Mastering Atari Games with Limited Data

Anonymous Author(s)
Affiliation
Address
email

Abstract

Reinforcement learning has achieved great success in many applications. However, sample efficiency remains a key challenge, with prominent methods requiring millions (or even billions) of environment steps to train. Recently, there has been significant progress in sample-efficient image-based RL algorithms; however, consistent human-level performance on the Atari game benchmark remains an elusive goal. We propose a sample efficient model-based visual RL algorithm built on MuZero, which we name EfficientZero. Our method achieves 180% mean human performance, or 84.8% median performance on the Atari 100k benchmark with only two hours of real-time game experience. This is the first time an algorithm achieves super-human mean performance on Atari games with such little data. EfficientZero's performance is also close to DQN's performance at 200 million frames, however we consumes 500 times less data. EfficientZero's low sample complexity *and* high performance can bring RL closer to real-world applicability. Unlike previous MCTS-based RL algorithm, our algorithm can run on a single machine instead of requiring a distributed system. We implement our algorithm in an easy to understand manner, and will open-source it upon publication. We hope it will accelerate the research of MCTS-based RL algorithms in the wider community.

1 Introduction

Reinforcement learning has achieved great success on many challenging problems. Notable work includes DQN [26], AlphaGo [36] and OpenAI Five [5]. However, most of these works come at the cost of a large amount of environment interactions. For example, AlphaZero needs to play 21 million games at training time. On the contrary, a professional human player can only play around 5 games per day, meaning it would take a human player 11,500 years to achieve the same amount of experience. The sample complexity might be less of an issue when applying RL algorithms in simulation and games. However, when it comes to real world problems, such as robotic manipulation, healthcare and advertisement recommendation systems, achieving high performance while maintaining low sample complexity is key to viability.

People have made a lot of progress in sample efficient RL in the past years [8, 10, 38, 24, 22, 35, 19]. Among them, model-based methods have attracted a lot of attentions, since both the data from real environments and the "imagined data" from the model can be used to train the policy, making these methods particularly sample-efficient [8, 10]. However, most of the successes are in state-based environments. In image-based environments, some model-based methods such as Muzero [30] and Dreamer V2 [15] achieves super-human performance, but they are not sample efficient; other methods such as SimPLe [19] is quite efficient but achieve inferior performance (0.144 human normalized median score). Recently, data augmentation and self-supervised methods applied to model-free methods have achieved more success in the data-efficient regime [35]. However, they still fail to achieve the levels which can be expected of a human.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Therefore, for improving the sample efficiency as well as keeping super performance, we think the following three components are essential to the image input model-based RL agent: a self-supervised environment model, a mechanism to alleviate the model compounding error and an efficient use of the model to explore the world. For example, the environment model in MuZero is only supervised by the scalar reward and value, thus leading to a sub-optimal environment model when data is limited. In this work, we propose EfficientZero, a model-based RL algorithm that can achieve high performance with limited data. Our proposed method is building upon MuZero. We make three intuitive but critical changes: (1) use self-supervised learning to learn a temporally consistent environment model in a sample efficient manner (2) propose to learn the *value prefix* in an end-to-end manner, thus helping to alleviate the compounding error in the model (3) use the learned model to correct off-policy value targets.

Our model achieves state-of-the-art performance on the widely used Atari [4] 100k benchmarks and it achieves human-level performance with only 2 hours of real-time game play. More specifically, our model achieves 180% mean human normalized performance, and 84.8% median human normalized performance. As a reference, DQN [26] achieves 220% mean human normalized performance, and 96% median human normalized performance, at the cost of 500 times more data (200 million frames). Our sample-efficient and high performance algorithm opens the possibility of having more impact on many real-world problems.

2 Related Work

2.1 Sample Efficient Reinforcement Learning

Sample efficiency has attracted significant work in the past. In RL with image inputs, model-based approaches [14, 13] which model the world with both a stochastic and a deterministic component, have achieved promising results for simulated robotic control. Kaiser et al. [19] proposes to use an action-conditioned video prediction model, along with a policy learning algorithm on top of the video prediction model. It achieves the first strong result in Atari games with as little as 400k frames. However, Kielak [20], van Hasselt et al. [42] argue that this is not necessary to achieve strong results with model-based methods, and they show that when tuned appropriately, Rainbow [17] can achieve comparable performance.

Recent advances in self-supervised learning, such as SimCLR [6], MoCo [16], SimSiam [7] and BYOL [12] have inspired representation learning in image-based RL. Srinivas et al. [38] proposes to use contrastive learning in RL algorithms and shows that it can achieve strong performance on image-based continuous and discrete control tasks. Later, Laskin et al. [24], Kostrikov et al. [22] find that the contrastive learning part is not necessary, but with the data augmentations alone, they can achieve even better performance. Schwarzer et al. [35] propose a temporal consistency loss, which is combined with the data augmentation, achieves state-of-the-art performance. We note that our self-supervised consistency loss is quite similar to Schwarzer et al. [35], except we use SimSiam [6] while they use BYOL [12] as the base unsupervised learning framework. However, Schwarzer et al. [35] only apply the learned representation in a model-free manner, while we combine the learned model with model-based exploration and policy improvement, thus leading to more efficient use of the environment model.

Despite the recent progress in the sample-efficient RL, today's RL algorithms are still well behind human performance when the amount of data is limited. Although traditional model-based RL is considered more sample efficient than model-free ones, current model-free methods dominate in terms of performance for image-input settings. In this paper, we propose a model-based RL algorithm that for the first time, achieves human-level performance on Atari games with limited data.

2.2 Reinforcement Learning with MCTS

Temporal difference learning [26, 41, 43, 17] and policy gradient based methods [27, 25, 32, 34] are two types of popular reinforcement learning algorithms. Recently, Silver et al. [36] propose to use MCTS as a policy improvement operator and has achieved great success in many board games, such as Go, Chess and Shogi [37, 37]. Later, the algorithm is adapted to learn the world model at the same time [30]. It has also been extended to deal with continuous action spaces [18] and offline data [31]. These MCTS RL algorithms are a hybrid of model-based learning and model-free learning.

However, most of them are trained with a lot of environment samples. Our method is built on top of MuZero [30], and we demonstrate that our method can achieve higher sample efficiency while still achieving competitive performance on the Atari 100k benchmark. de Vries et al. [9] have studied the potential of using auxiliary loss similar to our self-supervised consistency loss. However, they only test on two low dimensional state-based environments, and find the auxiliary loss has mixed effects on the performance. On the contrary, we find that the consistency loss is critical on most of the environments with high dimensional observations and limited data.

2.3 Multi-Step Value Estimation

In Q-learning [44], the target Q value is computed by one step backup. In practice, people find that incorporating multiple steps of rewards at once, i.e. $\sum_{i=0}^{k-1} \gamma^i r_i + \gamma^k \tilde{\mathcal{V}}(s_k)$, where $\tilde{\mathcal{V}}$ is the value estimation from the target network, to compute the value target leads to faster convergence [26, 17]. However, the use of multi-step value has off-policy issues, since $r_2, \cdots, r_{k-1}$ are not generated by the current policy. In practice, this issue is usually ignored when there is a large amount of data, since the data can be thought as approximately on-policy. TD(λ) [39] and GAE [33] improve the value estimation by better trading off the bias and the variance, but they do not deal with the off-policy issue. Recent image input model-based algorithms such as Kaiser et al. [19] and Hafner et al. [13] use model imaginary rollouts to avoid the off-policy issue. However, this approach has the risk of model exploitation. Our proposed model-based off-policy correction method starts from the rewards in the real-world experience, and use model-based value estimate to bootstrap. Our approach balances between the off-policy issue and model-exploitation.

3 Background

3.1 MuZero

Our method builds on top of the MuZero Reanalyze [30] algorithm. For brevity, we refer to it as MuZero throughout the paper. MuZero is a policy learning method based on the Monte-Carlo Tree Search (MCTS) algorithm. The MCTS algorithm operates with an environment model, a prior policy function and a value function. The environment model is represented as the reward function $\mathcal{R}$ and the dynamic function $\mathcal{G}$: $\hat{r}_t = \mathcal{R}(s_t, a_t)$, $\hat{s}_{t+1} = \mathcal{G}(s_t, a_t)$, which are needed when MCTS expands a new node. In MuZero, the environment model is learned. Thus the reward and the next state are approximated. Besides, the policy function $\pi(s_t)$ acts as a search prior over actions of a node. It helps the MCTS focus on more promising actions when expanding the node. MCTS also needs a value function $\mathcal{V}(s_t)$ that measures the expected return of the node s_t, which provides a long-term evaluation of the tree's leaf node without further search. MCTS will output an action visit distribution $\pi'(s_t)$ over the root node, which is potentially a better policy, compared to the current neural network. Thus, the MCTS algorithm can be thought of as a policy improvement operator.

In practice, the environment model, policy function and value function operate on a hidden abstract state s_t, both for computational efficiency and ease of environment modeling. The abstract state is extracted by a representation function $\mathcal{H}$ on observations o_t: $s_t = \mathcal{H}(o_t)$. All of the above mentioned models are usually represented as neural networks. During training, the algorithm collects roll-out data in the environment using MCTS, resulting in potentially higher quality data than the current neural network policy. The data is stored in a replay buffer. The optimizer minimizes the following loss on the data sampled from the replay buffer:

$$\mathcal{L}(r_t, \mathcal{R}(s_t, a_t)) + \lambda_1 \mathcal{L}(\pi'(s_t), \pi(s_t)) + \lambda_2 \mathcal{L}(\mathcal{V}'(s_t), \mathcal{V}(s_t)) \tag{1}$$

Here, $\mathcal{R}(s_t, a_t)$, $\pi(s_t)$ and $\mathcal{V}(s_t)$ as well as the representation function $s_t = \mathcal{H}(o_t)$ are trainable neural networks. r_t, $\pi'(s_t)$ and $\mathcal{V}'(s_t)$ are training targets. r_t is the reward observed in the environment rollout with action a_t at state s_t. $\pi'(s_t)$ is the output visit count distribution of the MCTS. $\mathcal{V}'(s_t) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k \tilde{\mathcal{V}}^(s_{t+k})$, where γ is the discount factor, r_{t+i} is the reward from the environment and $\tilde{\mathcal{V}}(s_{t+k})$ is a bootstrapped value computed by a target network.

It is worth to note that MuZero does not explicitly learn the environment model. Instead, it solely relies on the reward and value prediction to learn the model.

3.2 Monte-Carlo Tree Search

Monte-Carlo Tree Search [1, 36, 37, 15], or MCTS, is a heuristic search algorithm. In our setup, MCTS is used to find an action policy that is better than the current neural network policy.

More specifically, MCTS needs an environment model, including the reward function and the next-state function. It also needs a value function and a policy function, which act as heuristics for the tree search. MCTS operates by expanding a search tree from the current node. It saves computation by selectively expanding few nodes. In order to find a high quality decision, the tree expansion process has to balance between exploration versus exploitation, i.e. balance between expanding a node that is promising with many visits versus expanding a node with lower performance but less visits. MCTS employs the UCT [29, 21] rule, i.e. UCB [2] on trees. At every node expansion step, UCT will select a node as follows [15]:

$$a^k = \arg\max_a Q(s,a) + P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}\left(c_1 + \log\left(\frac{\sum_b N(s,b) + c_2 + 1}{c_2}\right)\right) \quad (2)$$

where, $Q(s,a)$ is the current estimate of the Q-value, $P(s,a)$ is the current neural network policy for selecting this action, helping the MCTS prioritize explore promising part of the tree. During training time, $P(s,a)$ is usually perturbed by noises to allow explorations. $N(s,a)$ denotes how many times this state-action pair is visited in the tree search, and $N(s,b)$ denote that of a's siblings. Thus this term will encourage the search to visit the nodes whose siblings are visited often, but itself less visited. Finally, the last term gives a weights to the previous terms. Usually, $c_1 = 1.25$ and $c_2 = 19652$ according to the literature [36, 37, 15].

After expanding the nodes for a pre-defined number of times, the MCTS will return how many times each action under the root node is visited, as the improved policy to the root node. Thus, MCTS can be thought as a policy improvement operator in the RL setting.

4 EfficientZero

Model-based algorithms have achieved great success in sample-efficient learning from low-dimensional states. However, current visual model-based algorithms either require large amounts of training data, or exhibit inferior performance to model-free algorithms in data-limited settings [35]. Many previous works even suspect whether model-based algorithm can really offer data-efficiency when using image observations [42]. We provide a positive answer here. We propose the EfficientZero algorithm, a model-based algorithm builds on top of the MCTS RL algorithm, that achieves human-level performance on the 100k Atari benchmark, outperforming all previous methods by at least 100% in terms of median performance.

When directly running MCTS-based RL algorithms such as MuZero, we find that they do not perform well on the limited-data benchmark. Through our ablations, we confirm the following three issues which pose challenges to algorithms like MuZero in data-limited settings.

Lack of supervision on environment model. First, the learned model in the environment dynamics is only trained through the reward and the value function. However, the reward is only a scalar signal and in many scenarios, the reward will be sparse. Value functions are trained with bootstrapping, and thus are very noisy. Either the reward or the value provide enough training signal to learn the environment model.

Hardness to deal with aleatory uncertainty. Second, we find that even with enough data, the predicted reward still have large prediction errors. This is caused by the aleatory uncertainty of the underlying environment. For example, the environment is hard to model. The reward prediction errors will accumulate when expanding the MCTS tree to a large depth, resulting in sub-optimal performance in exploration and evaluation.

Off-policy of multi-step value. Lastly, when computing the value target, MuZero uses the multi-step reward observed in the environment. Although this allows the reward to be propagated to the value function faster, we find that it suffers from severe off-policy issues and hinders convergence in the limited data scenario.

To address the above issues, we propose the following three critical modifications, which can greatly improve performance when samples are limited.

4.1 Self-Supervised Consistency Loss

In previous MCTS RL algorithms, the environment model is either given or only trained with rewards and values, but neither reward nor value provides sufficient training signal due to their scalar nature. The problem is more severe when reward is sparse or the bootstrapped value is not accurate. The MCTS policy improvement operator heavily relies on the environment model. Thus, it is vital to have an accurate one.

We notice that the output $\hat{s}_{t+1}$ from the dynamic function $\mathcal{G}$ should be the same as s_{t+1}, i.e. the output of the representation function $\mathcal{H}$ with input of the next observation o_{t+1} (Fig. 1). This can help to supervise the predicted next state $\hat{s}_{t+1}$ using the actual s_{t+1}, which is a tensor with at least a few hundred dimensions. This provides $\hat{s}_{t+1}$ with much more training signals than the default scalar reward and value.

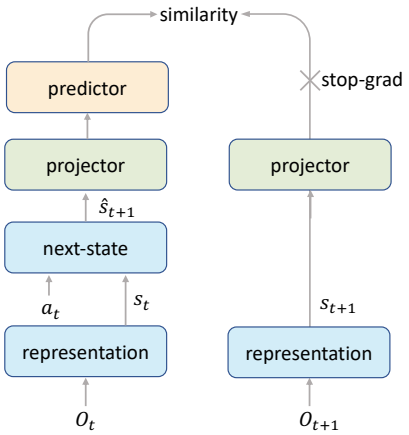

Figure 1: The self-supervised consistency loss.

More specifically, we adopt the recently proposed SimSiam [7] self-supervised framework. SimSiam [7] is a self-supervised method that takes two augmentation views of the same image and pulls the output of the second branch close to that of the first branch, where the first branch is an encoder network without gradient, and the second branch is the same encoder network with the gradient and a predictor head. The predictor head can simply be a two-layer MLP.

Note that SimSiam only learns the representation of individual images, and is not aware of how different images are connected. The learned image representations of SimSiam might not be a good candidate for learning the environment transition function, since adjacent observations might be encoded to very different representation encodings. We propose a self-supervised method that learns the transition function, along with the image representation function in an end-to-end manner. Figure 1 shows our method. Since we aim to learn the transition between adjacent observations, we pull o_t and o_{t+1} close to each other. The transition function is applied after the representation of o_t, such that s_t is transformed to $\hat{s}_{t+1}$, which now represents the same entity as the other branch. Then both of s_{t+1} and $\hat{s}_{t+1}$ go through a common projector network. Since s_{t+1} is potentially a more accurate description of o_{t+1} compared to $\hat{s}_{t+1}$, we make the o_{t+1} branch as the target branch.

The two adjacent observations provide natural two views of the same entity. In practice, we find that applying augmentations to observations such as random small shift of 0-4 pixels on the image helps to further improve the learned representation quality [38, 35]. We also unroll the dynamic function recurrently and also pull $\hat{s}_{t+k}$ close to s_{t+k}. Please see the appendix for more implementation details.

4.2 End-To-End Prediction of the Value Prefix

In model-based learning, the agent needs to predict the future states conditioned on the current state and a series of hypothetical actions. The longer the prediction, the harder to predict it accurately, due to the compounding error in the recurrent rollouts. This is called the state aliasing problem. The environment model plays an important roll in MCTS. The state aliasing problem has negative effect on MCTS expansion, which will result in sub-optimal exploration as well as sub-optimal action search.

Predicting the reward from an aliased state is a hard problem. For example, as shown in Figure 2, the right agent loses the ball. If we only see the first observation, along with future actions, it is very hard both for an agent and a human to predict at which *exact* future timestep the player would lose a point. However, it is easy to predict the agent will miss the ball after a sufficient number of timesteps if he does not

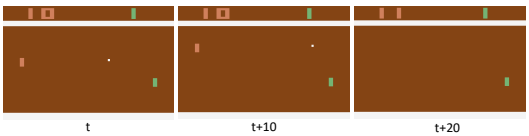

Figure 2: A sample trajectory from the Atari Pong game. In this case, the right player didn't move and miss the ball.

move. In practice, a human will never try to predict the exact step that he loses the point, but will imagine over a longer horizon and thus get a more confident prediction.

Inspired by this intuition, we propose an end-to-end method to predict the *value prefix*. We notice that the predicted reward is always used in the following manner:

$$Q(s,a) = \sum_{i=0}^{k-1} \gamma^i \hat{r}_{t+i} + \gamma^k \tilde{\mathcal{V}}(\hat{s}_{t+k})$$

where $\hat{r}_{t+i}$ is the reward predicted from unrolled state $\hat{s}_{t+i}$. We name the sum of rewards $\sum_{i=0}^{k-1} \gamma^i \hat{r}_{t+i}$ as the *value prefix*, since it is used as a prefix in the later Q-value computation.

We propose to predict value prefix from the unrolled states $(s_t, \hat{s}_{t+1}, \cdots, \hat{s}_{t+k-1})$ in an end-to-end manner, i.e. value-prefix $= f(s_t, \hat{s}_{t+1}, \cdots, \hat{s}_{t+k-1})$. Here f is some neural network architecture that takes in variable number of inputs and output a scalar. We choose the LSTM in our experiment. During the training time, the LSTM is supervised at every time step, since the value prefix can be computed whenever a new state comes in. This per-step rich supervision allows the LSTM can be trained well even with limited data. Compared with the naive per step reward prediction and summation approach, the end-to-end value prefix prediction is more accurate, because it can automatically handles the intermediate state aliasing problem. See Experiment Section 6.5 for empirical evaluations. As a result, it helps the MCTS to explore better, and thus increases the performance. See appendix for architectural details.

4.3 Model-Based Off-Policy Correction

In MCTS RL algorithms, the value function fits the value of the current neural network policy. However, in practice, the value target is computed by sampling a trajectory from the replay buffer and computing: $\mathcal{V}'(s_t) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k \tilde{\mathcal{V}}(\mathrm{s}_{t+k})$, where r_{t+i} is the reward from replayed trajectory. This value target suffers from off-policy issues, since the trajectory is rolled out using an older policy, and thus the value target is no longer accurate. When data is limited, we have to reuse the data sampled from a much older policy, thus exaggerating the inaccurate value target issue.

In previous model-free settings, there is no straightforward approach to fix this issue. On the contrary, since we have a model of the environment, we can use the model to imagine "online experience". More specifically, we propose to use l rewards from the old trajectory, where $l < k$ and l should be smaller if the trajectory is older. This reduces the policy divergence by fewer number of rollout steps. Further, we redo a MCTS search with the current policy on the last state s_{t+l}, and compute the empirical mean value at the root node. This effectively corrects the off policy issue using imagined rollouts with current policy, and reduces the increased bias caused by setting l less than k. Formally, we propose to use the following value target:

$$\mathcal{V}'(s_t) = \sum_{i=0}^{l-1} \gamma^i r_{t+i} + \gamma^l \mathcal{V}^{\mathrm{MCTS}}(\mathrm{s}_{t+l}) \tag{3}$$

where $l <= k$ and the older the sampled trajectory, the smaller the l. $\mathcal{V}^{\mathrm{MCTS}}(\mathrm{s}_{t+l})$ is the root value of the MCTS tree expanded from s_{t+l} with the current policy. See appendix for hyper-parameters on how to choose l.

5 Open Source EfficientZero Implementation

MCTS-based RL algorithms present a promising future research direction: to achieve strong performance with model-based methods. However, currently there are two major practical obstacles that prevents them from being widely used. First, there are no high quality open source implementations of these algorithms. Existing implementations [45, 23] can only deal with simple state-based

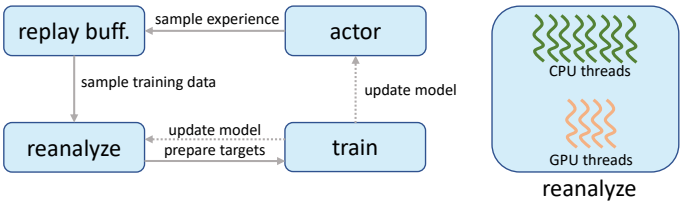

Figure 3: EfficientZero implementation overview.

environments, such as CartPole [3]. Accurately scaling to complex image input environments requires non-trival engineering efforts. Second, MCTS RL algorithms such as MuZero [30] require a large amount of computations. For example, MuZero needs 64 TPUs to train 12 hours for one agent on Atari games. The high computational costs pose problems both for future development of such methods as well as practical applications.

We think our open source implementation of EfficientZero can drastically accelerate the research in MCTS RL algorithms. Our implementation is computationally friendly. To train an Atari agent for 100k steps, it only needs 4 GPUs to train 7 hours. Our framework could also potentially have a large impact on many real world applications, such as robotics, since it requires significantly less samples.

Our open source framework aims to provide an easy way to understand the implementation, while keeping relatively high compute efficiency. As shown in Fig. 3, the system is composed of four components: the replay buffer, the experience sampling actor, the reanalyze training target preparation module and the training component. To make sure the framework is easy to use, we implement them based on Ray [28], and the four components are implemented as ray actors which runs in parallel. The main computation bottleneck is in the reanalyze module, which samples from the replay, and run MCTS search on each observations. To accelerate the reanalyze module, we split the reanalyze computation into CPU part and GPU part, such that computation on CPU and GPU are run in parallel. We use different number of actors between CPU and GPU to match their total throughput. To increase the throughput on GPU, we also collocate multiple batch computation threads on one GPU, as in Tian et al. [40]. We also implement the MCTS in C++ to avoid performance issue with Python on large amounts of atomic computations. For more details, please refer to the appendix.

6 Experiments

In this section, we aim to evaluate the sample efficiency of the proposed algorithm. Here, the sample efficiency is measured by the performance of each algorithm at a common, small amount of environment transitions, i.e. the better the performance, the higher the sample efficiency. More specifically, we use the Atari 100k benchmark. Intuitively, this benchmark asks the agent to learn to play Atari games within two hours of real-world game time.

6.1 Environments

We compare our method and all baselines on the Atari 100k benchmark. Atari 100k was first proposed by the SimPLe [19] method, and is now used by many sample-efficient RL works, such as Srinivas et al. [38], Laskin et al. [24], Kostrikov et al. [22], Schwarzer et al. [35]. The benchmark contains 26 Atari games, and the diverse set of games can effectively measure the performance of different algorithms. The benchmark allows the agent to interact with 100 thousand environment steps, i.e. 400 thousand frames due to a frame skip of 4, with each environment. 100k steps roughly correspond to 2 hours of real-time game play, which is far less than the usual RL settings. For example, DQN [26] uses 200 million frames, which is around 925 hours of real-time game play. Note that the human player's performance is tested after allowing the human to get familiar with the game after 2 hours as well. In some sense, the agent's score is comparable to that of human's. We report the raw performance on each game, as well as the mean and median of human normalized score. The human normalized score is defined as: $(\text{score}_{\text{agent}} - \text{score}_{\text{random}})/(\text{score}_{\text{human}} - \text{score}_{\text{random}})$.

6.2 Baselines

We compare our method to the following baselines. (1) SimPLe [19] is a model-based RL algorithm that learns an action conditional video prediction model and trains PPO within the learnt environment, which also first propose the Atari 100k benchmark (2) OTRainbow [20] is a method that tunes the hyper-parameters of the Rainbow [17] method to achieve higher sample efficiency. (3) CURL [38] is a method that use contrastive learning as a side task to improve the image representation quality. (4) DrQ [22] adds data augmentation to the input images, while learning the original RL objective. (5) SPR [35] propose to augment the Rainbow [17] agent with data augmentation as well as a multi-step consistency loss using BYOL-style self-supervision. (6) MuZero [30] which our method builds on top. We run it for 100k environment steps with our set of hyper-parameters, which is better for the limited data setting (7) Random Agent (8) Human performance.

All of the baselines and our model are trained with the same number of environment steps, i.e. 100k steps or 400k frames equivalently. During evaluation, the agent is limited to the standard 108,000 frames, or equivalently 27,000 steps per episode.

6.3 EfficientZero Hyper-Parameters

We note a few hyper-parameters that are important for the performance. For a full list, please refer to the appendix. We weigh original losses similar to MuZero, i.e. 1.0 for reward loss, 1.0 for policy loss and 0.25 for value loss. For our proposed self-supervised consistency loss, we weigh it by 0.5. We find that in our limit data scenario, a batch size of 256 performs the best. Larger batch sizes perform worse, potentially due to insufficient noise to escape local minima [11]. We use a constant learning rate of 0.2 with momentum SGD throughout the learning. Decaying the learning rate in the middle hurts the final performance. We train for 200k gradient steps. To stabilize the training, we use a target network that is updated every 200 training steps to compute the policy and value target. The self-play network is updated every 600 training steps. We also changed how Q-values are normalized in the MCTS, since we find that the previously used normalization tends to be over-confident at places with very small Q-values. This results in inefficient exploration towards the end of trajectory. Please see appendix for more details.

Table 1: Scores achieved on the Atari 100k benchmark. EfficientZero achieves similar to human-level performance with only 2 hours of real-time game play. Our method is 104% and 156% better than the previous SoTA performance, in median and mean human normalized score respectively.

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	**1245.6**
Amidar	5.8	1719.5	88.0	82.8	142.1	102.8	**176.3**	38.8	109.3
Assault	222.4	742.0	527.2	351.9	600.6	452.4	571.0	500.1	**1370.2**
Asterix	210.0	8503.3	1128.3	628.5	734.5	603.5	977.8	1734.0	**6518.8**
Bank Heist	14.2	753.1	34.2	182.1	131.6	168.9	**380.9**	192.5	338.8
BattleZone	2360.0	37187.5	5184.4	4060.6	14870.0	12954.0	16651.0	7687.5	**18188.0**
Boxing	0.1	12.1	9.1	2.5	1.2	6.0	35.8	15.1	**43.6**
Breakout	1.7	30.5	16.4	9.8	4.9	16.1	17.1	48.0	**411.1**
ChopperCmd	811.0	7387.8	1246.9	1033.3	1058.5	780.3	974.8	**1350.0**	1218.8
Crazy Climber	10780.5	35829.4	62583.6	21327.8	12146.5	20516.5	42923.6	56937.0	**98800.0**
Demon Attack	152.1	1971.0	208.1	711.8	817.6	1113.4	545.2	3527.0	**10465.0**
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	262.5
Gopher	257.6	2412.5	771.0	778.0	669.3	636.3	715.2	1256.0	**4758.8**
Hero	1027.0	30826.4	2656.6	6458.8	6279.3	3736.3	7019.2	3095.0	**9715.0**
Jamesbond	29.0	302.8	125.3	112.3	**471.0**	236.0	365.4	87.5	315.0
Kangaroo	52.0	3035.0	323.1	605.4	872.5	940.6	**3276.4**	62.5	1775.0
Krull	1598.0	2665.5	4539.9	3277.9	4229.6	4018.1	3688.9	4890.8	**7129.0**
Kung Fu Master	258.5	22736.3	17257.2	5722.2	14307.8	9111.0	13192.7	18813.0	**24912.5**
Ms Pacman	307.3	6951.6	**1480.0**	941.9	1465.5	960.5	1313.2	1265.6	1137.0
Pong	-20.7	14.6	12.8	1.3	-16.5	-8.5	-5.9	-6.7	**20.8**
Private Eye	24.9	69571.3	58.3	100.0	**218.4**	-13.6	124.0	56.3	100.0
Qbert	163.9	13455.0	1288.8	509.3	1042.4	854.4	669.1	3952.0	**15465.6**
Road Runner	11.5	7845.0	5640.6	2696.7	5661.0	8895.1	**14220.5**	2500.0	7331.0
Seaquest	68.4	42054.7	683.3	286.9	384.5	301.2	583.1	208.0	**1737.5**
Up N Down	533.4	11693.2	3350.3	2847.6	2955.2	3180.8	**28138.5**	2896.9	13435.0
Normed Mean	0.000	1.000	0.443	0.264	0.381	0.357	0.704	0.562	**1.800**
Normed Median	0.000	1.000	0.144	0.204	0.175	0.268	0.415	0.227	**0.848**

6.4 Results

Table 1 shows the results of EfficientZero on the Atari 100k benchmark. Normalizing our score with the score of human players, EfficientZero achieves a mean score of 1.80 and a median score of 0.848. As a reference, DQN [26] achieves a mean and median performance of 2.20 and 0.959 on these 26 games, however it is trained with 500 times more data (200 million frames). For the first time, an agent trained with only 2 hours of game data can outperforms human player in terms of the mean performance. Among all games, our method outperform human in 12 out of 26 games. Compared with the previous state-of-the-art method (SPR [35]), we are 156% and 104% better in terms of mean and median score respectively.

Table 2: Ablations of the self-supervised consistency, end-to-end value prefix and model-based off-policy correction on a subset of Atari games. (Scores on the Atari 100k benchmark)

Game	Full	w.o. consistency	w.o. value prefix	w.o. off-policy correction
Asterix	**6218.8**	1350.0	2268.8	2706.3
Qbert	**15268.8**	2304.7	14396.9	3948.4
UpNDown	**10238.1**	3040	5741.9	3240

6.5 Ablations

In Section 4, we discuss three issues that prevents MuZero from achieving high performance when data is limited. Those three issues are: the lack of environment model supervision, the state aliasing issue and the off-policy target value issue. We also propose three corresponding approaches to fix those issues, i.e. the self-supervised consistency loss, the end-to-end value prefix learning and the model-based off-policy correction. We demonstrate the usefulness of the combination of those approaches on a wide range of 26 Atari games. In this section, we will analyze each component individually.

First, we do an ablation study by removing the three components from our full model one at a time. Due to limited on computational resources, we select a subset of Atari environments, and only run 100k training steps. As shown in Table 2, we find that removing any one of the three components will lead to a performance drop compared to our full model. This shows that all of three components leads to the improved performance.

We further validate our assumptions in the end-to-end learning of value prefix, i.e. the state aliasing problem will cause the hardness in predicting the reward, and end-to-end learning of value prefix can alleviate this phenomenon. To fairly compare directly predicting the reward versus end-to-end learning of the value prefix, we need to control for the dataset that both methods are trained on. Since during the RL training, the dataset distribution is determined by the method, we opt to load a half-trained Pong model and rollout 100k steps as the common static dataset. We split this dataset into a training set and a validation set. Then we run both the direct reward prediction and the value prefix method on the training split. As shown in Figure 4, we find that the direct reward prediction method has lower losses on the training set. However, the value prefix's validation error is much smaller when unrolled for 5 steps. This shows that the value prefix method avoids to overfit the hard reward prediction problem, and thus it is able to reduce the state aliasing problem, reaching a better generalization performance.

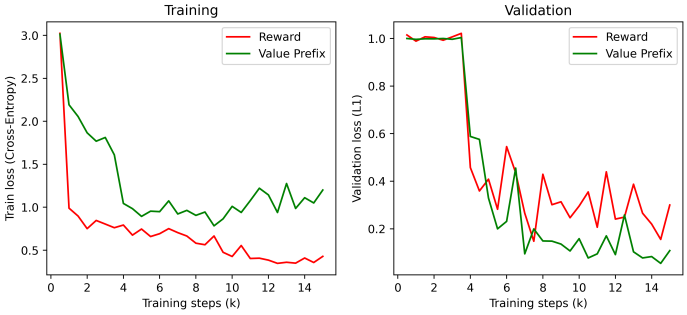

Figure 4: Training and validation losses of direct reward prediction method and the value prefix method.

Please refer to appendix for more ablations and results.

7 Discussion

In this paper, we propose a sample-efficient model-based method EfficientZero. It achieves human level performance on the Atari games with as little as 2 hours of game play experience. This work is one step towards running RL in the physical world with complex sensory inputs.

In this work, we do not deal with stochastic environments, as well as continuous action space. To apply this method on a wider range of tasks, we plan to extend to those directions in the future. The environment model is learned from scratch in our work, while a human can gradually expands his knowledge. We also plan to study how to combine this framework with life-long learning.

References

[1] Bruce Abramson. *The expected-outcome model of two-player games*. Morgan Kaufmann, 2014.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[3] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[5] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[7] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. *arXiv preprint arXiv:2011.10566*, 2020.

[8] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

[9] Joery A de Vries, Ken S Voskuil, Thomas M Moerland, and Aske Plaat. Visualizing muzero models. *arXiv preprint arXiv:2102.12924*, 2021.

[10] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[11] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pages 797–842. PMLR, 2015.

[12] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

[13] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[14] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[15] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[16] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[18] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. *arXiv preprint arXiv:2104.06303*, 2021.

[19] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[20] Kacper Kielak. Do recent advancements in model-based deep reinforcement learning really improve data efficiency? *arXiv preprint arXiv:2003.10181*, 2020.

[21] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[22] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[23] Anurag Koul. Pytorch implementation of muzero. `https://github.com/koulanurag/muzero-pytorch`, 2019.

[24] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

[25] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[27] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[28] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

[29] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[30] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[31] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *arXiv preprint arXiv:2104.06294*, 2021.

[32] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[33] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[35] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[37] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[38] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

[39] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[40] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and Larry Zitnick. Elf opengo: An analysis and open reimplementation of alphazero. In *International Conference on Machine Learning*, pages 6244–6253. PMLR, 2019.

[41] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[42] Hado van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *arXiv preprint arXiv:1906.05243*, 2019.

[43] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[44] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[45] Aurèle Hainaut Werner Duvaud. Muzero general: Open reimplementation of muzero. `https://github.com/werner-duvaud/muzero-general`, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 7
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Our work proposes a data-efficient method to learn policy from image based sensory. In the future, such method might be useful for training real world robots. The deployment of real-world robots might cause negative social effect, which has been widely discussed. Those negative effects for example, include the lose of working opportunities, and physical security risk for the people close to the robot when the robot malfunctions.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include them in an URL in the supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We specify them in the supplemental material.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We only run our experiment once, due to limitation on computation resources. We expect to run multiple times and report error bars for the camera ready version.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [Yes] The Atari environment, or ALE is released as free, open-source software under the terms of the GNU General Public License.

   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]