CPSC533V Final Project

Tianyu Hua(38066445)/ Ke Han Xiao(32196560)

16/Dec/2021

1 Sample efficient RL

Reinforcement learning has achieved great success on many challenging problems, but most of them require a large number of environmental interactions. On the other hand, human beings do not need that much amount of practice to master one thing, and the collection of those interactions is costly in reality. Therefore, sample efficient RL has been proposed.
One way to achieve this is to use model-based methods, which use both data from real environments and the "imagined data" from the model to train the policy, making these methods sample-efficient.

2 Model-based method in image-based environments:

2.1 MuZero algorithm

"Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model"
MuZero is based on the AlphaZero algorithm, which extends the agent to a broader set of environments including single-agent domains and non-zero rewards at intermediate timesteps [4].

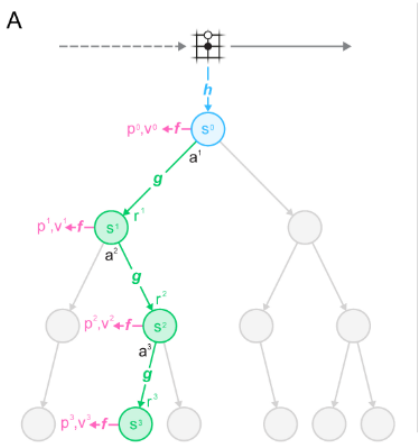
2.1.1 Prediction model

At each time-step t, for the next k steps, the predictions are made at each by a model μ_θ, with parameters θ, conditioned on past observations o1, o2,..., ot and future actions $a_{t+1}, ..., a_{t+k}$. The model predicts three future quantities: 1) The policy $p_t^k \approx \pi(a_{t+k+1}|o1, ..., ot, a_{t+1}, ..., a_{t+k})$, the value function $v_t^k \approx E[u_{t+k+1} + \gamma u_{t+k+2} + ...|o1, ..., ot, a_{t+1}, ..., a_{t+k}]$, and the immediate reward $r_t^k \approx u_{t+k}$, 3 where u. is the true, observed reward, π is the policy used to select real actions, and γ is the discount function of the environment.

2.1.2 Internally with each state

At each time-step t, given the current state s_t, the model is represented by the combination of a representation function, a dynamics function, and a prediction function. The dynamics (deterministic transition) function, $(r_k, s_k) = g_\theta(s_{k-1}, a_k)$, is a recurrent process that computes, at each

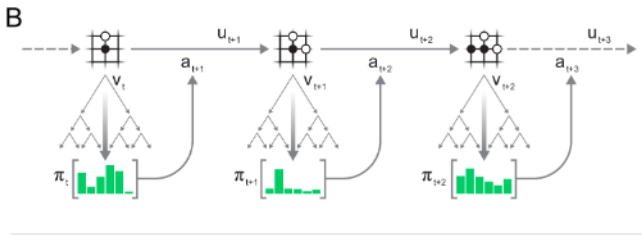
hypothetical step k, an immediate reward r_k and an internal state s_k.



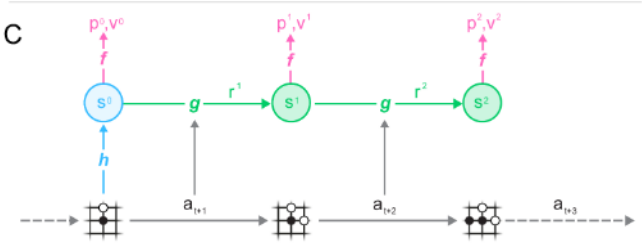
The policy and value functions are computed from the internal state s_k by the prediction function, $pk, vk = f_\phi(sk)$, akin to the joint policy and value network of AlphaZero, MuZero also uses the same architecture as one or two convolutional layers that preserve the resolution but reduce the number of planes, followed by a fully connected layer to the size of the output.
The "root" state s0 is initialized using a representation function that encodes past observations, $s0 = h_\phi(o1, ..., ot)$, the historical actions may also be encoded for atari games because their actions do not necessarily have a visible effect on the observation.

2.1.3 Interact with the environment

A Monte-Carlo Tree Search is performed at each timestep t, and the action at+1 is sampled from the search policy π_t, which is proportional to the visit count for each action from the root node.
Then, Similar to AlphaZero's search, as the agent makes use of the policy, value, and reward estimates produced by the current model parameters θ, the MCTS algorithm outputs a recommended policy π_t and estimated value v_t. An action a_t+ is then selected by π_t.
The environment receives the action and generates a new observation $o_t + 1$ and reward $u_t + 1$. Similar to AlphaZero, all parameters of the model are trained jointly to accurately match the policy, value, and reward, for every hypothetical step k, to corresponding target values observed after k actual timesteps have elapsed. The improved policy targets are generated by an MCTS search; the first objective is to minimize the error between predicted policy p_t^k and search policy π_{t+k}.
However, by the property of Atari games, to allow intermediate rewards and discounting reward for long episodes, we bootstrapping n steps into the future from the search value, $zt = u_{t+1} + \gamma u_{t+2} + ... + +\gamma^n v_{t+n}$



At the end of the episode, the trajectory data is stored in a replay buffer. When a trajectory is sampled from the replay buffer, the parameters of the representation, dynamics, and prediction functions are jointly trained, end-to-end by backpropagation-through-time, to predict three quantities: the policy $pk \approx \pi_{t+k}$, value function $v_k \approx z_{t+k}$, and reward $r_{t+k} \approx u_{t+k}$, where z_{t+k} is a sample return: either the final reward (board games) or n-step return (Atari).



Then, the second objective for the agent will be to minimize the error between the predicted value v_t^k and the value target, $zt + k$.
Finally, define the reward targets to be the observed rewards, we want to minimize the error between the predicted reward r_t^k and the observed reward u_{t+k}.
The loss-function is defined as:

$$l_t(\theta) = \sum_0^k l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, p_t^k) + c(\theta)^2 \tag{1}$$

where l^r, l^v, l^p are loss functions for reward, value and policy respectively.
Through this algorithm, we are able to extend AlphaZero to more general environments.

3 Sample efficient Model-based algorithm in image-based environments:

However, MuZero requires a lot of samples for training, which is not sample efficient. In reality, we normally do not have that much of samples. There are, in principle, 3 reasons for why MuZero is not sample efficient:

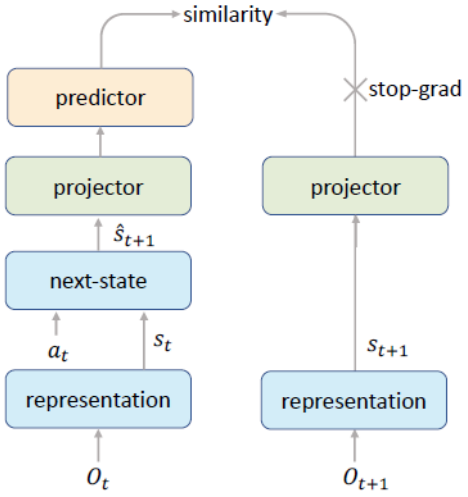
3.1 Self-Supervised Consistency Loss

3.1.1 Problem

In the original MuZero algorithm, the reward is only a scalar signal and often is sparse. It does not well describe the environment. Moreover, training the Value functions using bootstrapping is noisy, and the policy is trained solely by the MCT search. Thus, in a complicated scenario (as Atari games), the MuZero model is not able to accurately and efficiently simulate the environment. Therefore, it needs a huge amount of training data to achieve proper accuracy.

3.1.2 Solution

The idea comes from a recently proposed SimSiam self-supervised framework [1]. The authors proposed a self-supervised method that learns the transition function, along with the image representation function in an end-to-end manner.



By the similar idea of SimSiam approach, the method takes two augmentation views of the state transition and pulls the output of the $S^{\wedge}_{t+1}$ branch close to the O_{t+1}'s representation, where the O_{t+1} branch is an encoder network without gradient, and the $S^{\wedge}_{t+1}$ is the same encoder network with the gradient and a predictor head. The asymmetric predictor design and the stop gradient operation were first introduced by BYOL to avoid collapse situations [2, 3].

3.2 End-To-End Prediction of the Value Prefix

3.2.1 Problem

There are aleatoric uncertainty of the underlying environment happened in the game, which inevitably causes prediction error during the error prediction process. Then, due to the property of the MCT search, with deeper searching depth, these uncertainties will cause more severe problems.

3.2.2 Solution

Predicting the reward from an aliased state is generally hard, especially as MCT expands deeper. This will result in sub-optimal exploration as well as sub-optimal action search. However, it is easy to expand a failure path at a given state within some finite time period. With this idea, instead of using the naive value estimation in UCT, the author instead proposed an end-to-end manner per-step rich supervision neural network to take a future chain of states $(s_t, s_{t+1}, ..., s_{t+k}$ to predict a scalar output, which is the value-prefix(sum of the future rewards) of the UCT.
During the training time, as the neuro network is supervised at every time step since the value prefix can be computed whenever a new state comes in, this per-step rich supervision can be trained well even with limited data.
Moreover, since the output value does not depend on the previous reward, this method can automatically handle the intermediate state aliasing problem.

3.3 Model-Based Off-Policy Correction

3.3.1 Problem

For the sake of modeling complex environment, "the MuZero algorithm uses a multi-step reward rather than the true reward of winning or losing. In the MuZero algorithm, the value target is computed by sampling a trajectory from the replay buffer and computing. 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. This causes severe off-policy issues during the training process, thus hindering the agent from converging.
Moreover, when data is limited, we have to reuse the data sampled from a much older policy, thus exaggerating the inaccurate value target issue.

3.3.2 Solution

To solve this issue, as in MuZero (AlphaZero), we have modeled the environment. The authors use the model to imagine an "online experience". They only use rewards of the old trajectory within a dynamic horizon l, which the horizon is smaller if the trajectory is older, and the lefting trajectory will be filled by MCTS search with the current policy on the last state s_t+l. The empirical mean will be computed 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(the trajectory we try to expand). Overall, the value target function can be written as following:

$$z_t = sum_{i=0}^{l-1} \gamma^i u_{t+i} + \gamma_l^{MCTS} V_{t+l} \tag{2}$$

4 Experiment

The following table presents our reproduced result of the EfficientZero algorithm [5]. Each of our experiments takes 10 hours on 4 rtx6000, which is more computationally expensive than MuZero. In this experiment, we can see except Freeway, the EfficientZero performs better in all other games.

	Breakout	Boxing	Private Eye	Freeway
MuZero	48.0	15.1	56.3	21.8
EfficientZero	**408.7**	**51.9**	**98.6**	**21.9**

Table 1: A comparison between our reproduced EfficientZero and MuZero. The table shows the scores for three different games on the Atari 100k benchmark.

In the paper "Mastering atari games with limited data", the authors also show the same poor performance of the agent as MuZero on the game Freeway. Moreover, in the Appendix part, they mentioned that the performance on Freeway does not vary even taking off any one of the improving techniques we discussed above.
We estimate the reason may be because:
1) The environment of the Freeway is mostly dependent on the local area around the "chicken". Therefore, the marginal gain of accurately simulating the entire environment may not effectively increase the game performance.
2) Freeway has no unforeseen uncertainty in the environment, and due to the game policy, the native UCT itself has intuitively the same effect as the value prefix we try to estimate. Therefore, the End-To-End Prediction of the Value Prefix does not bring much improvement based on the original Mu-Zero agent.
3) As the failure scenario happens densely, the short-term policy/action may not vary a lot even we use the MCT to simulate the future actions.

5 How to debug RL

The following are a few tips that we have found when experimenting with the EfficentZero algorithm.

- **Try to freeze random seeds EVERYWHERE.** In our experiments, the model output can not be fixed every run, and we've found out that it is because we didn't set the non-deterministic parameter of PyTorch to be False.
- **Be careful with batch norm** When training the self-supervised backbone on multiple GPUs, it is important to convert the batch normalization in the encoder to synchronized batch norm such that the batch statistics can be copied to different devices during training.
- **Visualisation and logging are two important ways to debug your model** The first time we wrote this algorithm, we forgot to apply the stop gradient to our predictor, and we got a very weird loss function. When we finally plot out the internal feature variance, we found that it is the stop gradient that is collapsing the training and making it difficult to optimize.

References

[1] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[2] 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.

[3] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. *arXiv preprint arXiv:2110.09348*, 2021.

[4] 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.

[5] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34, 2021.