Efficient Learning for AlphaZero via Path Consistency

Group 11: 謝振杰、陳冠廷、郭育誠

Introduction

Introduction

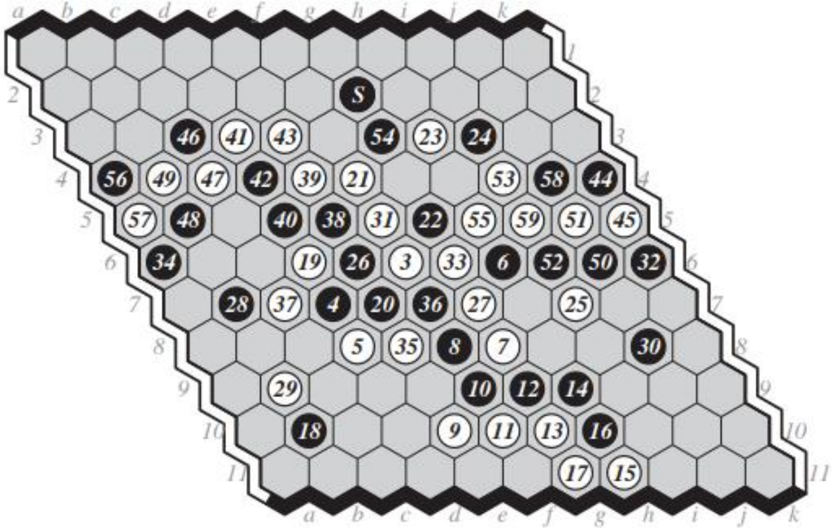
- Deep RL made a great breakthroughs on board games
  - AlphaGO, AlphaZero
- Problems of AlphaZero
  - Huge computational resources
  - Highly depends on the number of self-play games
- Solutions
  - Add **path consistency** on AlphaZero objective

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

Related Works

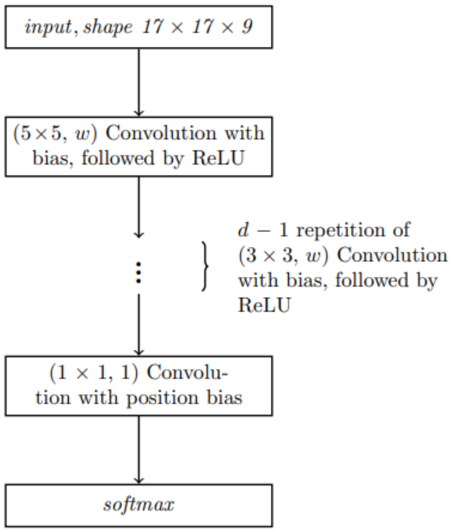
Related Works

- Hex Game
    - Connects the player’s two sides of the board

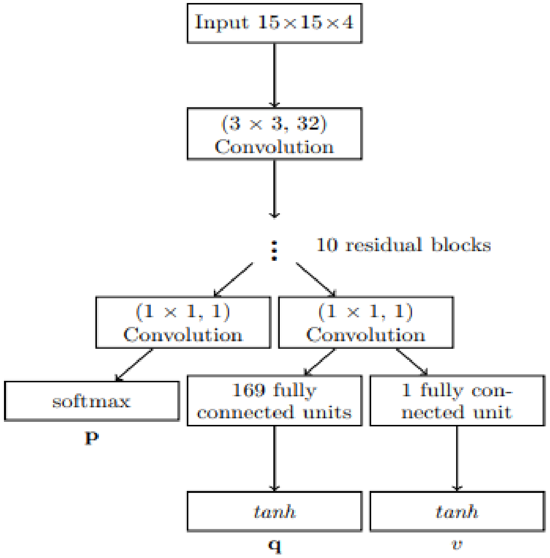


Related works

- Two baseline methods on Hex Game



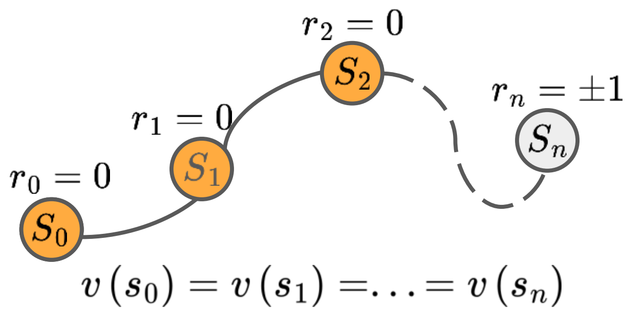
MOHEX-CNN



MOHEX-3HNN

Method

Why PC constraint is reasonable?



- **Board games** are discussed in this paper
- Imediate reward $r(s,a) = 0$ and $g(s) = 0$ until termination state
- => $v(s) = h(s)$
- When optimal playing policy and state transition function are deterministic
- $v^*(s) = r(s, \pi^*(s)) + \gamma v^*(s')$, set $\gamma = 1$
- => $v^*(s) = v^*(s')$
- Any two neighboring nodes should have identical state values $\boldsymbol{v}$

PC computation with historical paths

- In AlphaZero, $v(s)$ is computed by a neural network
- In PCZero, estimated $v(s)$ should be consistent along the optimal path
- Variance of the values within a sliding window W_s is added to loss function

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

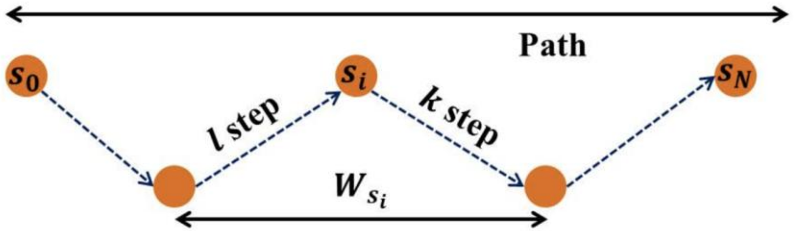
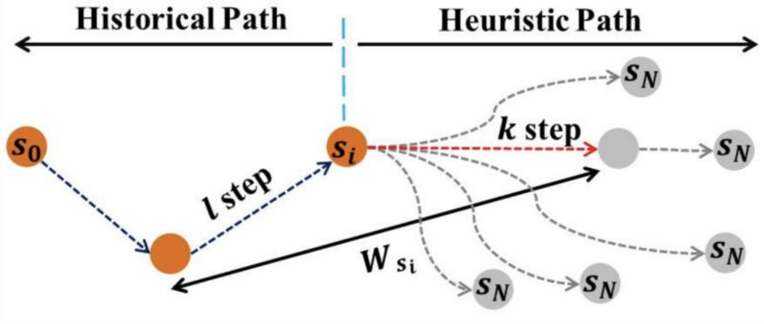


Figure 1. $\bar{v}$ calculation with historical path in a terminated game.

- This method converge to a local optimum quickly

PC computation with historical and heuristic paths

- To prevent local optimum problem, calculate $\bar{v}$ with:
  - Upstream historical trajectory
  - Downstream **heristic path provided by MCTS** (using visit count)



- MCTS selected nodes may differ from historical nodes, but such variance rescue the network from a local optimum
  (similar to the trade-off between exploration and exploitation)

Feature Map Consistency

- PC can also be applied to the feature maps f_v of the value network $L_{PC}^{f} = \|\mathbf{f}_v - \bar{\mathbf{f}}_v\|^2$, $\bar{f}_v$ is the element-wise average of the feature maps within the sliding window
- A tighter constraint, used as a supplement

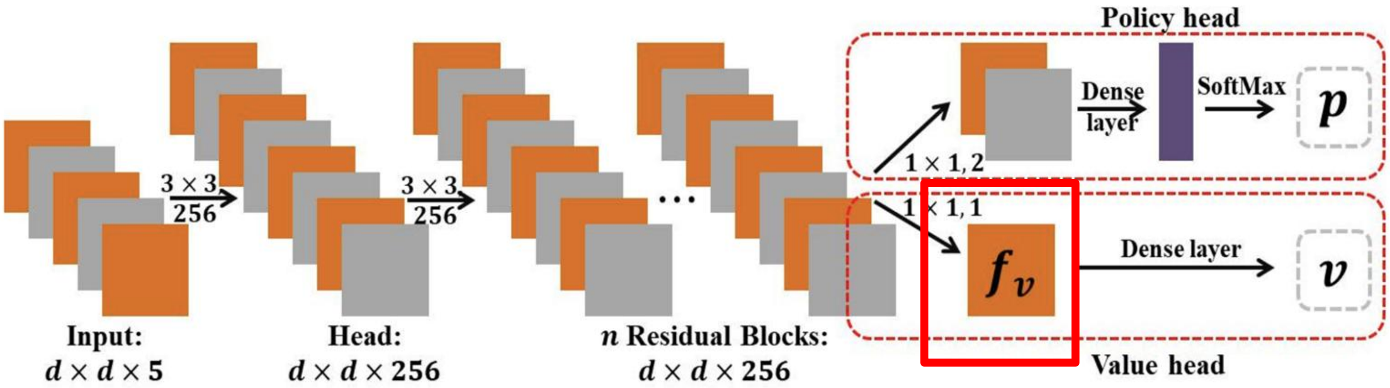


Figure 3. Detailed information about the architecture of policy-value network.

Loss function

In the loss function, we will add L_{PC} to enforce path consistency, ensuring that the value estimations for neighboring states remain consistent.

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

- y : One hot vector of move
- p :policy
- v :value
- z :-1 or 1 denote the final result of game
- $||\theta||^2$: L2 regularization
- λ, β : non-negative coefficients
- $L_{PC}(s) = (v - \bar{v})^2$
- $L_{PC}^f = ||\mathbf{f}_v - \bar{\mathbf{f}}_v||^2$

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

- π : Target policy

How do MCTS choose action?

The first N_r moves : $\left[N\left(s,a\right)/N\left(s\right)\right]^{1/\tau}$

- N_r is sampled from an exponential distribution with a mean of $0.04 \times h^2$, and h denotes the boardsize.
- $N\left(s,a\right)$:The count of choosing action a in state s
- $N\left(s\right)$:The number of occurrences of state

After first N_r moves :

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

- C_{puct} : Balances exploration and exploitation

PC in stochastic games

$$\Delta_s^k = f(s) - f(s'_k) = v(s) - (r_k + v(s'_k))$$

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

- The squared value deviation becomes the upper bound of the satisfaction of Bellman equation for the state-value function

Experiment

Performance on Hex Game

- Overall performance on Hex: the best
- MoHex 2.0 as the baseline and players have 10s to select the position
- **338 games** are played for each model competition

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**%

Compared with AlphaZero

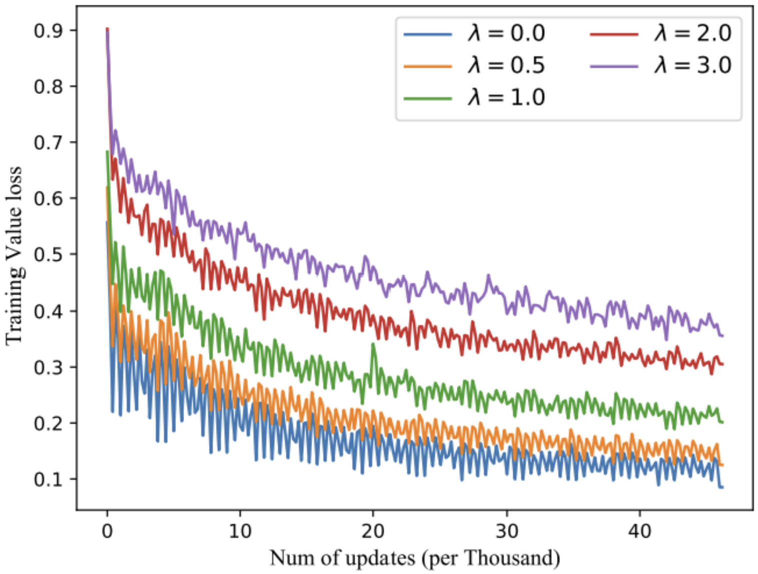
- Offline training on fixed dataset

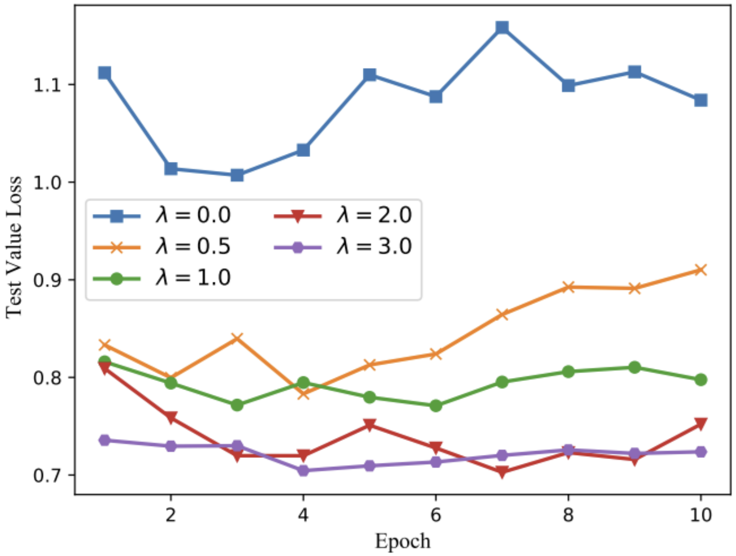
Table 3. Winning rate of offline PCZero against offline AlphaZero at $\lambda = 2.0, \beta = 0.0$.

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%

Ablation Study

- **lambda** (the weight of PC Loss)
- AlphaZero generalization is worse than PCZero





Conclusion

Conclusion

Solution

- Add **path consistency loss** to guide the model training

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

- Historical searching trajectories and MCTS's lookahead simulated paths

Result

- PCZero significantly improves the win rate on Hex Game
- Training with only small amount of self-play data (900K)
- Offline learning is still effective in performance improvement

Thank You