Efficient Learning for AlphaZero via Path Consistency

Dengwei Zhao [1] **Shikui Tu** [1] **Lei Xu** [1]

Abstract

In recent years, deep reinforcement learning have made great breakthroughs on board games. Still, most of the works require huge computational resources for a large scale of environmental interactions or self-play for the games. This paper aims at building powerful models under a limited amount of self-plays which can be utilized by a human throughout the lifetime. We proposes a learning algorithm built on AlphaZero, with its path searching regularised by a path consistency (PC) optimality, i.e., values on one optimal search path should be identical. Thus, the algorithm is shortly named PCZero. In implementation, historical trajectory and scouted search paths by MCTS makes a good balance between exploration and exploitation, which enhances the generalization ability effectively. PCZero obtains 94.1% winning rate against the champion of Hex Computer Olympiad in 2015 on 13×13 Hex, much higher than 84.3% by AlphaZero. The models consume only $900K$ self-play games, about the amount humans can study in a lifetime. The improvements by PCZero have been also generalized to Othello and Gomoku. Experiments also demonstrate the efficiency of PCZero under offline learning setting.

1. Introduction

Combining heuristic search with the deep neural network has been a key to success in Reinforcement Learning (RL) (Silver et al., 2016; 2017; 2018). Many efforts have been made to sequential decision-making problems following such an idea. However, tree search is a slow reasoning process and needs heuristic as a counselor to narrow the search space (Anthony et al., 2017). In AlphaZero, the policy network reduces the width of the search tree while the value network bounds the depth, both of which are learned through the iterative generation of higher quality games. The network's final performance highly depends on the number of self-play games, thus requiring huge computational power for model training. For example, reproductions of AlphaZero[1][2][3] either employ thousands of GPUs or train for years.

In AlphaZero, deep neural network is updated with the self-play generated games, and each game can be regarded as a playing path, which consists of the sequential game states from the initial to the terminal. For one complete game or path, the learning objective is predicting the policy and the value of current state accurately, so as to guide the Monte-Carlo Tree Search (MCTS) effectively. The above regular training process can be improved by a path optimality constraint called *path consistency* (PC), i.e., "*values on one optimal path should be identical*" (Xu et al., 1987), which is suggested to regularize path searching for many sequential decision-making problems simply by adding a weighted penalty to the loss function (Xu, 2018) :

$$L(\theta) = L_{RL}(\theta) + \lambda L_{PC}(\theta), \tag{1}$$

where L_{RL} is the RL loss that results from the interaction with the environment, L_{PC} is the PC constraint, and θ denotes the network parameters.

The key problem is how the term L_{PC} is formulated, which is L_1 or L_2 deviation from an estimated value of optimal path that could be simply a moving average within a window of estimated optimal path (Xu, 2018). It needs further investigation to implement and test the potential of this direction, as well as to explore various possible ways to estimated the value of optimal path. This paper provides a solid evidence to demonstrate the effectiveness of PC for the first time. Our contribution is summarized from three aspects[4]:

- We propose a data-efficient learning algorithm for games, called PCZero, which is built on AlphaZero

*Equal contribution [1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Shikui Tu <tushikui@sjtu.edu.cn>, Lei Xu <leixu@sjtu.edu.cn>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

[1] https://github.com/pytorch/ELF

[2] https://github.com/Tencent/PhoenixGo

[3] https://github.com/leela-zero/leela-zero

[4] The source codes are available at https://github.com/CMACH508/PCZero.

by incorporating PC. Specifically, the training process of AlphaZero is augmented by minimizing the violation of the PC condition. Taking the 13×13 Hex as an example, PCZero is able to defeat AlphaZero, and obtain 94.1% winning rate against MoHex 2.0 (the champion of Hex Computer Olympiad in 2015), which is significantly better than 84.3% by AlphaZero. It is worth noting that PCZero consumes only 900K self-play games during learning, which is a small-scale data that humans can make in a lifetime. Similar results can also be observed when generalizing PCZero to other games, e.g., Othello and Gomoku.

- We propose an effective implementation of PCZero by including the MCTS simulated paths into the computation for PC, in addition to the historical trajectory. The heuristics drawn from the lookahead search of MCTS is beneficial to the network learning in jumping out of the local optimum.
- We extend PC from the consistency of state values to the consistency of feature maps, which are taken from the network layers before the state value's final estimation. Feature consistency provides a stricter constraint on PC nature, and it works as a complement to the value consistency.

2. Related work

Hex is a PSPACE-complete problem (Even & Tarjan, 1976) and widely studied for decades. While playing the game, players place a stone of their color on an empty cell alternatively, and who joins their two sides first is the winner. Early Hex computer players, like Hexy (Anshelevich, 2002), Wolve (Arneson et al., 2008) and MoHex (Arneson et al., 2010), usually utilized game's mathematical properties to assist search algorithms. Learning prior knowledge of the search process by pattern regression, MoHex 2.0 (Huang et al., 2013) has won the champion of Hex Computer Olympiad in 2015. MoHex-CNN (Gao et al., 2017) and EZO-CNN (Takada et al., 2017) further adopt convolutional networks to extract prior information. NeuralHex (Young et al., 2016) applied Q-Learning and EXIT (Anthony et al., 2017) independently designed an Alpha Go-like algorithm to train programs playing Hex. MoHex-3HNN (Gao et al., 2018) built a three-head network to assist MCTS and DeepEZO (Takada et al., 2019) introduced self-play to collect game experience.

Othello is another PSPACE-complete game (Iwata & Kasai, 1994) and computer players are usually built based on tree search assisted by a state value function, some of which have achieved superhuman game capability, like Edax[5]. In recent studies, neural networks (Liskowski et al., 2018) are adopted to predict expert movements and Q-learning (Lucas & Runarsson, 2006; Krol & Brandenburg, 2020) is used to learn playing policy directly. Gomoku is also PSPACE-complete (Reisch, 1980) and some previous work attempts to build strong computer programs using deep learning methods (Schaul & Schmidhuber, 2008; Zhao et al., 2012; Shao et al., 2016) and MCTS (Tang et al., 2016).

In the context of sequential games with perfect information, all possible game states can be included in a game tree starting at the initial state and containing all possible moves. The task of search algorithms is finding an optimal path, which can arrive the preferred terminal state n from initial state s_0, guided by a evaluation function $f(s)$. One classical example is A^* search algorithm (Hart et al., 1968), in which $f(s)$ is computed as the summation of $g(s)$, accumulated cost/reward from s_0 to current state s, and $h(s)$, the future cost/reward from s to the preferred termination n. The path optimality tells that $f(s) = f(n)$ for every node s on an optimal path and $f(s) > f(n)$ for every node s not on an optimal path. CNneim-A (Xu et al., 1987) relies on this optimality to use A^* to make a lookahead scouting to estimate a segment on the optimal path and use the average of f-values from the root to n (i.e., the historical trajectory) and also one on this segment to guide A^* search. The current paper proceeds along this directions with three new developments. First, $f(n)$ is estimated by deep neural networks. Second, a lookahead scouting is made by MCTS instead of A^*. Third, a moving average within a window of estimated optimal path (Xu, 2018) is considered in place of the entire of historical trajectory plus lookahead scouting.

The concept of consistency is also widely employed in reinforcement learning. For example, Q-learning (Mnih et al., 2015) has applied the well-known hard-max Bellman temporal consistency between two adjacent state values. PCL (Nachum et al., 2017a;b) further developed the above one-step consistency to path-wise consistency between the optimal policy and optimal values by assuming the policy is in Boltzmann distribution.

In reinforcement learning, the latest learned policy interacts with the environment to collect experience for policy improvement iteratively (Sutton et al., 1998). Such an online learning paradigm is one of the biggest obstacles to their widespread adoption when the interaction is expensive, or infeasible. And the utilization of collected data is insufficient, requiring a huge amount of experience to train a working model (Yu, 2018). Correspondingly, offline reinforcement learning applied the supervised learning method, which is more sample efficient, on the RL objectives, which utilizes only previously collected offline data without interactions (Fu et al., 2020). However, an important challenge with offline RL is answering counterfactual queries, generalizing the model to unexperienced situations (Levine et al., 2020).

[5]https://github.com/abulmo/edax-reversi

In this paper, we will show that PC can not only improve learning efficiency for online RL but also enhance generalization capacity on value estimation in offline situations.

3. Method

PC is investigated under the scenario of board games in this paper, which are typical delayed reward applications. Immediate reward $r(s,a)$ is always zero and $g(s) = 0$ holds until arriving the termination state. Thus, $f(s)$ is equal to $h(s)$, which is equivalent to the state value $v(s)$ in RL conceptually. Therefore, the identical of $f(s)$ is turned into the consistency of $v(s)$. What's more, if the optimal playing policy and state transition function are deterministic, which is $\pi^*(s) = \arg\max_{a \in \mathcal{A}} r(s,a) + v(s')$ and $s' = T(s,a)$ (T is the state transition function), consistency of state values can be derived from the hard-max Bellman temporal consistency by setting $\gamma = 1.0$:

$$v^*(s) = r\left(s, \pi^*(s)\right) + \gamma v^*(s') = v^*(s'). \tag{2}$$

Any two neighboring nodes have identical state values in Equation 2 is an equivalent expression of PC constraint. Therefore, path consistency is also a condition that optimal state value $v*$ should satisfy in RL. Adding PC constraint into the objective function is reasonable to help learn the optimal policy from the perspective of solving the constrained optimization problem. Although PC for RL and AlphaGo type search was schematically proposed four years ago(Xu, 2018), not only it is yet unknown whether it works well but also there are potentials of further developments on estimating the value of optimal path.

3.1. PC computation with historical and heuristic paths

In AlphaZero, $v(s)$ is computed by a deep neural network that takes the state configuration s as input. PCZero will be trained by further restricting that the estimated $v(s)$ should deviate as less as possible along the optimal path. In practice, there are many possible ways to realize the PC constraint. One way is to compute the variance of the values within a sliding window W_s containing state s, i.e.

$$L_{PC}(s) = (v - \bar{v})^2, \tag{3}$$

where $\bar{v}$ is the average state value in W_s. As illustrated in Figure 1, for a terminated game sequence, $\bar{v}$ is averaged over the l upstream nodes and k downstream nodes. The computation procedure is summarized in Algorithm 1.

If playing sequence is collected through expert competitions or self-play of near-optimal policy, $\bar{v}$ obtained after the termination of games is more reliable for the training of PCZero. However, at the initial stage of the training process, the quality of generated game experience is poor and the network's estimation of state value is not accurate. PC constraint computed by Algorithm 1 will lead the model to converge to a local optimum quickly. To tackle this problem, PCZero is extended to calculate $\bar{v}$ using not only historical trajectory but also scouted heuristic path provided by MCTS while doing self-play. As displayed in Figure 2, the upstream path of the segment window consists of the already obtained historical nodes before state s, while the downstream path is determined by the best first search according to the visiting times in the search tree, which is constructed by MCTS with state s as the root node. Among the obtained downstream nodes, the ones far from the root may not be consistent with the historical ones and may not be necessary to be selected into the final optimal sequence. This uncertainty increases the variance of the value predictions within the window. Such variation, which will be captured by PCZero, is exactly the strength to rescue the network from a local optimum. The relative length of the heuristic path to the history path controls the balance between the satisfaction of optimal PC nature, which can accelerate convergence, and randomness injection, avoiding local optimum, which is similar to the trade-off between exploration and exploitation in RL. The computational procedure for PCZero using MCTS self-play (MCTS-PCZero) is provided in Algorithm 2.

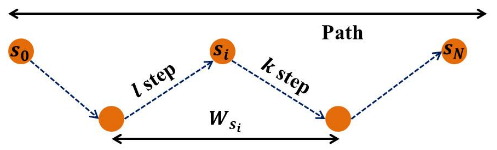

Figure 1. $\bar{v}$ calculation with historical path in a terminated game.

Algorithm 1 $\bar{v}$ estimation for a terminated sequence

```
**Input:** game sequence $S$ and value network $f$.
Initialize $V = \{f(s_i)|\forall s_i \in S\}$.
**for** $i = 1$ **to** $|S|$ **do**
    $\bar{v}(s_i) = \mathbf{mean}\{V_{\max\{1,i-l\}:\min\{i+k,|S|\}}\}$
**end for**
```

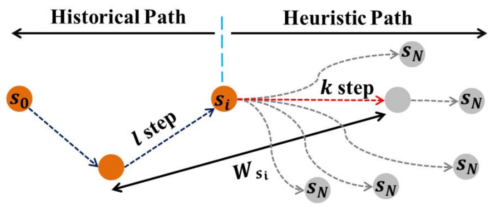

Figure 2. $\bar{v}$ calculation with both historical and heuristic path.

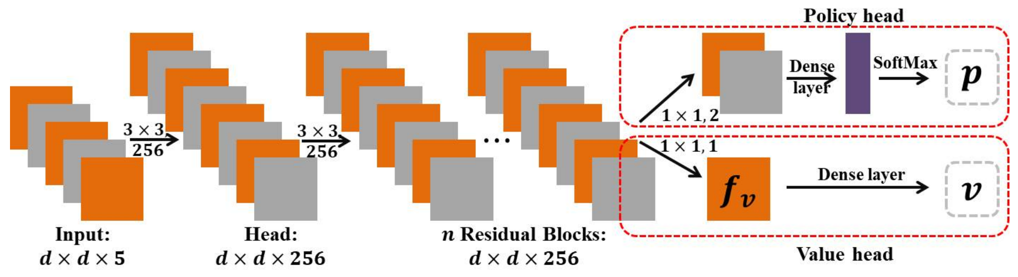

Figure 3. Detailed information about the architecture of policy-value network.

Algorithm 2 MCTS-PCZero self-play

```
**Input:** value network $f$ and policy $p$.
Initialize historical sequence $S = \emptyset$
Choose a random move as the beginning of the game
$s =$ Initial state
**while** $s$ is not termination state **do**
  **if** $|S| > l$ **then**
    Pop the earliest state
  **end if**
  $S = S \cup \{s\}$
  $H, a =$ HeuristicPath$(s, f, p)$
  $\bar{v}(s) = \frac{\sum_{s' \in S \cup H} f(s')}{|S|+|H|}$
  $s = s$ plays action $a$
**end while**
```

Algorithm 3 Heuristic Path

```
**Input:** state $s$, value function $f$ and policy $p$.
Initialize heuristic sequence $H = \emptyset$
Perform MCTS with $N_k$ simulation times
Sample action $a$ according to $N(s, a)$
**while** $s$ is not leaf node and $|H| \leq k$ **do**
  action$= arg\,max_{a' \in \mathcal{A}} N(s, a')$
  $s = s$ plays action
  $H = H \cup \{s\}$
**end while**
```

3.2. Feature consistency

We further extend path consistency to the feature maps of the value network. In the AlphaZero framework, the policy and the value network share a common residual tower to extract information from the board state. The value head performs convolution operation with one filter, yielding a feature map $\mathbf{f}_v$ before the final output layer. Specifically, the output v is the linear combination of the entries of $\mathbf{f}_v$ followed by a activation function σ, i.e., $v(s) = \sigma(Tr(\mathbf{w}^T \mathbf{f}_v))$, where $\mathbf{w}$ is the weight matrix of the same size as the feature map $\mathbf{f}_v$ and Tr denotes the trace operator of a matrix. In addition to maintaining the consistency on a one-dimensional value v, PC constraint is proposed to impose on the high-dimensional feature map $\mathbf{f}_v$:

$$L^f_{PC} = \|\mathbf{f}_v - \bar{\mathbf{f}}_v\|^2, \tag{4}$$

where $\bar{\mathbf{f}}_v$ is the element-wise average of all the feature maps over the nodes within the window. It can be noted that the feature consistency is a sufficient but not necessary condition for value consistency. And it contains more information about the game situation. Thus L^f_{PC} is a tighter constraint and can be used as a supplement of L_{PC} in Equation 3.

3.3. Loss functions

Based on Equation 1, following learning loss is adopted to training the PCZero,

$$L_1 = -y^T \log p + (z - v)^2 + \lambda L_{PC} + \beta L^f_{PC} + c\|\theta\|^2, \tag{5}$$

where p and v is the policy and value which are predicted by the neural network, y is the one-hot action vector which labels the move from the data, $z \in \{-1, +1\}$ denotes the final result of the game, either lose or win, and λ and β are the non-negative coefficients to adjust the influence of PC in Equation 3 & 4 respectively. $||\theta||^2$ is $L2$ regularization term on parameters.

While self-play is activated, MCTS is implemented to scout different paths from the existing state. Then, a policy π is computed according to the number of visiting times of each child of the root node, which represents the current state. The next playing move is selected according to the policy π. The loss function in Equation 5 can be improved by replacing y with π, i.e.,

$$L_2 = -\pi^T \log p + (z - v)^2 + \lambda L_{PC} + \beta L^f_{PC} + c\|\theta\|^2, \tag{6}$$

because the effects of some moves are equivalent due to game's symmetry property or from a long-term perspective, and deterministic action selection by the one-hot vector might be harmful for exploration. And loss function with $\lambda = \beta = 0$ will be used to train the AlphaZero model.

The policy-value (PV) network is built as follows. A convolutional head extracts input information, following n residual blocks with ReLU activation and batch normalization. n is set as 3 in this paper. Then the extracted information is passed to a policy head and a value head respectively, giving final estimations. Input data consists of five binary planes, representing the state of the game board. The first three channels indicate the position of the pieces on the board, i.e. black stones, white stones, and empty points. The remaining two channels are used to indicate the current game player, i.e., one is for whether black plays while the other is for whether white plays. For the Hex game, each board's side is padded as MoHex-3HNN (Gao et al., 2018). Detailed network information is displayed in Figure 3.

While playing games with search player, actions are chosen by Policy Value MCTS (PV-MCTS). PUCT criterion in Equation 7 is adopted to select the leaf node by setting $c_{puct} = 1.5$ and the search procedure is the same with AlphaZero (Silver et al., 2018).

$$a = \underset{a}{argmax} \left\{ Q(s,a) + c_{puct} p(s,a) \frac{\sqrt{N(s)}}{1+N(s,a)} \right\}, \tag{7}$$

While doing self-play, Dirichlet noise is injected into nodes' prior probability as a source of randomness. The first N_r moves are selected proportionally to $[N(s,a)/N(s)]^{1/\tau}$, where N_r is sampled from an exponential distribution with a mean of $0.04 \times h^2$, and h denotes the board size. Action with maximum $N(s,a)$ is chosen after t steps until reaching the termination. And those random operations will be removed in test tournaments.

3.4. PC in stochastic games

In stochastic games, where the state transition is probabilistic, i.e., $s' \sim p(s'|s,a)$, Equation 2 does not hold and PC will not be strictly established. State value $v(s)$ is calculated over all possible following states, whereas the PC nature is only related with a specific path. As illustrated in Figure 4, neither $v(s_w)$ nor $v(s_l)$ is equal to $v(s)$.

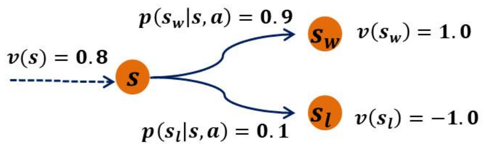

Figure 4. Example of PC nature in stochastic games.

Suppose K_s game experiences containing state s are collected. For kth sequence,

$$H_k : \{s_k^0, a_k^0, r_k^1, s_k^1, \cdots, r_k^N, s_k^N\}, \tag{8}$$

the difference of f value between s and s'_k is

$$\Delta_s^k = f(s) - f(s'_k) = v(s) - (r_k + v(s'_k)), \tag{9}$$

following the definition that $f(s_k^n) = g(s_k^n) + h(s_k^n) = \sum_{i=1}^n r_k^i + v(s_k^n)$. Applying Cauchy-Schwarz inequality on the deviation between s and s'_k for all K_s sequences, we have

$$\begin{aligned} \frac{1}{K_s}\sum_{k=1}^{K_s}(\Delta_s^k)^2 &= \frac{1}{K_s^2}\sum_{k=1}^{K_s}(\Delta_s^k)^2 \sum_{k=1}^{K_s} 1^2 \geq \frac{1}{K_s^2}\left[\sum_{k=1}^{K_s}\Delta_s^k\right]^2 \\ &= \left[v(s) - \frac{1}{K_s}\sum_{k=1}^{K_s}(r_k + v(s'_k))\right] \\ &= \{v(s) - E_{a,s'}[r(s,a) + v(s')]\}^2. \end{aligned} \tag{10}$$

The squared value deviation becomes the upper bound of the satisfaction of Bellman equation for the state-value function, which is $v(s) = E_{a\sim\pi, s'\sim p}[r(s,a) + \gamma v(s')]$, when setting $\gamma = 1.0$. Therefore, optimizing the PC constraint still has positive effect on maintaining Bellman temporal consistency in stochastic games.

4. Experiment

Taking Hex, Othello, and Gomoku as examples, the advantage of PCZero will be investigated in both offline and online learning. Three board sizes of Hex are considered, including 8×8, 9×9 and 13×13, while the sizes of Othello and Gomoku are 8×8 and 15×15 respectively.

4.1. Effectiveness of PC on online learning

To illustrate the learning improvement brought by PC, we first compare models for online learning trained with the same self-play games. Firstly, an AlphaZero model (Silver et al., 2018) is trained for 13×13 Hex. During the self-play, MCTS runs 400 simulations to select moves and 1000 games are played in each iteration. For the first 200 epochs, the learning rate r is 0.01 and temperature parameter τ is 0.8. In the following 200 epochs, $r = 0.001$ and $\tau = 0.6$. For the rest 500 epochs, $r = 0.0001$ and $\tau = 0.2$. $900K$ games are generated to train the AlphaZero model in total, which will also be used to update our PCZero model with the same learning hyperparameter setting except $\lambda = 3.0$ and $l = k = 5$. Typically, it takes humans 0.5 hours to finish a 13×13 Hex game. Assuming a person studies 12 hours a day, $900K$ is roughly the amount a human can study in eight decades. The collected games is then used to update network parameters, taking Equation 6 as loss function by setting $\lambda = \beta = 0$ and $c = 10^{-5}$ for AlphaZero model. We use 8 GeForce RTX 2080Ti GPU and Intel(R) Xeon(R) Gold 6130 CPU with 125G RAM to do self-play. A single GTX 1050Ti GPU and Intel i7 8750H CPU with 16 GB RAM are used to test. MoHex

2.0, MoHex-CNN and MoHex-3HNN are the champion of Hex Computer Olympiad in 2015 (Hayward et al., 2015), 2017 (Hayward & Weninger, 2017) and 2018 (Gao et al., 2019) respectively and their game abilities exceed most human players. Tournaments are conducted to evaluate the performance of models, in which MoHex 2.0[6] is used as the baseline and players have 10s to select the position to place their stone. 338 games are played for each model competition. It is worth noting that simulation times of our programs are only $2\% \sim 5\%$ of MoHex programs, due to the difference in the implementation language.

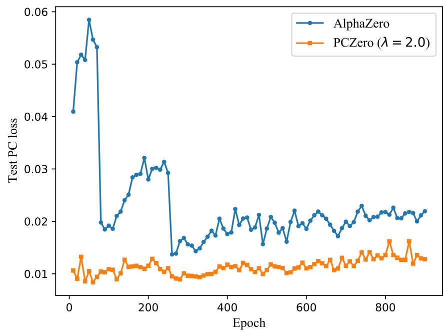

Figure 5. Test PC loss on expert dataset for online learning.

Results in Table 1 shows that our PCZero model outperforms not only state-of-the-art Hex players, but also the AlphaZero model greatly. What's more, PCZero trained after $600K$ games has defeated the final AlphaZero model trained after $900K$ games with a $175 : 163$ score, indicating the efficiency brought by PC constraint. The expert dataset in Section 4.3 is used to gather test information of the model trained with self-play games. As shown in Figure 5, test PC loss of AlphaZero also decreases, but is higher than PCZero. It suggests that path consistency is a nature required for strong value predictors and the term of PC loss guides the learning process towards it proactively providing efficiency. The same experiment is also conducted on Othello. Using 3-ply Edax as the baseline model, winning rate of our PCZero ($\lambda = 2.0$) is 74.8% exceeding AlphaZero's 69.5%, while playing with MCTS player.

If the PC in terms of feature consistency is activated by setting $\beta > 0$, the winning rate against MoHex 2.0 is further improved to 94.1% in 13×13 Hex. What's more, trained after $900K$ games, PCZero ($\beta = 0$) beats AlphaZero with a 58.3% winning rate and PCZero considering only feature consistency ($\lambda = 0$) obtains a 56.8% winning rate. If both consistencies are activated, winning rate against AlphaZero achieves 63.3%. Feature consistency is a positive supplement for value path consistency.

Table 1. Winning rate of models against MoHex 2.0 (10s) in 13×13 Hex, trained with $0.9M$ selfplay games.

Model	As Black	As White	Overall
MoHex-CNN	78.6%	61.2%	69.9%
MoHex-3HNN	/	/	82.4%
AlphaZero	89.3%	79.3%	84.3%
PCZero ($\beta = 0$)	**96.4%**	90.5%	93.5%
PCZero	94.7%	**93.5%**	**94.1%**

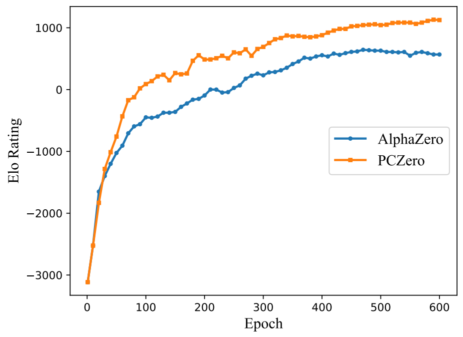

Figure 6. Elo ratings for 8×8 Hex without data sharing.

Besides, experiments without sharing the generated data are conducted on 8×8 Hex. All models are trained with the self-play games generated by themselves. The ability improvement during training is evaluated by the Elo rating system[7], in which models with higher rating scores have the stronger game capability. Models have been sampled every 10 training epoch and tournaments for each opening are played twice between MCTS players of the current sampled model and the previous sampled model. Game results will be collected for scoring. As shown in Figure 6, the learning curve of PCZero grows faster and maintains at a higher ratings score than AlphaZero. For the final model, PCZero outperforms AlphaZero with a $78 : 50$ tournament score. These results indicate that PC constraint brings not only learning efficiency but also improvements on players' final performance for online reinforcement learning.

4.2. Investigation of heuristic path

MCTS-PCZero in Algorithm 2 is implemented on 8×8 Hex. Programs are trained individually without data sharing. During self-play, MCTS runs 200 simulations and 500 games are generated for each iteration. $r = 0.01, 0.001, 0.0001$

[6]https://github.com/cgao3/benzene-vanilla-cmake

[7]https://www.remi-coulom.fr/Bayesian-Elo/

Table 2. Tournament results among MCTS-PCZero with different window length for 8×8 Hex (row player against column player).

	$l=5, k=0$	$l=0, k=5$	$l=5, k=5$
$l=0, k=5$	53.1%	/	/
$l=5, k=5$	57.0%	60.9%	/
$l=5, k=\infty$	57.8%	56.8%	50.8%

and $\tau = 0.8, 0.6, 0.2$ for the first 200 epochs, middle 200 epochs and last 200 epochs separately. Each model is trained with $300K$ self-play games in total. The learning process is reported in Figure 7 by Elo rating system. Compared with regular PCZero, MCTS-PCZero converges faster to a better solution, obtaining higher Elo rating scores. What's more, MCTS-PCZero learning after 500 epochs has already played against the final PCZero with 50.0% winning rate.

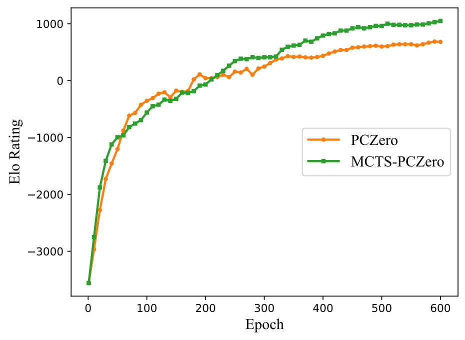

Figure 7. Elo ratings of MCTS-PCZero for 8×8 Hex.

The influence of historical path and heuristic path is investigated by setting $l = 5, k = 0$ and $l = 0, k = 5$ respectively. Tournament results is listed in Table 2. Programs may benefit more from the heuristic path than the historical path because the model trained with the former wins more games. And the performance is the best when both are included in the selected window. The importance of randomness during learning has been demonstrated in AlphaZero reproduction programs (Wu, 2019). Therefore, if the heuristic path is considered more by setting $k = \infty$, which means scouting until the leaf node without explicit length limit, model's performance has been further improved, beating AlphaZero model and MCTS-PCZero ($l = k = 5$) with 55.5% and 50.8% winning rate respectively. Poor window selection provides a negative impact by giving a $\bar{v}$ deviating from the optimal path greatly. The heuristic path is the source of deviation and the historical path aims to maintain W_s in the optimal path. The relative length of the heuristic path and the historical path is similar to the classical exploration and exploitation trade-off. Both parts are beneficial but the heuristic path is more important. Larger k brings more efficiency, cooperating with appropriate l.

Besides 8×8 Hex, the experiment is also conducted on the more complicated 13×13 Hex. An intermediate model updated after $750K$ self-play games during the training of AlphaZero is adopted as the initial model to continue to be trained as AlphaZero, MCTS-PCZero ($l = k = 5$), and MCTS-PCZero ($l = 5, k = \infty$) separately. After training with $150K$ self-play data generated independently without sharing, MCTS-PCZero ($l = 5, k = \infty$) outperforms AlphaZero and MCTS-PCZero ($l = k = 5$) with 51.8% and 53.0% winning rates. For a well-trained intermediate model, MCTS-PCZero obtains a better performance because the exploratory brought by the heuristic path helps to escape local optimum.

4.3. Effectiveness of PC on offline learning

Besides online reinforcement learning, we will show that PC also benefits the offline learning process. There are three residual blocks in the network, of which parameters are updated by Equation 5. And the window length is set to be $l = k = 5$. For Hex, expert dataset is collected by the self-play of MoHex 2.0 (Gao et al., 2018), containing $50K$, $101K$ and $18K$ games for 8×8, 9×9 and 13×13 Hex respectively. WThor[8] and RenjuNet[9] are adopted as the expert dataset for Othello and Gomoku, containing $126K$ and $70K$ games respectively. Those datasets are divided into training set and test set randomly and the proportion of test set is 20%. While conducting test tournaments, the offline AlphaZero model ($\lambda = \beta = 0$) is used as baseline. For Hex game with $h \times h$ boardsize, there are h^2 opening cells and one game is played for the first and second players for each opening. A total of $2h^2$ games are played. For Othello, 100 openings are randomly generated according (Runarsson & Lucas, 2014) and 200 openings with 8 placed stones are sampled for Gomoku. While playing test tournaments with MCTS player, simulation times are 1600 for Hex, 800 for Othello and Gomoku. For greedy player, actions with maximum policy probability will be chosen.

Winning rate of our offline PCZero against offline AlphaZero is listed in Table 3, in which λs are all set as 2.0. For tournaments between greedy players, offline PCZero can beat offline AlphaZero with a winning rate slightly exceeding 50.0% for all games. Because policy head and value head share the same residual tower extracting information, policy learning can also benefit from the constraint on estimated values. For tournaments between MCTS players, the advantage of our offline PCZero over offline AlphaZero

[8] `https://www.ffothello.org/informatique/la-base-wthor/`

[9] `http://www.renju.net/downloads/games.php`

Table 3. Winning rate of offline PCZero against offline AlphaZero at $\lambda = 2.0, \beta = 0.0$.

GAME	GREEDY PLAYER	MCTS PLAYER
HEX (8×8)	51.6%	58.6%
HEX (9×9)	53.1%	59.9%
HEX (13×13)	52.1%	61.5%
OTHELLO	50.5%	80.5%
GOMOKU	56.8%	64.0%

is more significant. In Equation 7, $Q(s, a)$ is the mean action value, controlling exploitation during the search. If the value network gives inaccurate estimations about $v(s)$, MCTS will be misled and poor moves will be selected. As displayed in Figure 8 & 9, value loss of offline PCZero is greater than offline AlphaZero during training, but less than offline AlphaZero while testing. The reason is that the PCZero slightly sacrifices the prediction accuracy on values to meet the constraint of path consistency during training. And the satisfaction of the latter captures more valuable information, improving the model's generalization ability. Therefore, backup values provided by PCZero is more reasonable, enhancing the search quality of MCTS player greatly.

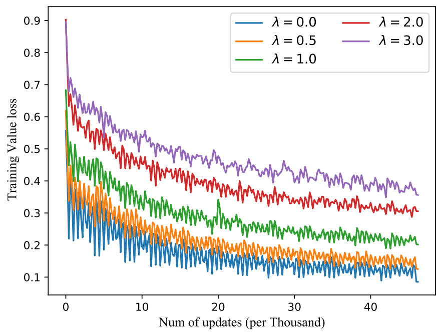

Figure 8. Training value loss on 13×13 Hex for different λ.

The expression of value loss $(z - v)^2$ is similar with L_{PC}, both of which are related with value estimation. Take γ as the weight of value loss and above models are trained by keeping $\gamma = 1.0$ in Equation 5. Experiments are conducted about the influence of γ by keeping $\lambda = \beta = 0$ and offline AlphaZero with $\gamma = 1.0$ is used as the baseline model. Results in Table 4 show that increasing the weight of value loss can not achieve the same effect as adding the PC constraint. Especially, the winning rates of offline PCZero in the tournaments among MCTS players are much higher than the winning rates among greedy players in Table 3. However, the winning rates of MCTS player with larger γ are even

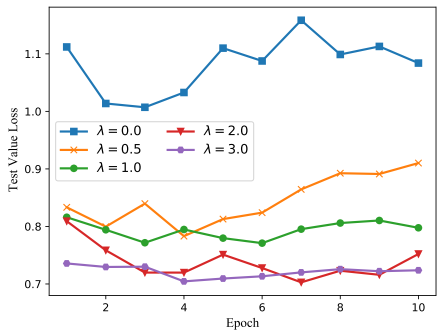

Figure 9. Test value loss on 13×13 Hex for different λ.

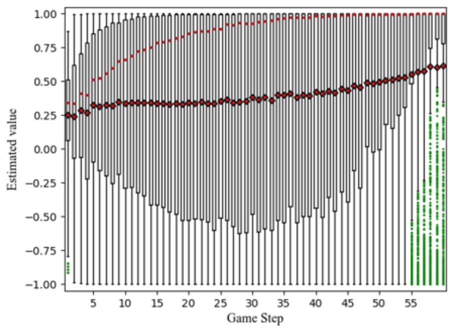

Figure 10. Distribution of AlphaZero's estimated values for 13×13 Hex at each game step ($z = 1$).

worse than the greedy player, for which the model is too concerned about the accuracy of the prediction, weakening the generalization ability and giving wrong value estimation feedback for MCTS. What's more, PC loss $(v - \bar{v})^2$ is trying to reduce the prediction variance along the path, while value loss $(v - z)^2$ is more concerned with reducing prediction bias. Taking 13×13 Hex as an example, PCZero's standard deviation of the estimated values along the path is 0.259, averaged on all test samples with $z = +1$, much smaller than AlphaZero's 0.550, as shown in Figure 10 & 11. Therefore, larger γ cannot improve the performance of the final MCTS player and it is worthwhile to introduce L_{PC} into the loss function to incorporate with value loss.

Feature consistency in offline case is also investigated. Using offline AlphaZero as baseline model, results for 13×13 Hex in Table 5 show that the winning rate is improved from 61.5% to 70.7% for MCTS player with 1600 simulations by switching on feature consistency at $\beta = 1.0$. If only considering feature consistency at $\lambda = 0$, the winning rate is 50.3%, which is consistent with the results in online learning case in Section 4.1. The same experiment is also conducted

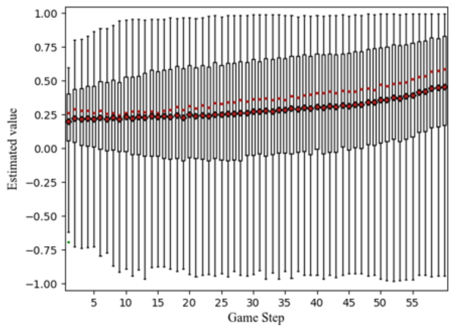

Figure 11. Distribution of PCZero's estimated values for 13×13 Hex at each game step ($z = 1$).

Table 4. Winning rate of offline AlphaZero with different γ against offline AlphaZero with $\gamma = 1.0$.

GAME	γ	GREEDY PLAYER	MCTS PLAYER
HEX (13×13)	2.0	48.8%	45.9%
HEX (13×13)	3.0	55.9%	55.0%
OTHELLO	2.0	49.2%	34.8%
OTHELLO	3.0	42.8%	42.8%

for Othello and Gomoku. Trained with the expert dataset, the winning rate of offline PCZero against offline AlphaZero is improved to 83.0% for Othello by setting $\beta = 0.5$, and 69.0% for Gomoku by setting $\beta = 1.0$ for the competition between MCTS players. Therefore, value path consistency takes the main role in performance improvement, while feature consistency is a positive supplement.

5. Conclusion

In this paper, we have proposed PCZero, a data-efficient learning algorithm, based on the PC optimality condition. PCZero is more efficient in learning than AlphaZero in board games like Hex, Othello, Gomoku. In particular, PCZero can defeat AlphaZero, and significantly improve the winning rate against the Hex Champion from 84.3% to 94.1%, in comparisons with AlphaZero, by training on a small amount of self-play data that is human-achievable in a lifetime. This is realized by an effective implementation of PC that includes both historical searching trajectories and MCTS's lookahead simulated paths, striking a good balance between exploitation and exploration. Ablation studies examine the effectiveness of the proposed components in PCZero. Experiments further demonstrate that the offline version of PCZero is also effective in performance improvement when learning from a fixed dataset. PC is helpful in consuming fewer computational resources but obtaining strong artificial intelligence programs.

Table 5. Winning rate of different offline PCZero against offline AlphaZero for 13×13 Hex.

λ	β	GREEDY PLAYER	MCTS PLAYER
2.0	0.0	52.1%	61.5%
0.0	1.0	50.6%	50.3%
2.0	1.0	**53.3%**	**70.7%**

In the future, we plan to study the PCZero framework on the real applications in combinatorial nature. Moreover, CNneim-A (Xu et al., 1987) utilized path consistency to assist the search process firstly, guaranteeing to find the optimal solution with $O(N \log N)$ complexity. Therefore, PC is not an ad hoc heuristic, and there is room for theoretical studies in the future. In this paper, A^* search is replaced by MCTS to make the lookahead scouting. A comparison of these two methods will be finished in future work. What's more, the calculation of $\bar{v}$ relies on the selection of the window, which has fixed length in this paper. As illustrated in CNneim-A (Xu et al., 1987), the length of historical path should increase with the depth of current state to control $\bar{v}$'s estimation bias. Therefore, window selection will be further studied in our following work.

Acknowledgement

This work was supported by National Science and Technology Innovation 2030 Major Project (2018AAA0100700) of the Ministry of Science and Technology of China, and NSFC (62172273), and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

References

Anshelevich, V. V. A hierarchical approach to computer hex. *Artificial Intelligence*, 134(1-2):101–120, 2002.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv:1705.08439*, 2017.

Arneson, B., Hayward, R., and Henderson, P. Wolve wins hex tournament. *ICGA Journal*, 32(1):49–53, 2008.

Arneson, B., Hayward, R. B., and Henderson, P. Monte carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258, 2010.

Even, S. and Tarjan, R. E. A combinatorial problem which is complete in polynomial space. *Journal of the ACM (JACM)*, 23(4):710–719, 1976.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine,

S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Gao, C., Hayward, R., and Müller, M. Move prediction using deep convolutional neural networks in hex. *IEEE Transactions on Games*, 10(4):336–343, 2017.

Gao, C., Müller, M., and Hayward, R. Three-head neural network architecture for monte carlo tree search. In *IJCAI*, pp. 3762–3768, 2018.

Gao, C., Takada, K., and Hayward, R. Hex 2018: Mohex3hnn over deepezo. *J. Int. Comput. Games Assoc.*, 41 (1):39–42, 2019.

Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Hayward, R. and Weninger, N. Hex 2017: Mohex wins the 11x11 and 13x13 tournaments. *ICGA Journal*, 39(3-4): 222–227, 2017.

Hayward, R., Pawlewicz, J., Takada, K., and van der Valk, T. Mohex wins 2015 hex $11\times$ 11 and hex $13\times$ 13 tournaments. *J. Int. Comput. Games Assoc.*, 39(1):60–64, 2015.

Huang, S.-C., Arneson, B., Hayward, R. B., Müller, M., and Pawlewicz, J. Mohex 2.0: a pattern-based mcts hex player. In *International Conference on Computers and Games*, pp. 60–71. Springer, 2013.

Iwata, S. and Kasai, T. The othello game on an $n\times$ n board is pspace-complete. *Theoretical Computer Science*, 123 (2):329–340, 1994.

Krol, D. and Brandenburg, J. v. *Q-learning adaptations in the game Othello*. PhD thesis, 2020.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Liskowski, P., Jaśkowski, W., and Krawiec, K. Learning to play othello with deep neural networks. *IEEE Transactions on Games*, 10(4):354–364, 2018.

Lucas, S. M. and Runarsson, T. P. Temporal difference learning versus co-evolution for acquiring othello position evaluation. In *2006 IEEE Symposium on Computational Intelligence and Games*, pp. 52–59. IEEE, 2006.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Bridging the gap between value and policy based reinforcement learning. *arXiv preprint arXiv:1702.08892*, 2017a.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Trust-pcl: An off-policy trust region method for continuous control. *arXiv preprint arXiv:1707.01891*, 2017b.

Reisch, S. Gobang ist pspace-vollständig. *Acta Informatica*, 13(1):59–66, 1980.

Runarsson, T. P. and Lucas, S. M. Preference learning for move prediction and evaluation function approximation in othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):300–313, 2014.

Schaul, T. and Schmidhuber, J. A scalable neural network architecture for board games. In *2008 IEEE Symposium On Computational Intelligence and Games*, pp. 357–364. IEEE, 2008.

Shao, K., Zhao, D., Tang, Z., and Zhu, Y. Move prediction in gomoku using deep learning. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pp. 292–297. IEEE, 2016.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Sutton, R. S., Barto, A. G., et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Takada, K., Iizuka, H., and Yamamoto, M. Computer hex algorithm using a move evaluation method based on a convolutional neural network. In *Workshop on Computer Games*, pp. 19–33. Springer, 2017.

Takada, K., Iizuka, H., and Yamamoto, M. Reinforcement learning to create value and policy functions using minimax tree search in hex. *IEEE Transactions on Games*, 12 (1):63–73, 2019.

Tang, Z., Zhao, D., Shao, K., and Le, L. Adp with mcts algorithm for gomoku. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–7. IEEE, 2016.

Wu, D. J. Accelerating self-play learning in go. *arXiv preprint arXiv:1902.10565*, 2019.

Xu, L. Deep bidirectional intelligence: Alphazero, deep ia-search, deep ia-infer, and tpc causal learning. In *Applied Informatics*, volume 5, pp. 1–38. Springer, 2018.

Xu, L., Yan, P., and Chang, T. Algorithm cnneim-a and its mean complexity. In *Proc. of 2nd international conference on computers and applications. IEEE Press, Beijing*, pp. 494–499, 1987.

Young, K., Vasan, G., and Hayward, R. Neurohex: A deep q-learning hex agent. In *Computer Games*, pp. 3–18. Springer, 2016.

Yu, Y. Towards sample efficient reinforcement learning. In *IJCAI*, pp. 5739–5743, 2018.

Zhao, D., Zhang, Z., and Dai, Y. Self-teaching adaptive dynamic programming for gomoku. *Neurocomputing*, 78 (1):23–29, 2012.