.5
Amidar	5.8	1719.5	88.0	82.8	142.1	102.8	**176.3**	38.8	148.6
Assault	222.4	742.0	527.2	351.9	600.6	452.4	571.0	500.1	**1263.1**
Asterix	210.0	8503.3	1128.3	628.5	734.5	603.5	977.8	1734.0	**25557.8**
Bank Heist	14.2	753.1	34.2	182.1	131.6	168.9	**380.9**	192.5	351.0
BattleZone	2360.0	37187.5	5184.4	4060.6	14870.0	12954.0	**16651.0**	7687.5	13871.2
Boxing	0.1	12.1	9.1	2.5	1.2	6.0	35.8	15.1	**52.7**
Breakout	1.7	30.5	16.4	9.8	4.9	16.1	17.1	48.0	**414.1**
ChopperCmd	811.0	7387.8	1246.9	1033.3	1058.5	780.3	974.8	**1350.0**	1117.3
Crazy Climber	10780.5	35829.4	62583.6	21327.8	12146.5	20516.5	42923.6	56937.0	**83940.2**
Demon Attack	152.1	1971.0	208.1	711.8	817.6	1113.4	545.2	3527.0	**13003.9**
Freeway	0.0	29.6	20.3	25.0	**26.7**	9.8	24.4	21.8	21.8
Frostbite	65.2	4334.7	254.7	231.6	1181.3	331.1	**1821.5**	255.0	296.3
Gopher	257.6	2412.5	771.0	778.0	669.3	636.3	715.2	1256.0	**3260.3**
Hero	1027.0	30826.4	2656.6	6458.8	6279.3	3736.3	7019.2	3095.0	**9315.9**
Jamesbond	29.0	302.8	125.3	112.3	471.0	236.0	365.4	87.5	**517.0**
Kangaroo	52.0	3035.0	323.1	605.4	872.5	940.6	**3276.4**	62.5	724.1
Krull	1598.0	2665.5	4539.9	3277.9	4229.6	4018.1	3688.9	4890.8	**5663.3**
Kung Fu Master	258.5	22736.3	17257.2	5722.2	14307.8	9111.0	13192.7	18813.0	**30944.8**
Ms Pacman	307.3	6951.6	**1480.0**	941.9	1465.5	960.5	1313.2	1265.6	1281.2
Pong	-20.7	14.6	12.8	1.3	-16.5	-8.5	-5.9	-6.7	**20.1**
Private Eye	24.9	69571.3	58.3	100.0	**218.4**	-13.6	124.0	56.3	96.7
Qbert	163.9	13455.0	1288.8	509.3	1042.4	854.4	669.1	3952.0	**13781.9**
Road Runner	11.5	7845.0	5640.6	2696.7	5661.0	8895.1	14220.5	2500.0	**17751.3**
Seaquest	68.4	42054.7	683.3	286.9	384.5	301.2	583.1	208.0	**1100.2**
Up N Down	533.4	11693.2	3350.3	2847.6	2955.2	3180.8	**28138.5**	2896.9	17264.2
Normed Mean	0.000	1.000	0.443	0.264	0.381	0.357	0.704	0.562	**1.943**
Normed Median	0.000	1.000	0.144	0.204	0.175	0.268	0.415	0.227	**1.090**

（表 1：EfficientZero 在 Atari 100k benchmark 上与基线算法的比较。）

在根据人类水平分数进行标准化后，EfficientZero 的平均得分达到了 1.943，中位数得分达到了 1.090。作为比较，DQN 达到了 2.20 的平均得分和 0.959 的中位数得分，然而需要 500 倍于 EfficientZero 的数据量（2 亿帧）。这是第一个只需要 2 小时游戏数据就能超过人类水平的模型，在 26 个游戏上有 14 个游戏优于人类。先对于当前最优算法 SPR，EfficientZero 的平均得分和中位数得分分别提高了 170%和 180%。

4.3 DMControl 100k Benchmark 上的基线比较结果

EfficientZero 在模拟任务中也取得了显著的效果。如表 2 所示，EfficientZero 相对于当前最优算法 CURL 有明显的性能提升。此外，EfficientZero 也达到了和 state SAC 相当的水平，而 state SAC 由于使用的是真实 state，因此被视作 oracles。

Task	CURL	Dreamer	MuZero	SAC-AE	Pixel SAC	State SAC	EfficientZero
Cartpole, Swingup	582± 146	326±27	218.5 ± 122	311±11	419±40	835±22	**813±19**
Reacher, Easy	538± 233	314±155	493 ± 145	274±14	145±30	746±25	**952±34**
Ball in cup, Catch	769± 43	246 ± 174	542 ± 270	391± 82	312± 63	746±91	**942±17**

（表 2：EfficientZero 在 DMControl 100k benchmark 上和基线模型的比较结果。）

4.4 三个改进的 Ablation Study

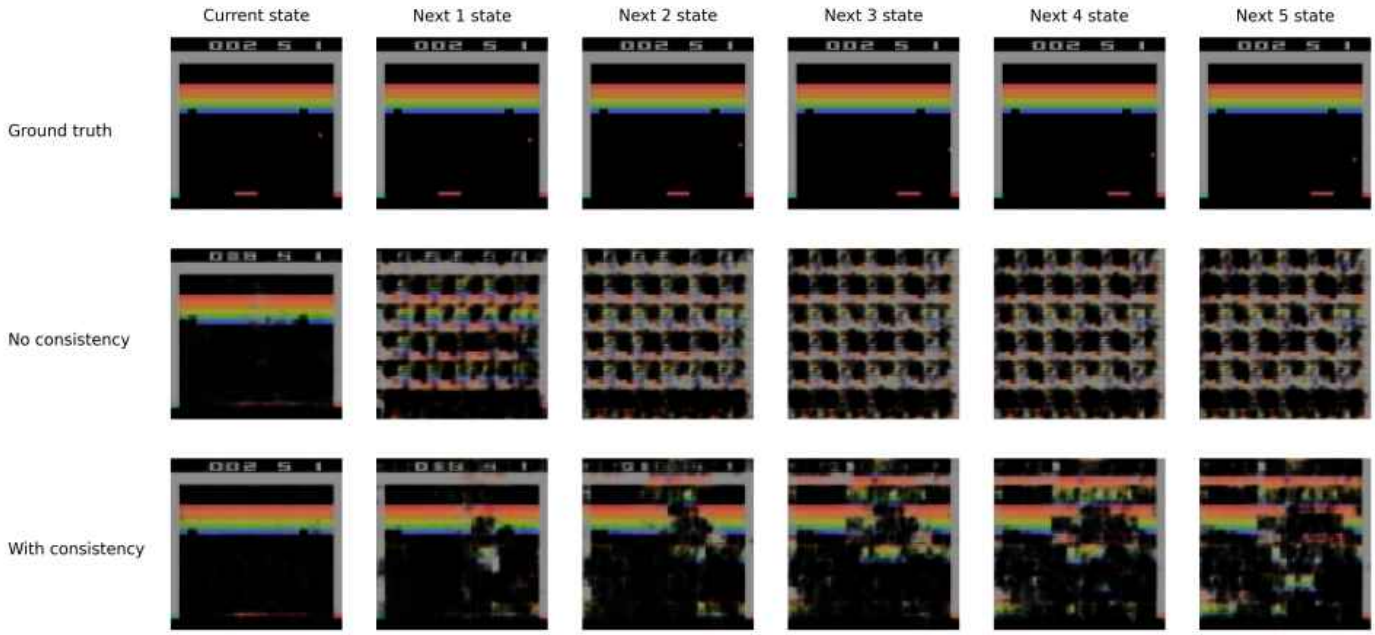
首先，作者从模型中分别去掉这三种改进，并在 Atari 上实验效果如下：

Game	Full	w.o. consistency	w.o. value prefix	w.o. off-policy correction
Normed Mean	**1.943**	0.881	1.482	1.475
Normed Median	**1.090**	0.340	0.552	0.836

（表 3：EfficientZero 中去除三种模块后在 Atari 上的性能效果。）

自监督一致性损失对图像重建的影响

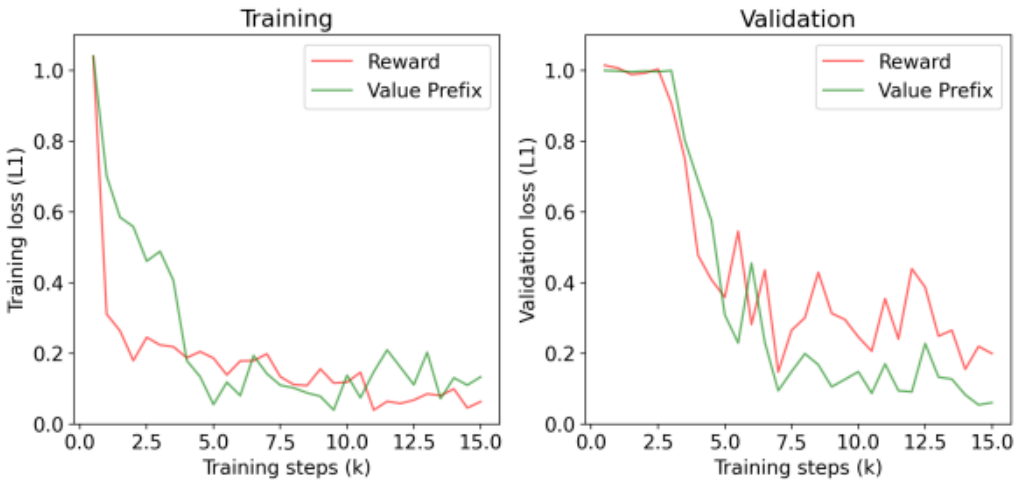
为了探究自监督一致性损失这一模块带来的影响，作者设计了解码器，从 latent state 中重建 observation。从结果中观察到，当没有一致性损失时，representation network 分支的 latent states 和 dynamics function 分支的 state 中存在一些分布的偏移，而一致性损失的加入可以减少偏移，提供更多监督信号。



（图 8：自监督一致性损失对图像重建的影响。）

Value Prefix 在 Training 和 Validation 过程中对 Loss 的影响

作者在同一个训练集和验证集上进行实验，发现不使用 value prefix 在 training 时能达到更低的 loss，但在 validation 时，使用 value prefix 的 loss 明显更低。这说明 value prefix 可以避免奖励预测的过拟合，从而避免 state aliasing problem。



（图 9：使用 value prefix 和不使用 value prefix 在训练和验证时的损失变化趋势。）

Off-Policy 修正对未来状态预测 L1 Error 的影响

为了验证 off-policy 修正的效果，作者比较了 target value 和 ground truth value 之间的误差。以游戏 UpNDown 为例，玩家需要操控一辆车在两条交错的车道上行驶和跳跃，当跳跃到其他车上或吃到旗帜时可以得分，当被其他车撞到时失去一条生命。在该环境中，无论是在当前状态还是 unrolled 状态下，有 off-policy 修正都会降低 error。

States	Current state	Unrolled next 5 states (Avg.)	All states (Avg.)
Value error without correction	0.765	0.636	0.657
Value error with correction	**0.533**	**0.576**	**0.569**

（表 4：UpNDown 上 target value 和 ground truth value 之间的 L1 误差。）

作者还比较了在不同的游戏阶段上（20k，...，100k）的 value error。发现当轨迹越新时，value error 也越小，这说明 off-policy 问题是因为数据的滞后性导致的。

Stages of trajectories	20k	40k	60k	80k	100k
Value error without correction	0.657	0.697	0.628	0.574	0.441
Value error with correction	**0.569**	**0.552**	**0.537**	**0.488**	**0.397**

（表 5：UpNDown 上不同阶段的平均 L1 误差。）

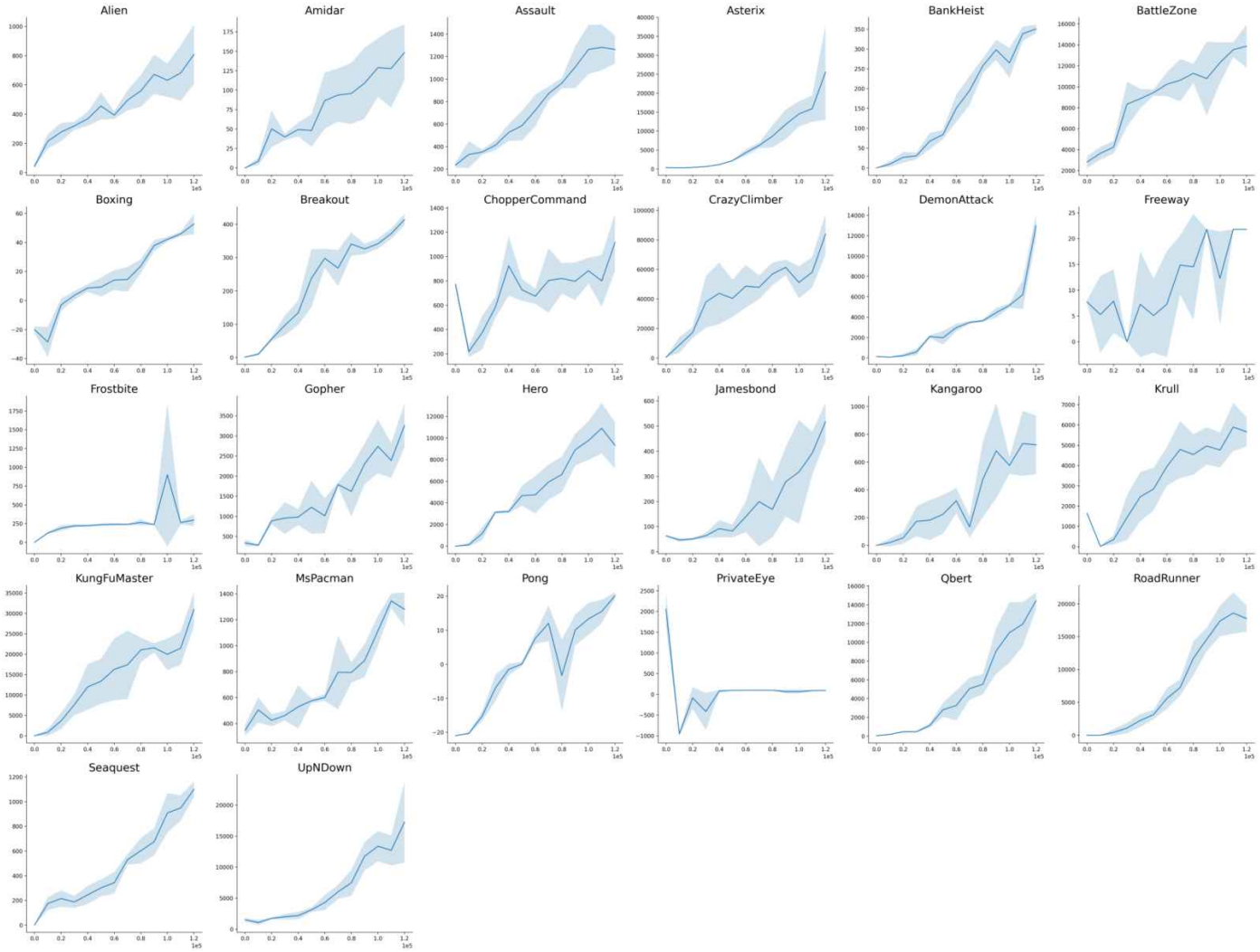
4.5 EfficientZero 在 Atari 上的超参数与性能

在 Atari 的 26 个游戏上，EfficientZero 以如下超参数进行训练：

Parameter	Setting
Observation down-sampling	96×96
Frames stacked	4
Frames skip	4
Reward clipping	True
Terminal on loss of life	True
Max frames per episode	108K
Discount factor	0.997^4
Minibatch size	256
Optimizer	SGD
Optimizer: learning rate	0.2
Optimizer: momentum	0.9
Optimizer: weight decay (c)	0.0001
Learning rate schedule	$0.2 \rightarrow 0.02$
Max gradient norm	5
Priority exponent (α)	0.6
Priority correction (β)	$0.4 \rightarrow 1$
Training steps	120K
Evaluation episodes	32
Min replay size for sampling	2000
Self-play network updating inerval	100
Target network updating interval	200
Unroll steps (l_{unroll})	5
TD steps (k)	5
Policy loss coefficient (λ_1)	1
Value loss coefficient (λ_2)	0.25
Self-supervised consistency loss coefficient (λ_3)	2
LSTM horizontal length (ζ)	5
Dirichlet noise ratio (ξ)	0.3
Number of simulations in MCTS (N_{sim})	50
Reanalyzed policy ratio	0.99

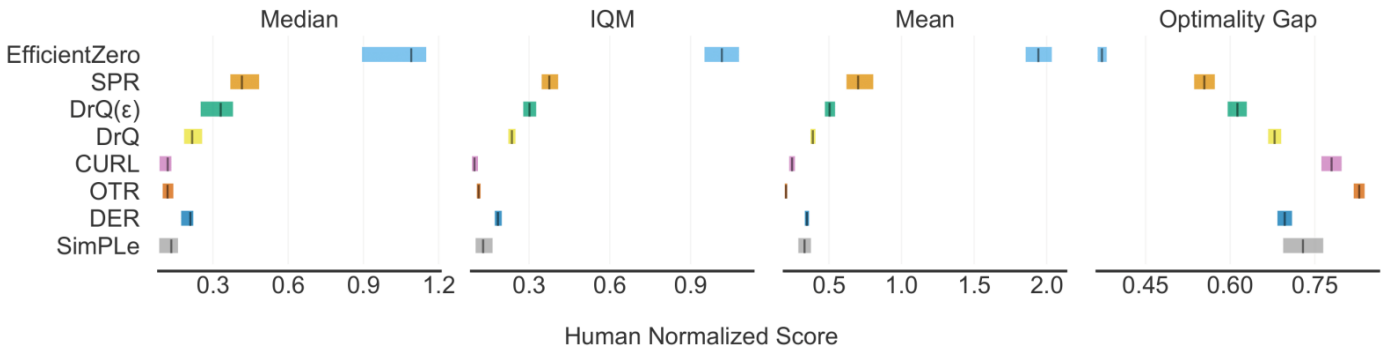
（表 6：EfficientZero 在 Atari 游戏上训练的参数设定。）

得到的验证曲线如下：



（图 10：EfficientZero 在 Atari 上验证时的奖励曲线。）

此外，由于使用统计工具能够得到更鲁棒和有效的指标，作者在图 11 中展示了基于 Agarwal 的开源代码得到的置信区间结果。可以看到，无论用哪种指标进行比较，EfiicientZero 都明显优于其他方法。



（图 11： EfficientZero 及基线模型在 Atari 上的 95%置信区间。）

5. 总结与展望

总的来说，EfficientZero 是将 MuZero 扩展到了复杂感知输入领域，在 MuZero 的基础上提出了三种改进：自监督一致性损失、端到端的 value prefix 和 off-policy 修正，并分别设计 ablation 实验说明了这三种改进的有效性：

- 样本利用效率较高，在少量样本下就能达到与先前算法相当甚至更好的效果。
- 用 value prefix 代替 reward，从预测单个时间点的奖励变为预测长期奖励，在数据集规模较小时，减少对奖励预测的过拟合。
- 通过在历史轨迹中使用动态长度的经验，并用当前策略获得最新经验，修正了 off-policy 问题和 MCTS 的估计偏差。

未来的研究方向包括：

- 如何将 EfficientZero 延伸到其他类型场景上，例如连续动作空间。
- 如何进行 MCTS 的加速。
- 如何将这一框架和 life-long study 进行结合。

6. 参考文献

[1] Schwarzer M, Anand A, Goel R, et al. Data-efficient reinforcement learning with self-predictive representations[J]. arXiv preprint arXiv:2007.05929, 2020.

[2] Hafner D, Lillicrap T, Fischer I, et al. Learning latent dynamics for planning from pixels[C]//International conference on machine learning. PMLR, 2019: 2555-2565.

[3] Kielak K P. Do recent advancements in model-based deep reinforcement learning really improve data efficiency?[J]. 2020.

[4] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.

[5] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge[J]. nature, 2017, 550(7676): 354-359.

[6] Hubert T, Schrittwieser J, Antonoglou I, et al. Learning and planning in complex action spaces[C]//International Conference on Machine Learning. PMLR, 2021: 4476-4486.

[7] Schrittwieser J, Antonoglou I, Hubert T, et al. Mastering atari, go, chess and shogi by planning with a learned model[J]. Nature, 2020, 588(7839): 604-609.

[8] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[9] Hessel M, Modayil J, Van Hasselt H, et al. Rainbow: Combining improvements in deep reinforcement learning[C]//Proceedings of the AAAI conference on artificial intelligence. 2018, 32(1).

[10] Chen X, He K. Exploring simple siamese representation learning[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021: 15750-15758.