MCTS + RL series of technical popularization and research blogs (2):

In Zero

0. Introduction

- The “Mu” in the name of MuZero can be understood as meaning “nothing” or “dream”. The core idea of the MuZero algorithm The purpose of MuZero is to search in a "dream" or "imaginary space" without understanding the dynamic transfer rules of the environment. Specifically, it constructs an abstract Markov decision process (abstract MDP) model, and predicts future data (strategies, value functions, and rewards) directly related to planning based on this model, and then plans based on these predicted data. In short, MuZero first learns an environment model in the latent state space, and then plans based on this learned environment model without interacting with the real environment.

1. Overview

- Building agents with planning capabilities has always been one of the main challenges in the field of artificial intelligence. Tree-based search methods have achieved great success in challenging (and perfect simulator) environments such as chess and go. However, unlike these game environments, in the real world, perfect simulators are often lacking and the environment dynamics are generally complex and unknown.

- In this article, we will introduce the MuZero algorithm, which is an algorithm that can detect the dynamics of the environment without knowing the environment's In the case of Deep Neural Networks (CNNs), by combining tree-based search with a learned model, we achieve superhuman performance on a range of challenging domains where the input is complex visual images.

- Specifically, the MuZero agent learns a latent dynamics model by predicting (predict) future information directly related to the decision (strategy, value, and reward function) to learn the model, and then use this When evaluated on 57 different Atari games (classic video game environments used to test artificial intelligence technology and where model-based planning methods have historically performed poorly), MuZero achieved state-of-the-art performance. When evaluated on the games of Go, chess, and shogi, without any knowledge of the rules of the games, MuZero achieved superhuman performance on par with the AlphaZero algorithm, which relies on the rules of the games.

2. Research background and related work

- General planning algorithms rely on dynamic transition rules of the environment, but in real-world application scenarios, agents usually cannot fully obtain such information. Model-Based RL solves this problem by learning an environment dynamic model. The classic Model-Based RL algorithms include: Dyna, MVE, PILCO , etc. For a detailed summary, see Awesome Model-Based Reinforcement Learning. Model-Free RL is just the opposite. The model-free method does not need to learn the model of the environment first, but directly learns the mapping relationship from state to action through continuous trial and error. The classic algorithms include Deep Q-Network (DQN), Proximal Policy Optimization (PPO) wait.

- Model of the Environment is an important element of the reinforcement learning system. It is used to simulate the behavior of the environment.

  The goal of an environment is to model the behavior of an environment, or more generally, to allow one to infer the behavior of the environment. Traditionally, this model is represented by a Markov-decision process (MDP), where given the current state and action, the environment model can predict the reward and the next state. Once the model is constructed, MDP planning algorithms such as value iteration or Monte Carlo Tree Search (MCTS) can be directly applied to compute the optimal value or optimal policy of the MDP. In large or partially observable environments, the algorithm must first construct a state representation that the model will predict. Representation Learning, Model Learning

  Separation between Model Learning and Planning There may be problems because the agent may not be

  To optimize its representation or model for the purpose of efficient planning, for example, model errors may be exacerbated during planning.

- A common strategy in model-based RL is to model the observation stream directly at the pixel level. It has been hypothesized that deep stochastic models may be able to alleviate the problem of error accumulation[2][3]. However, for large problems, detailed planning at the pixel level is computationally impractical. Other approaches build a latent state space model that can reconstruct the observation stream at the pixel level, or predict its future latent states[4], which helps improve planning efficiency but may focus too much on irrelevant details. All of these previous methods have failed to successfully build models that can perform effective planning in visually complex domains (such as Atari games), and their performance lags behind fine-tuned model-free methods even in terms of data efficiency.

- Recently, a new type of Model-Based RL method has begun to emerge, whose main goal is to directly predict the value function[5].

  The core idea of the method is to construct an abstract Markov decision process (abstract MDP) model so that planning in this abstract MDP model is equivalent to planning in the real environment. In order to achieve this equivalence, it is necessary to ensure that, starting from the same actual state, Value Equivalence

  ÿvalue equivalenceÿ the cumulative reward of the trajectory obtained through the abstract MDP model is the same as that in the real environment.

  The cumulative rewards of the trajectories obtained in the actual environment are consistent.

  ÿ Predictron [5] first introduced the value equivalent model for predicting value (no action).

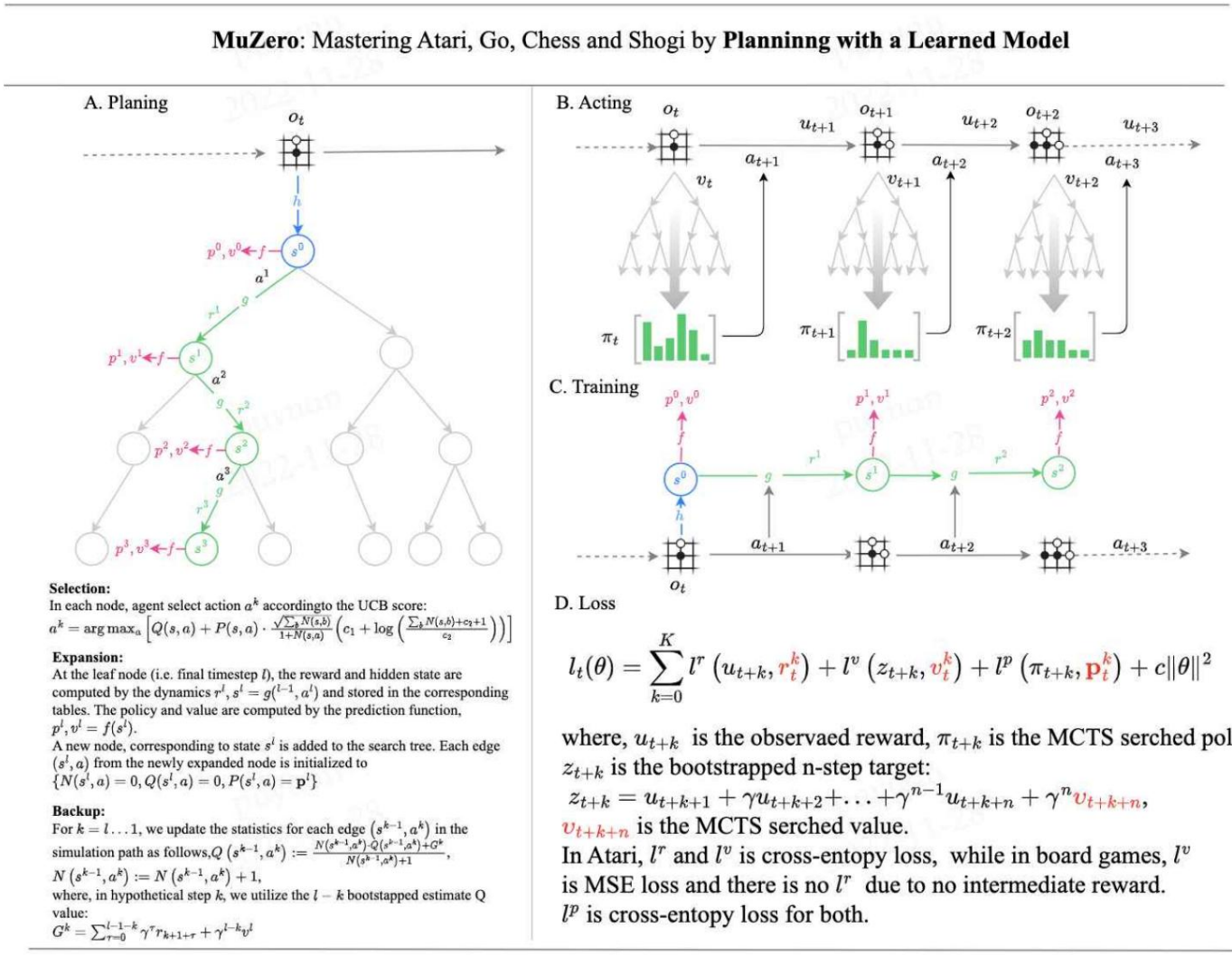
  models). Although the underlying model is still in MDP form, there is no requirement that the transition model matches the real state in the environment. Instead, the MDP model is treated as a hidden layer of a deep neural network. The unfolded MDP is trained so that the expected cumulative sum of rewards matches the expected value in the real environment, for example, by using a temporal difference learning algorithm.

  ÿ Later, value equivalent models were extended to include action value optimization. TreeQN [6] learns a

  Abstract MDP models are learned so that tree search on them (represented by tree neural networks) can approximate the optimal value function. Value iteration networks [7] learn a local MDP model so that value iteration on it (represented by convolutional neural networks) can approximate the optimal value function. Value prediction networks [8] are perhaps the closest predecessors to MuZero: they learn an MDP model based on actual actions; the unfolded MDP is trained so that the reward accumulation and match the real environment under the condition of actual action sequences. However, unlike MuZero, it does not perform policy prediction and the search process relies only on value prediction.

3. Detailed explanation of MuZero algorithm

3.1 Algorithm Overview



(Figure 1: Overview of the MuZero algorithm. A: MuZero agent uses MCTS for planning: representation network **h** encodes consecutive frame observations into a latent state **s0**; the dynamics network predicts **g** The latent state and reward that will be transferred to when executing the action **k r** ÿPrediction network is below **f s** Output policy (probability of selecting each action) and value. B: How MuZero interacts with the environment: For the **ot** In other words, a strategy is obtained using Monte Carlo tree search, and sampling is performed based on this strategy to select executable actions. The probability of the action being selected is It is proportional to the visit count of each action of the MCTS root node. After executing the action, the environment generates a real reward **ot+1** The next moment of observation continues to interact with the environment. At the end of the episode, the trajectory data is stored in the playback buffer. C: MuZero needs to use the dynamics network to expand the step during training, by predicting each step **K** reward, value, and policy to learn the model. D: Specific loss function definition. For the meaning of symbols, refer to the symbol table in the algorithm overview diagram.)

3.2 MuZero Model

- AlphaZero relies on MCTS to make decisions. MCTS is performed in a real environment simulator and requires a policy network during the search process.

  Assistance of the network and value network.

- MuZero also relies on MCTS to make decisions, but it does so in the learned abstract state space.

  Assistance for the next 3 networks:

  - Representation Network

    ÿ $s^0 = h_\theta(o_1, ..., o_t)$. Encode past observations into latent states (or

    The abstract state (sometimes referred to as state) corresponds to the root node at the t moment, and subsequent predictions are all in the potential

    in space.

  - Dynamics Network (also called Mechanism Network)

    ÿ $r^k, s^k = g_\theta(s^{k-1}, a^k)$ That is, simulating the dynamics of the environment, given the state and the selected action, giving the corresponding

    The reward and the transfer to the next moment latent state.

  - Prediction Network

    $v^k, p^k = f_\theta(s^k)$. Given a latent state, predict the policy and value of this latent state.

    ÿ The optimization target of the probability distribution is obtained by MCTS search and is optimized by cross entropy loss $-\pi^T \log(p)$

    change.

    ÿThe optimization target of the value function v^k is the TD target value, which is obtained by MSE loss. $l = (v - z)^2$ to optimize.

- The prediction network is similar to the strategy network and value network used in AlphaZero, that is, MuZero

  Based on AlphaZero, a representation network and a dynamics network are added.

  The two networks are designed to allow MuZero to search in the learned abstract state space.

- Below we will discuss the following questions in turn:

  ÿ What does MuZero mean by “searching in an abstract state space without knowing the rules”?

  ÿ What is special about the MCTS process in MuZero and how does it interact with the environment?

  ÿ How does MuZero train models?

3.3 Abstract State Space

3.3.1 Rules of chess environment

The reason why MuZero learns and searches without knowing the rules is that MuZero uses deep neural networks and MCTS to run

During the process, the following rules of the game were not used:

ÿRule 1: Determination of final status and final reward.

ÿRule 2: Given a state, action, and rules for how to transition to the next state (i.e., environmental dynamics rules, Dynamics).

ÿFor example, in the game of Go (in Go, the piece surrounded by the opponent will be removed from the board).

ÿRule 3: Given a state, who makes the move, that is, which player is the agent that needs to take action; given a state, it is allowed

The set of legal actions.

3.3.2 How AlphaZero uses rules: Taking Tic-Tac-Toe as an example

In Tic-Tac-Toe, suppose the current board is , that is, the root node of the MCTS search is , in each simulation of each MCTS search

When the current state is input into the strategy network, 9 probability values are obtained, and then the final 9 actions are obtained through the following formula:

scoreÿ

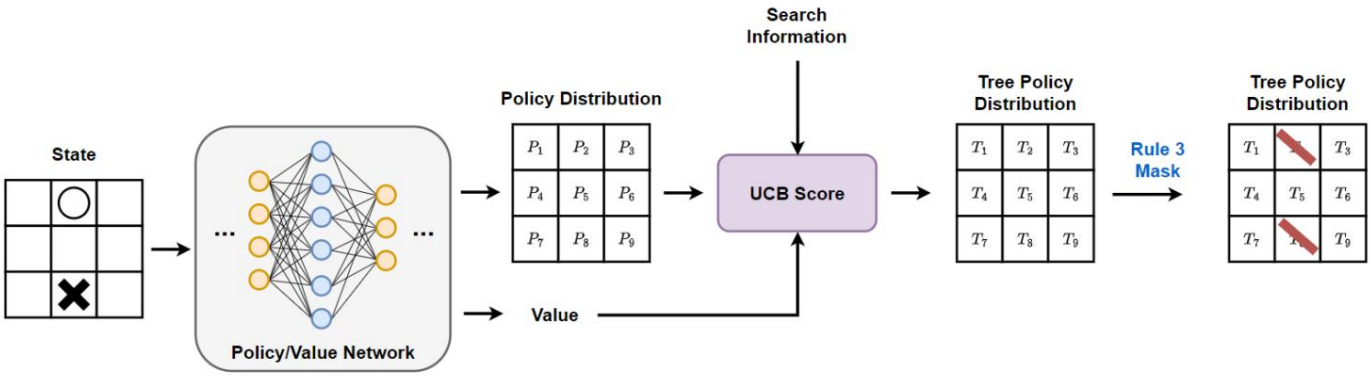
$$score(a) = Q(s, a) + P(s, a) \frac{\sqrt{\ddot{y}\, N(s, b)}}{1 + N(s, a)} 1 \log ( )] \frac{\ddot{y}\, N(s, b) + c + 1 [c}{c2}$$

is the number of visits, is the average value estimated by MCTS simulation, is the strategy, and is the state.

The child is the square root of the total number of visits to the node, and the denominator is the number of visits to the child node. This design will make the child nodes with fewer visits

The value corresponding to this item is relatively large.

(TODO(pu): action mask ÿ)



(Figure 2: AlphaZero's use of Rule 3 in decision making.)

- However, if there are already two pieces on the board, that is, the actions corresponding to these two positions are illegal, you need to block these two actions.

  The corresponding probability value is obtained, and then argmax or sampling is performed to obtain the actual action to be executed. Rule 3 above is used here.

- After selecting action, the board enters a new state, ***(st, at)*** **ÿ st+1 ,** where Rule 2 is used.

- If the game ends during the search and reaches a termination state (need to determine whether it is a termination state), the neural network will not be used at this time

  The network is used to estimate the value of this node, but the actual win or loss ***v* = ±1** is used directly. (Rewards need to be given and also included in the calculation of the final Q

  The output of the value network is combined), and Rule 1 is used here.

3.3.3 MuZero's use of rules

- Using representation network: Latent State **0 *s* =** ***hÿ (o1 , ..., ot)*** , converting the past real chessboard state into an abstract potential

  s0 ÿ

- Using this potential state as the root node of the search tree, instead of the actual situation, it is input into the prediction network:

  p ***k*** , ***k v*** **= *fÿ (s)*** ***k*** , but at the root node MuZero uses rule 3, which is where you can't make moves (already

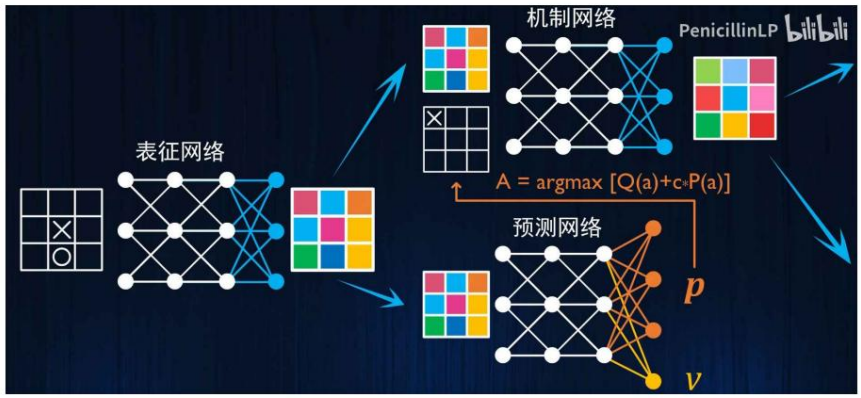
  There is a sub), first mask out the illegal action probability, and then select the action based on ucb_score.

- MuZero does not use Rule 2, but uses a mechanism network dynamics network: to simulate and transfer $r^k, s^k = g\ddot{y}(s^{k\ddot{y}1}, a)$ Model to the next potential state after executing an action.
- There is no final state in the latent state space, and any action in the latent state is legal, so MuZero does not use the rule 1ÿ
- In the search from the root node, since the potential state is unrelated to the real state, it is unknown which actions are legal or whether the game is over. The search ends and MuZero can continue. At this time, MuZero does not use Rule 3.

3.4 MCTS process in MuZero

Similar to AlphaZero, MuZero also obtains the actual interaction with the environment by sampling the distribution of visit counts obtained from MCTS. Interactive actions.

However, in MCTS, MuZero first needs to map the current state to the potential state space through the representation network. Then use the dynamic network and prediction network to search. The search process is shown in the following figure:



(Figure 3: MuZero uses representation networks, dynamic networks, and prediction networks to search in latent space. intention. [9])

Assume that at this time, the three models of MuZero (representation network, dynamics network, prediction network) have been trained and the initial nodes have been s^0 The original chessboard state is generated by the representation network. The information stored in each edge is: $s^0 = h\ddot{y}(o1, ..., of)$ $N(s, a), P(s, a), Q(s, a), R(s, a), S(s, a)$, the process of MCTS searching in the abstract state space can be divided into the following three stages: part:

ÿ 1. Selection

ÿAt each hypothetical time-step $k = 1, ..., l$ in each simulation, Combining value function and strategy function And the number of node visits, select the action with the highest UCB score:

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

ÿ The symbols here mean the number of visits, average value, strategy, reward, and state transition N Q P R S ÿ

ÿ $s^k = S(s^{k\ddot{y}1}, a^k), r^k = R(s^{k\ddot{y}1}, a^k)$ transition The state obtained based on dynamic network records table ÿ reward tableÿ

2. Expansion and evaluation

ÿ Based on the action selected above, expand to a new state node, and the dynamic network will predict the next state and reward.

Encourage.

ÿUntil the moment reaches the leaf node, it is predicted by the dynamic network and saved in

Calculation **a), S(s , a)** In the table, the prediction network is used to predict the, UCB

of **R(s ,** score).

3. Backpropagation

ÿUpdate the sum of the branches where the search trajectory is located, and the evaluation value will be backpropagated back to all nodes in the search path.

For each visited node, its visit count increases and its value is updated for all its child nodes.

The average value of .

$$Q(s, a) := \frac{N(s, a) \times Q(s, a) + G}{N(s, a) + 1},$$

$$N(s, a) := N(s, a) + 1$$

ÿwhere ***G*** is the bootstrapped estimate of the cumulative reward based on:

$$G_k = \sum_{\ddot{y}=0}^{l-k-1} \ddot{y}^{t} r_{k+1+\ddot{y}} + \ddot{y}^{l-k} v^{+}$$

ÿ Because in some environments the magnitude of rewards varies widely, it is necessary to normalize rewards and value estimates to

[0,1] interval:

$$\ddot{y}(s, a) = \frac{Q(s, a) - \min_{s,a \in Tree} Q(s, a)}{\max_{s,a \in Tree} Q(s, a) - \min_{s,a \in Tree} Q(s, a)}.$$

• Execution: The above process is repeated many times (in MuZero, about 800 iterations per move) until the MCTS search is complete.

After that, the visit counts set is returned at the root node. These visit counts are normalized to get relative

The last training had an improved policy. Then from this distribution

$$\pi(a|s) = N(s, a) / \sum_{b} N(s, b)$$

The actual actions of interacting with the environment are sampled.

3.5 Model Training

3.5.1 Loss Function for MuZero Model Training

• All parameters of the model are trained jointly to accurately integrate the policy,

value and reward, which match the corresponding target values observed after actual time steps. However, unlike traditional model-based

Unlike reinforcement learning methods, the latent state in MuZero has no semantics attached to the state of the environment; it is just the latent state of the model.

Its only purpose is to accurately predict future related information: policy, value, reward.

• Since MuZero searches in the abstract MDP space, in order to ensure that the planning in the abstract MDP is equivalent to the planning in the real environment, by ensuring that the Value Equivalence Principle That is, starting from the same real state cumulative return of the trajectory through the abstract MDP matches the cumulative return of the trajectory in the real environment. Therefore, there are the following three optimization goals for MuZero.

ÿ Objective 1: Similar to AlphaZero, the objective improvement strategy is generated through MCTS search. In order to improve the strategy, To minimize the prediction strategy p_t^k The error $l^p(\ddot{y}_{t+k}, p_t^k)$

ÿ Objective 2: Unlike AlphaZero, which uses the average action value of the entire sequence as the target value, MuZero fuses the intermediate reward with a decay factor and the estimated value obtained from the MCTS search to update the target value, which is equivalent to using ***TD (n) ,*** which is faster and has a smaller variance.

ÿ The n-step bootstrapped target value is defined as:

$z_t = u_{t+1} + \gamma u_{t+2} + \ldots + \gamma^{n-1} u_{t+n} + \gamma^n v_{t+n}$ represents the estimated target value. where t+i represents the observation reward and u_{t+i} value obtained by MCTS search. Then minimize the error z_{t+k} between the predicted value of the prediction network and the Difference: $l^{in}(z_{t+k}, in)$

ÿ Goal 3: Minimize the error between the predicted reward and the actual observed reward: $l^r(u_{t+k}, r_t^k)$

$$l^p(\ddot{y}, p) = \ddot{y}^T \log p \quad (1)$$

$$l^{in}(from, in) = \begin{cases} (z - v)^2 & \text{for games} \\ \ddot{y}(z)^T \log v & \text{for general MDPs} \end{cases} \quad (2)$$

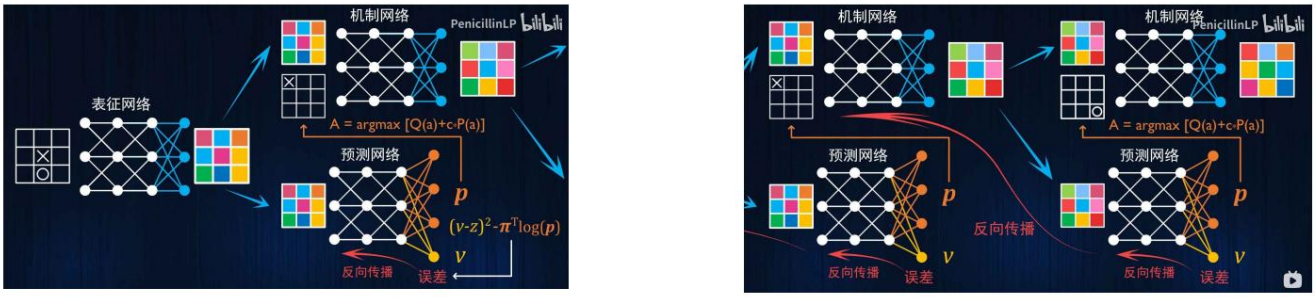
$$l^r(u, r) = \begin{cases} 0 & \text{for games} \\ \ddot{y}(u)^T \log r & \text{for general MDPs} \end{cases} \quad (3)$$

In summary, the total loss function for the three networks used to train MuZero is as follows:

$$lt(\ddot{y}) = \sum_{k=0}^{K} l^p(\ddot{y}_{t+k}, p) + l^{in}(z_{t+k}, \ddot{y}_{t+k}) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k) + c\ddot{y}\ddot{y}\ddot{y}\ddot{y}^2$$

3.5.2 Important details

Minimum sequence length required for training



(Figure 4: (Left) Illustration of the error backpropagating from the prediction network to the representation network in MuZero. (Right) Illustration of the error backpropagating from the prediction network to the dynamics network in MuZero. [9])

According to the search process in the latent space described above, we can see that at time ***t***

- For representation network $s^0 = hÿ(o1, ..., ot)$, whose output is the input s^0 of the prediction network

  So the representation network can be directly updated by the gradient back-propagated from the prediction network.

- For dynamics network $r^k, s^k = gÿ(s^{kÿ1}, a)$, its output will be s^k the next prediction network $p^k, v^k = fÿ(s^k)$ So the gradient of the next prediction network can be back-propagated to dynamics networkÿ

- Therefore, at least two states and their related information are required to propagate gradients between different networks, that is, the minimum sequence length required for training

  The degree is 2.

Replay buffer

- AlphaZero stores each true state and its true policy (obtained from MCTS search to select actions) ot $ÿt$, the strategy generated by the strategy network, the value predicted by the value network, and the real cumulative reward information, that is pt vt zt ***(ot, pt, ÿt, vt, zt)***, without storing the actual selected action.

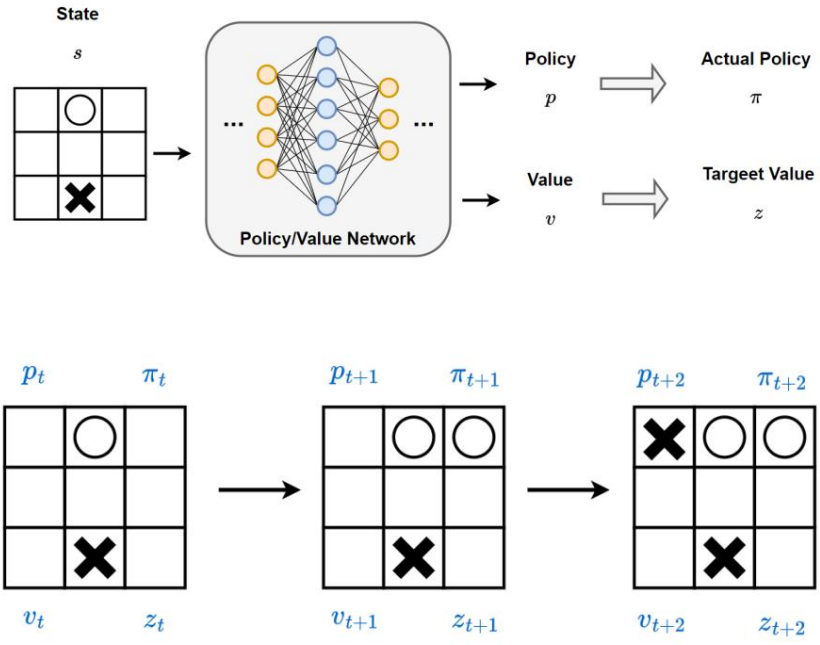


Figure 5: (Top) Output of the policy value network for each state input and optimization goal in AlphaZero. (Bottom) The replay buffer stores information about each state)

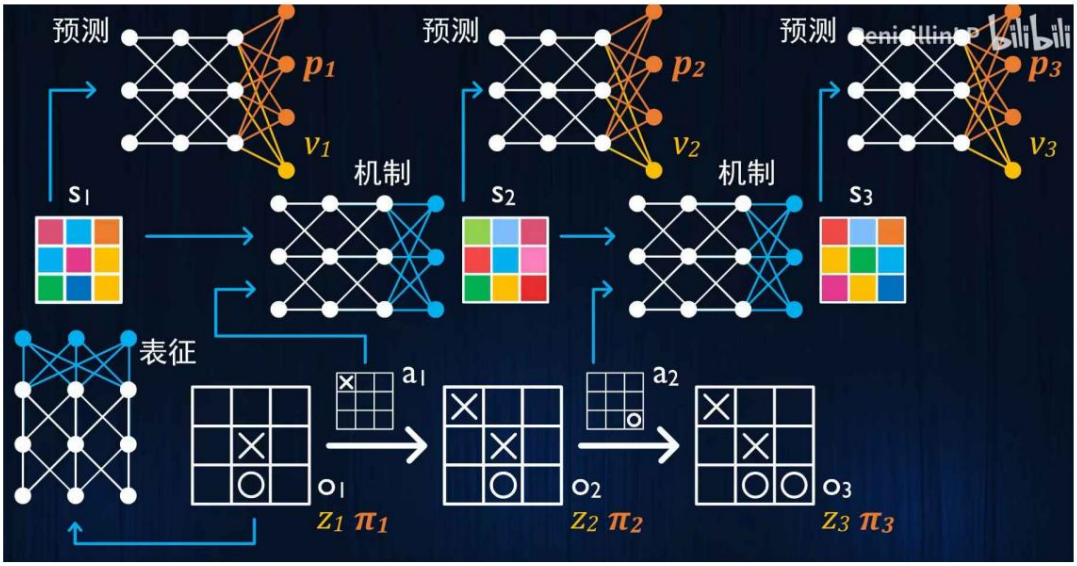
- Compared with AlphaZero, MuZero also stores the actual actions taken between consecutive states, the potential states, i.e. at st ***< ot, at, zt, ÿt, st, pt, vt >***, and as mentioned above, the minimum sequence length required for MuZero training is 2, but the actual

  The sequence length used by MuZero is 6, that is

  < (ot, at, zt, ÿt, st, pt, **vt), ...,(ot+5 , at+5 , zt+5 , ÿt+5 , st+5 , pt+5 , vt+5) >**

  As a training data

  Stored in the replay buffer.



(Figure 6: Schematic diagram of the information stored in MuZero's replay buffer. [9])

4. MuZero Experiment

4.1 Experimental Setup

4.1.1 Experimental Environment

- Classic board games as a basic validation environment for planning problems: Go, chess, shogi.
- 57 games in the Atari environment as a baseline validation environment for the domain of visually complex RL.

4.1.2 Network structure and main hyperparameters

Network structure

The specific structures of the three networks used by MuZero are shown in Figure 7.

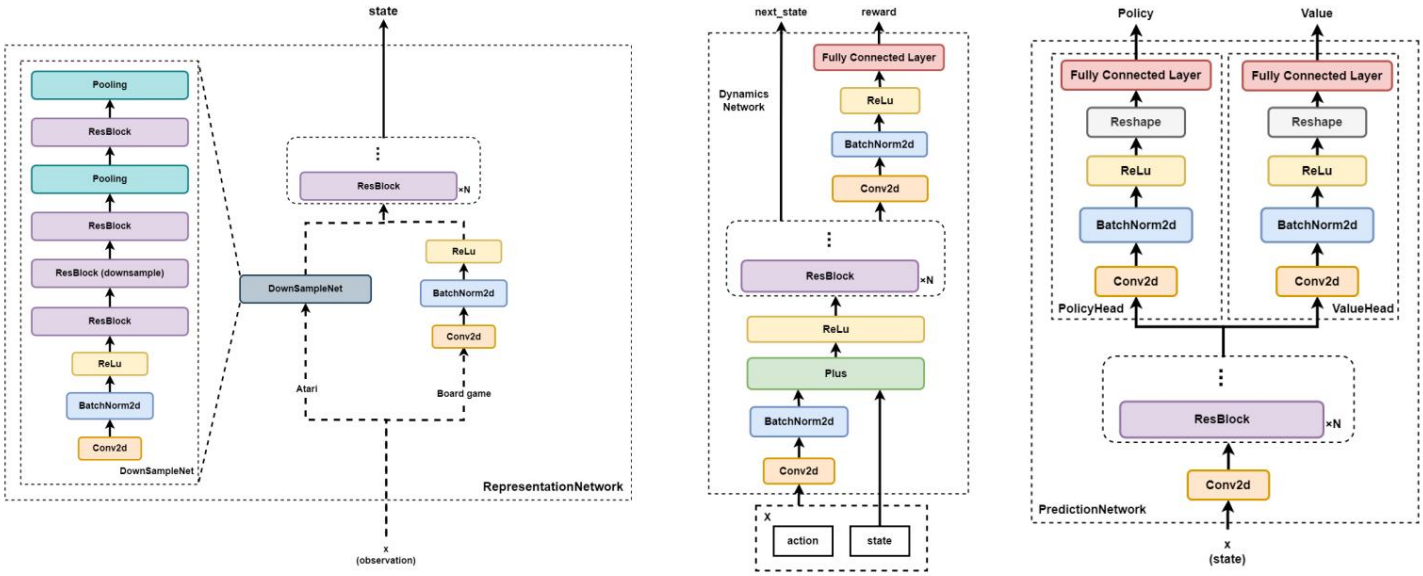
- In MuZero, the representation and dynamic functions use the same architecture as AlphaZero, but the number of ResBlocks used is 16 instead of 20. Each convolution process uses a 3x3 convolution kernel and 256 hidden layers.
- Representation network: For Atari games, since the observation data has a large spatial resolution, we first reduce its spatial resolution through a series of convolution operations with a stride of 2. Specifically, we start with an input observation with a resolution of 96x96 and 128 planes (containing 32 historical frames, each frame contains 3 color channels, and 32 actions, which are broadcast to each plane, so 32*3+32=128), and then use the following strategy for downsampling:
  - ÿ Using convolution with stride 2 and 128 output planes, the output resolution is 48x48.
  - ÿUse 2 ResBlocks of 128 planes.
  - ÿ Using convolution with stride 2 and 256 output planes, the output resolution is 24x24.
  - ÿUse 3 ResBlocks of 256 planes.
  - ÿUse average pooling with a stride of 2 and an output resolution of 12x12.

ÿUse 3 ResBlocks of 256 planes.

ÿUse average pooling with a stride of 2 and an output resolution of 6x6.

ÿAll of the above operations use a 3x3 convolution kernel.

- Dynamics network: Uses a similar architecture to the representation network. It always operates on a latent state. First, the action is encoded as an image, which is then superimposed along the plane dimension with the hidden state from the previous step.
- The prediction network uses a similar architecture to AlphaZero, starting with N common ResBlocks, then split into 2 head, outputs strategies and values respectively.



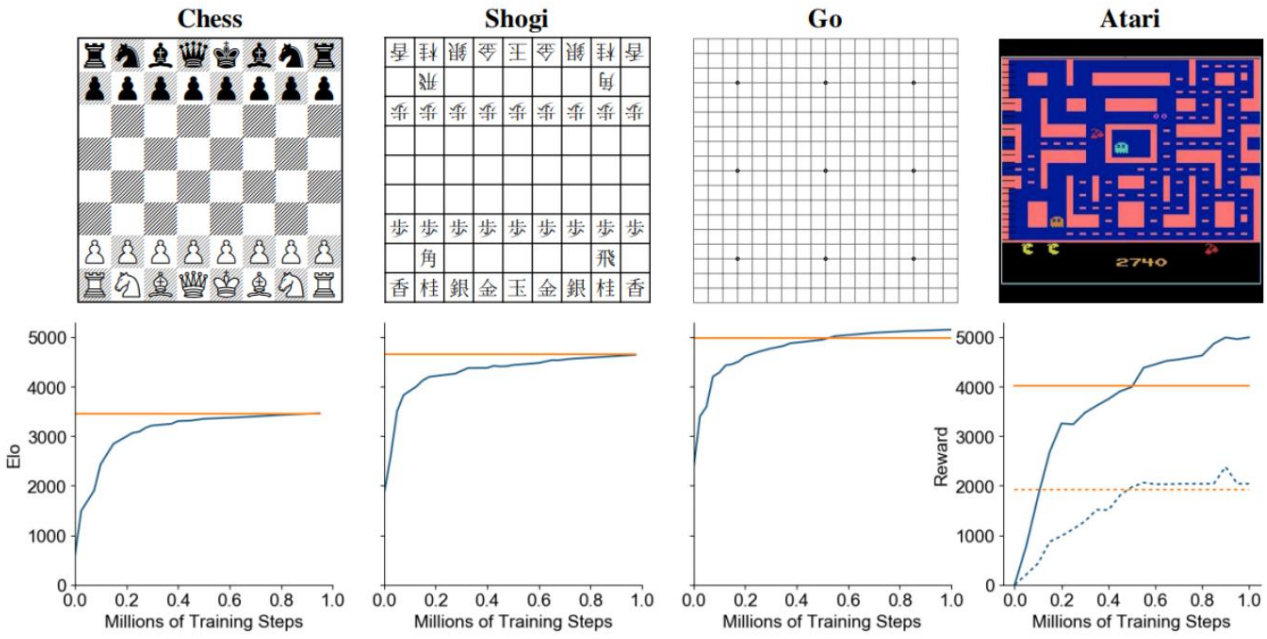
(Figure 7: Left: MuZero's representation network structure diagram. Middle: MuZero's dynamics network structure diagram. Right: MuZero's prediction network structure diagram. Note: The network structure diagrams in this document are based on the actual implementation of LightZero and are slightly different from the original paper.)

Main hyperparameters

> - Hypothetical steps: In each environment, the authors trained MuZero with hypothetical steps K = 5, that is, they deduced 5 steps downward from the current state through the dynamics network, predicted the latent state and corresponding reward to be transferred to at each step, and obtained the predicted policy and value based on the prediction network. The corresponding loss was calculated at these 5 steps to update the network.

parameter	hypothetical steps	dialed nt	mini-batches size	simulations for each search	unroll steps num	weight_dec is
Board Games	5	0.97	2048	800	5	0.0001
Atari			1024	50		

(Table 1: MuZero main hyperparameters. For other detailed parameters, see the pseudo code in the supplementary material of the original paper.)



(Figure 8: Learning curves of MuZero on chess, shogi, go, and Atari.)

4.2.1 Classic board games

In Figure 8, the learning curves of MuZero on chess, shogi, go and Atari are shown. The blue line is the change of MuZero's Elo (or reward) as the number of training steps increases. The orange line is the Elo of AlphaZero. From the first three columns in Figure 8, we can see that in Chess and Shigi, MuZero has achieved similar performance to AlphaZero. In Go, MuZero's performance is slightly better than AlphaZero.

4.2.2 Atari

In the fourth column of Figure 8, the vertical axis reward represents the score in the game, the dotted line represents the median score of the 57 games, and the solid line represents the average score of the 57 games. The blue line shows the change of MuZero's score with the number of training steps, and the orange solid line shows the SOTA R2D2 at that time. (a model-free RL method). As we can see, in Atari, MuZero reached the level of SOTA at the time and greatly surpassed It far exceeds the best previous model-based method of the same type, SimPLe (orange dashed line).

4.2.3 Reanalyze

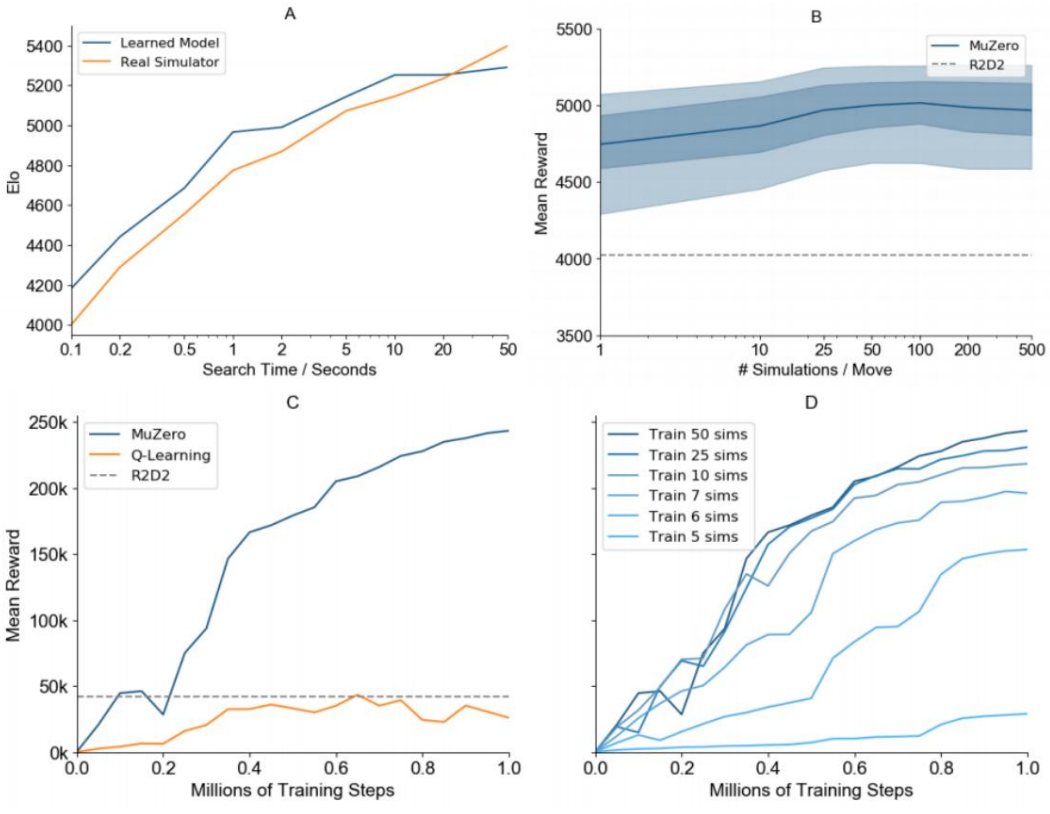
In simple terms, the Reanalyze technique refers to using the latest network to re-run MCTS on the collected data to re-analyze the old trajectory and obtain a new and more accurate target policy/value for training. As can be seen from Table 2, on Atari, MuZero Reanalyze outperforms all previous model-free RL methods.

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

(Table 2: Performance comparison of MuZero and previous model-free RL algorithms on Atari games in large-data (top) and small-data (bottom) settings. All agents except MuZero use model-free RL techniques. The table shows the mean and median scores, as well as the number of environment frames, training time, and number of training steps used for training. The best results are highlighted in bold. In both settings, MuZero achieves new state-of-the-art performance.)

4.2.4 The role of potential environmental models

In order to explore the role of the potential environment model learned by MuZero, the following series of exploratory experiments were conducted on the board game Go and the Ms. Pacman game environment in Atari games.



(Figure 9: MuZero's exploration experiment on Go (A), on 57 Atari games (B), and on Ms. Pacman (C) Exploration experiment (CD).

Scalability of planning in the Go environment • As shown in Figure

9 (A), the performance of AlphaZero, which has been trained using a perfect environment model, and MuZero, which has been trained using a learning environment model (note that both networks are trained with 800 simulations per search, equivalent to 0.1 seconds per search), are evaluated using different search times. It can be seen that MuZero's performance is even slightly better than AlphaZero using a perfect environment model at the same search time.

• It is worth noting that when the single-step search time at evaluation is two orders of magnitude longer than at training, MuZero using the learned environment model Still performing well.

Scalability of Planning in the Atari Environment

• As shown in Figure 9 (B), using a trained MuZero (trained with 50 simulations per search), but using different numbers of simulations during MCTS (per move), observe how the average reward of MuZero changes, with R2D2 as the baseline. The dark line represents the average score, and the shaded area represents the 25th to 75th and 5th to 95th percentiles.

• When the number of simulations per search is less than 100, MuZero's performance increases with the number of simulations. And note that MuZero performs well with only one simulation (equivalent to relying solely on the policy network), indicating that the policy network has learned to internalize the benefits of search during training.

• Furthermore, even when using a much larger number of searches during evaluation than during training (simulations=500), the performance of the learned model remains stable, with only a slight drop. This is in stark contrast to the much better scaling in Go, likely due to the larger inaccuracy of the models learned on the Atari environment compared to Go.

Comparison between Model-based and Model-free methods

• Replace MuZero's Loss with the Q-learning loss function in R2D2, and the policy value dual-head network with a single-head Q-value network. Do not use MCTS during training or evaluation, and you will get a Model-free version of MuZero, as shown by the Q-learning line in Figure 9 (C). The curve comparing it with the original MuZero is shown in Figure 9 (C).

• We can see that MuZero's Q-Learning implementation achieves the same final score as R2D2, but improves more slowly and has much worse final performance than MCTS-based training. This is likely due to MuZero's search-based policy improvement providing a stronger learning signal for the high-bias, high-variance targets used by Q-learning.

The impact of simulation number

• As shown in Figure 9 (D), the effect of different simulation times on performance during training in the Ms. Pacman environment was explored. Note that the number of simulations used to collect data during training is different, but the number of simulations is equal to 50 during evaluation.

• It is obvious that when the number of simulations during training increases, the strategy improves faster. In addition, it can be observed that MuZero When the number of simulations is less than the number of legal actions, effective learning can still be achieved. For example, in the Ms. Pacman environment, there are 8 selectable actions and the algorithm performance is still very good when the number of simulations is 6.

5. Algorithm Practice

• While fully exploring the huge potential of the MCTS series of algorithm technologies, the Shanghai Artificial Intelligence Laboratory Open Source Decision Intelligence Platform (OpenDILab) team sincerely invites developers to explore the LightZero project together. For more information, please refer to the GitHub repository and the technical blog LightZero: Using MCTS as a sail, sailing towards the starry sea of decision AI .

• https://github.com/opendilab/LightZerogithub.com/opendilab/LightZero

• LightZero provides comprehensive benchmarking tools for many complex decision problems. For MuZero, LightZero provides Atari's MuZero entry files in the repository, and its tree search part includes a python version and cpp With the implementation of the new version, developers can gradually get in touch with related algorithms and applications. We welcome all friends who are interested in this to join the practice, discuss and contribute your wisdom.

• To help developers better understand and use LightZero, we also contribute tips on LightZero open source A detailed guide to getting started, collaboration specifications, and future development plans are provided in the LightZero project. You are welcome to read it and make valuable suggestions. Evolution is infinite. Let us explore the unknown and create the future together in the LightZero journey!

6. Summary and Outlook

Many breakthroughs in artificial intelligence are based on either high-performance planning or model-free reinforcement learning methods. MuZero combines the best of both worlds, not only matching the superhuman performance of high-performance planning algorithms in logically complex board games like chess and Go, but also outperforming state-of-the-art model-free RL algorithms in visually complex Atari games. Crucially, MuZero requires (almost) no knowledge of the game rules or environment dynamics, which enables it to be applied to many real-world tasks that lack perfect simulators.

• Future research directions include:

ÿ How to combine with the latest advances in model-based RL to further improve the sample efficiency of the algorithm?

ÿHow to accelerate MCTS?

ÿHow to make the representation space learned by MuZero more semantic and interpretable?

7. References

[1] Schrittwieser, J., Antonoglou, I., Hubert, T. planning et al. Mastering Atari, Go, chess and shogi by with a learned model. 588, 604ÿ609 (2020). Nature

[2] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson.

Learning latent dynamics for planning from pixels. arXiv preprint arXiv:1811.04551, 2018.

[3] 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.

[4] David Ha and J¨urgen Schmidhuber. Recurrent world models facilitate policy evolution. In Proceedings of

the 32Nd International Conference on Neural Information Processing Systems, NIPS'18, pages 2455ÿ2467,

USA, 2018. Curran Associates Inc.

[5] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-

Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and

planning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages

3191ÿ3199. JMLR.org, 2017.

[6] Gregory Farquhar, Tim Rocktaeschel, Maximilian Igl, and Shimon Whiteson. TreeQN and ATreec: Differentiable

tree planning for deep reinforcement learning. In International Conference on Learning Representations,

2018.

[7] Aviv Tamar, YiWu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In Advances

in Neural Information Processing Systems, pages 2154ÿ2162, 2016.

[8] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In Advances in Neural Information

Processing Systems, pages 6118ÿ6128, 2017.

[9] You can play Go without knowing the rules!? [Terrible as MuZero][Part 2]_ÿÿÿÿ_bilibili, https://www.bilibili.com/video/BV1Ey4y1s7zB/?spm_id_from=333.337.search-card.all.click&vd_source=35dd2c5bd9fbf55c9681c89ce63d38c1 2021.