Journey to Learn Omok

lumiknit

Intro.

Goal

- Omok! (=Connect 5 / Renju / Gomoku)
- Can we train an agent PLAYING OMOK?
- Can we make an agent OVERWHELMING some opponent agent?

Sparring Partner: Algorithm

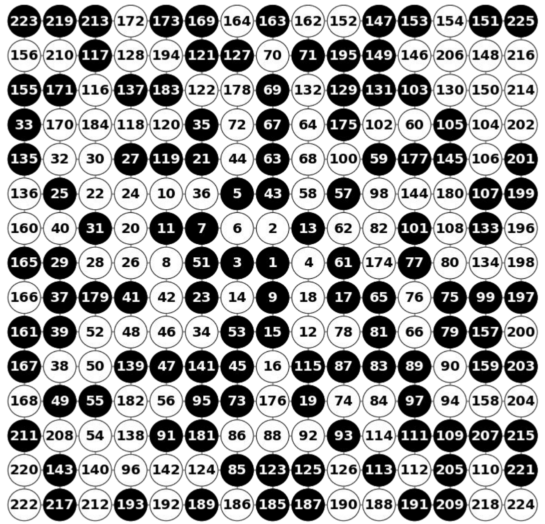
- Omok agent must choose winning movements.
- Intuitively, longer connections give more chance to win.
- And some shape of stones assure that the game is DONE.
- e.g. open-4 (4-connection and both side is not blocked by opponent, the player with open-4 must win in the his/her next turn), open-3 (3-connection which can be open-4 with one more stone. The player with open-3 must win if the opponent does not block them.)

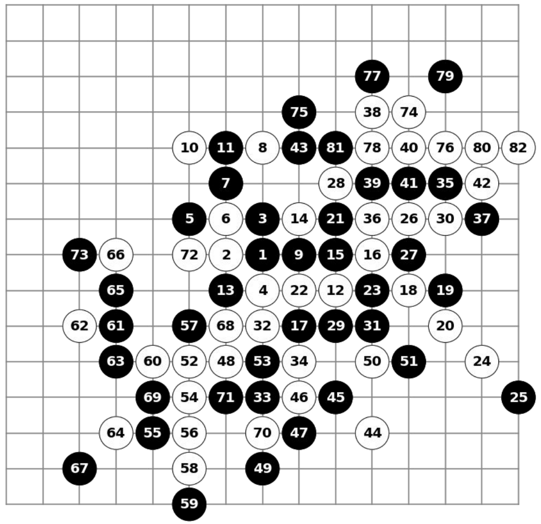
Sparring Partner: Algorithm (Cont.d)

- How to choose BEST movement? => Calculate the value of each cell by the expected connection length!
- `mock5.Analysis` scans whole Omok board, and find the connection length and open-ness of each cell.
- If the movement makes longer connection, give more score.
- If the movement leads to finish game, give more score.
- The value of each cell is the sum of scores of each direction (horizontal, vertical and two diagonal directions.)
- And choose almost greedy action! (Choose the greatest score cell.)
- This is VERY SIMPLE to implement!

Sparring Partner: Algorithm (Cont.d)

- Ok, then does it work well?
- Not very powerful, but FAR REASONABLE than random movements.
- Algorithm v.s. Algorithm examples



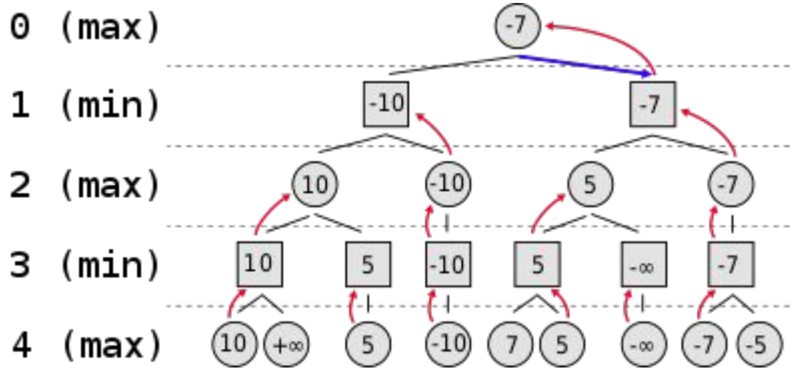


Better Way to Think Movements

- To beat the algorithm down, we need better algorithm/strategy/thinking.
- Most of classic algorithms for Omok is not such simple.

Tree Search

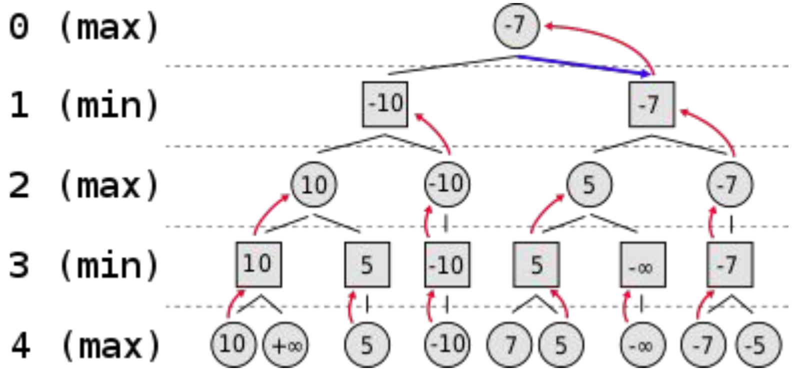
- The most successful agent uses tree search, such as Monte Carlo Tree Search or Min-Max Tree.
- The idea is just traverse ALL POSSIBLE movements.
- Tree is introduced to store and query the accumulated information.



https://en.wikipedia.org/wiki/Minimax

Tree Search (Cont.d)

- However, our interesting problems have too many states and movement.
- It means, the tree is TOO big (maybe bigger than the space!)
- There are many algorithm to reduce the search space: alpha-beta pruning, proof number search, treat-space search.
- Monte Carlo method provides finite time termination by sacrificing precision.



https://en.wikipedia.org/wiki/Minimax

AlphaGo / AlphaZero

- AlphaGo and AlphaZero basically uses Monte Carlo Tree Search (MCTS).
- But calculating whole MCTS in naive way takes too long time!
- Therefore, they choose actions by SOME ESTIMATOR instead of random,
- and calculate win rate by SOME ESTIMATOR instead of play games at the end.
- These ESTIMATORs are implemented by parametric models: neural networks.
- i.e. Alpha- is a tree search algorithm accelerated by neural networks.
- Then, what are the networks?

Reinforcement Learning

- Reinforcement learning (RL) is ‘training an AGENT which look the STATE and take an ACTION, in the ENVIRONMENT which gives a REWARD for each actions.’.
- The goal is maximizing a RETURN which is a sum of (decayed) rewards.

Reinforcement Learning (Cont.d)

- To formularize agent, we use policy function π: S*A->[0,1], which takes a state and an action and give the probability of the action chosen in the state.
- For fixed policy, we can calculate a state-value function v: S->R and an action-value function q: S*A->R, which is the expectation of return for the state and action.
- The ultimate goal of RL is ‘obtaining optimal policy function π*’ which give maximum return,
- and this policy choose the action maximizing the expectation of return, which can be represented by ‘the state-value function q* of π*’
- Note that policy give value, and value induces policy.

Reinforcement Learning (Cont.d)

- Therefore, in this sense, RL is just an approximation of an optimal policy function for the given environment!
- However, if there are too many states and actions, it’s hard to make/calculate/use policy and value functions.
- Therefore, we use parametric models, such as neural networks,
- and we use the ‘policy network’ and ‘value network’ sometimes.

Two Approaches to Approx. Policy

- Value-based approach approximates value function (state-value function v, or action-value function q), then, induces a policy function by applying argmax on value function.
- e.g. Policy Iteration, SARSA, Q-Learning
- Policy-based approach approximates policy function directly.
- e.g. Policy Gradient, Actor-Critic, A2C

Is Omok Easy to Apply RL?

- NOT AT ALL.
- For 15x15 board (Renju Rule), there are at most 225! ~= 1.25 * 10^433 states, and at most 225 possible actions!
- For 19x19 board, the number of states and actions match to the ones of Baduk(Go).
- However, it shows that any other precise analysis are almost impossible too.
- Thus, policy learning is hard but worth to try.

Are There Any Other EASIER Games?

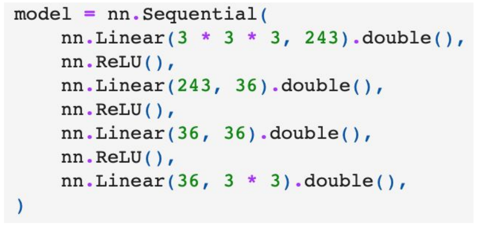
- Omok is one of (m,n,k)-game.
- There are a lot of (m,n,k)-game: Tic-Tac-Toe and Connect 4.
- Tic-Tac-Toe is a 3-connecting game on 3*3 board. Only 5478 states and at most 9 possible actions. (Even you can use BRUTEFORCE.)
- Connect 4 is 4-connecting game on 6*7 board, where each stone falls down.
- The board is even smaller than half of Omok’s one, but there are a lot of states. (About 1.4*10^31 ~= 1.38*2^103)
- However, only 7 actions are possible.

- They are good to make a pre-test for RL method!

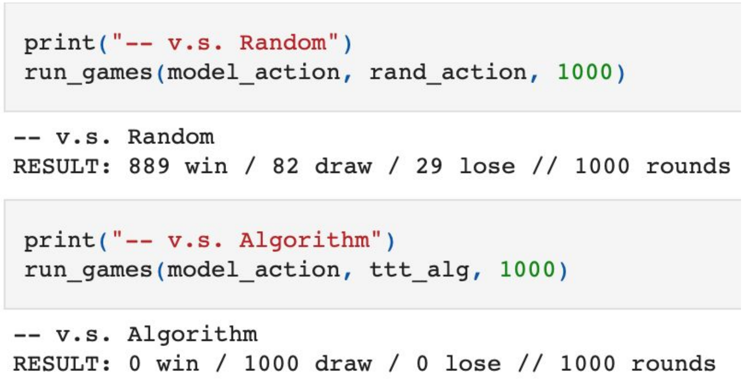
Attempts and Results

Warminig-up with Tic-Tac-Toe

- Tic-Tac-Toe is very easy case to approach ANY algorithms and method.
- Even if we traverse all possible way, it does not take 0.01 second.
- Only with 4 linear layers and ReLU, you can train the q-network with 20k games.
- Also, you can CONSTRUCT almost correct Monte Carlo Tree in one or two minutes. Then, you can use all tree search techniques such as minimax, or alpha-beta pruning, etc.



```
model = nn.Sequential(
    nn.Linear(3 * 3 * 3, 243).double(),
    nn.ReLU(),
    nn.Linear(243, 36).double(),
    nn.ReLU(),
    nn.Linear(36, 36).double(),
    nn.ReLU(),
    nn.Linear(36, 3 * 3).double(),
)
```



```
print("-- v.s. Random")
run_games(model_action, rand_action, 1000)
```

```
-- v.s. Random
RESULT: 889 win / 82 draw / 29 lose // 1000 rounds
```

```
print("-- v.s. Algorithm")
run_games(model_action, ttt_alg, 1000)
```

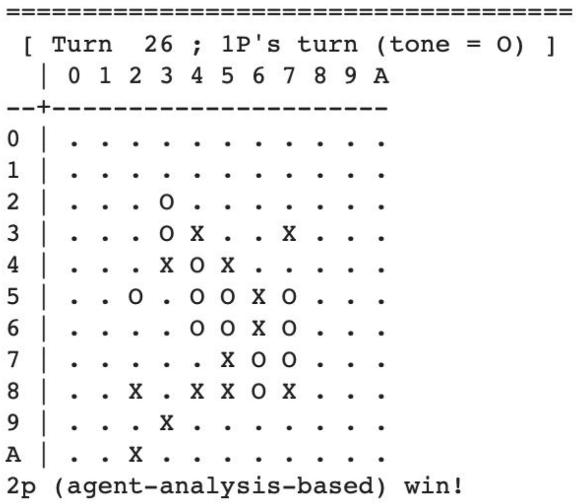
```
-- v.s. Algorithm
RESULT: 0 win / 1000 draw / 0 lose // 1000 rounds
```

Value-based Approach with Omok

- Value function gives an expectation of sum of rewards.
- Omok gives a reward ONLY at the end of game. (WIN=1, LOSE=-1, DRAW=0)
- The input is kibo(record), the label is the return, and in this case we can use tanh at the end of network to squeeze output, and use MSE as loss.
- Use deterministic policy (argmax of value function).

Value-based Approach with Omok (Cont.d)

- However, it’s not work very well…
- If we use a lot of ‘algorithm’s kibos’ and plays with algorithm, it can learn that ‘oh, connection is important, maybe?’
- However, it does not converge well.
- Also, argmax is not cool for good exploration.
- (Since value function gives only the expectation
- of reward, we cannot make a probablistic distribution
- which denote the comparison between actions.)



```
=====================================
 [ Turn  26 ; 1P's turn (tone = O) ]
  | 0 1 2 3 4 5 6 7 8 9 A
--+----------------------
0 | . . . . . . . . . . .
1 | . . . . . . . . . . .
2 | . . . O . . . . . . .
3 | . . . O X . . X . . .
4 | . . . X O X . . . . .
5 | . . O . O O X O . . .
6 | . . . . O O X O . . .
7 | . . . . . X O O . . .
8 | . . X . X X O X . . .
9 | . . . X . . . . . . .
A | . . X . . . . . . . .
2p (agent-analysis-based) win!
```

Value-based Approach

- I tested value-based approach to Connect 4, which is easier problem.
- But I couldn’t get good result (= win algorithms), using DQN with 5 convolution layers, 2 linear layers, and training with 20k games.

Policy-based Approach

- Policy-based approach, more specifically Monte-Carlo Policy Gradient (MCPG), or REINFORCE algorithm
- (RJ Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, 1992)
- is VERY SUCCESSFUL for connecting games.

Policy-based Approach (Cont.d)

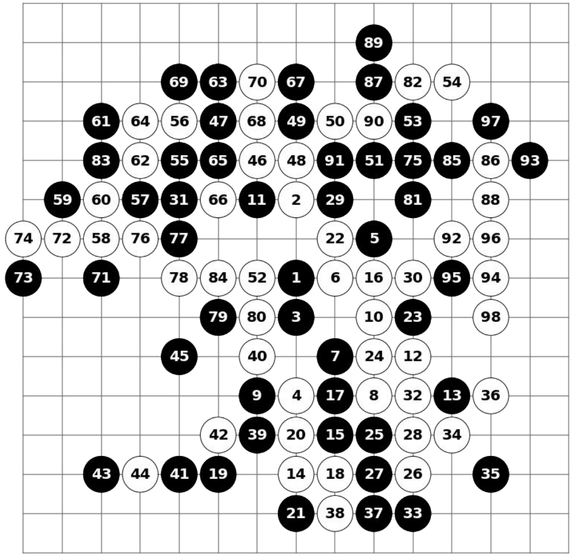
- MCPG calculates loss from the policy functions and change parameters of policy function directly.
- During training, it reinforces the probability of each actions, weighted by return. Thus, action of more reward will have more probability.
- MCPG uses loss looks like NLL for gradient ascent.
- In value learning, policy function only give 0 or 1 because of argmax, but policy learning give any value in [0,1]. Just the probability of better movements have more chances. i.e. MCPG train a stochastic policy.

Policy-based Approach (Cont.d)

- For connect 4, a policy network, almost same structure of the previous DQN, training with 40k games (since REINFORCE algorithm does not allow replay, both of time to train DQN with 20k games and one to train policy with 40k games are 30 minutes.), could achieve a win rate 0.5% against to algorithm.
- After some optimization and tuning, the policy network trained by REINFORCE algorithm during 10k games (10-15 minutes), achieved a win rate 88% against to the algorithm.

Policy-based Approach for Omok 1

- First, I apply almost same approached which I did for connect 4 to Omok.
- Just make a policy network larger, and use fully CNN.
- After it plays 15k games (about 53 minutes) played with an algorithm, it achieved 19.4% of win rate.
- In first 5k games, it can’t win the algorithm,
- but after that, it starts to win the algorithm
- occasionally.



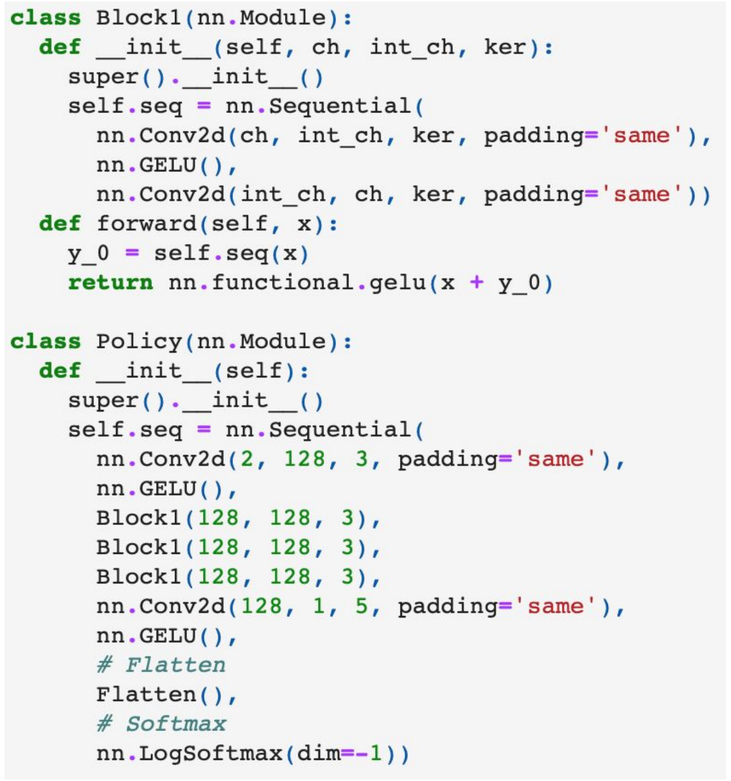
White stone(trained network) wins agains to black stone(algorithm)

Policy-based Approach for Omok 2

- To accelerate learning, I used algorithm’s kibo for first 1.5k games.
- In this case, policy could win the algorithm at the first of its play.
- However, after 13.5k games (i.e. 15k kibos), the max win rate was 19%, which is almost same as training without kibo.
- Thus, more time and more complexity to overcome the algorithm.

Policy-based Approach for Omok 3

- My final policy network structure is on the right.
- It's fully CNN, but add some skip connections as ResNet.
- It use 8 convolutional layers with 128 features.

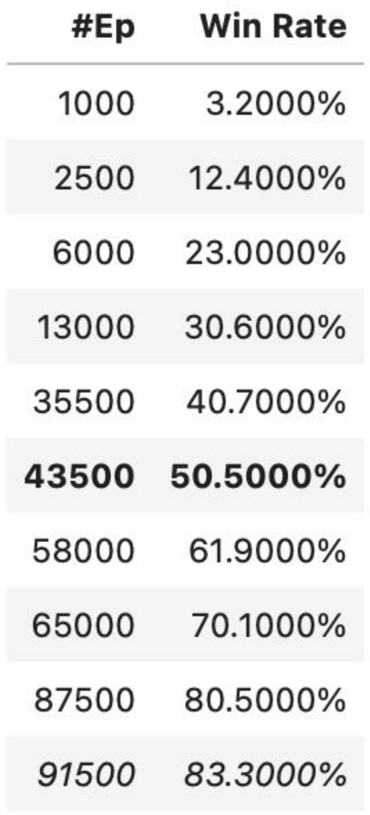


```python
class Block1(nn.Module):
  def __init__(self, ch, int_ch, ker):
    super().__init__()
    self.seq = nn.Sequential(
      nn.Conv2d(ch, int_ch, ker, padding='same'),
      nn.GELU(),
      nn.Conv2d(int_ch, ch, ker, padding='same'))
  def forward(self, x):
    y_0 = self.seq(x)
    return nn.functional.gelu(x + y_0)

class Policy(nn.Module):
  def __init__(self):
    super().__init__()
    self.seq = nn.Sequential(
      nn.Conv2d(2, 128, 3, padding='same'),
      nn.GELU(),
      Block1(128, 128, 3),
      Block1(128, 128, 3),
      Block1(128, 128, 3),
      nn.Conv2d(128, 1, 5, padding='same'),
      nn.GELU(),
      # Flatten
      Flatten(),
      # Softmax
      nn.LogSoftmax(dim=-1))
```

Policy-based Approach for Omok 3 (Cont.d)

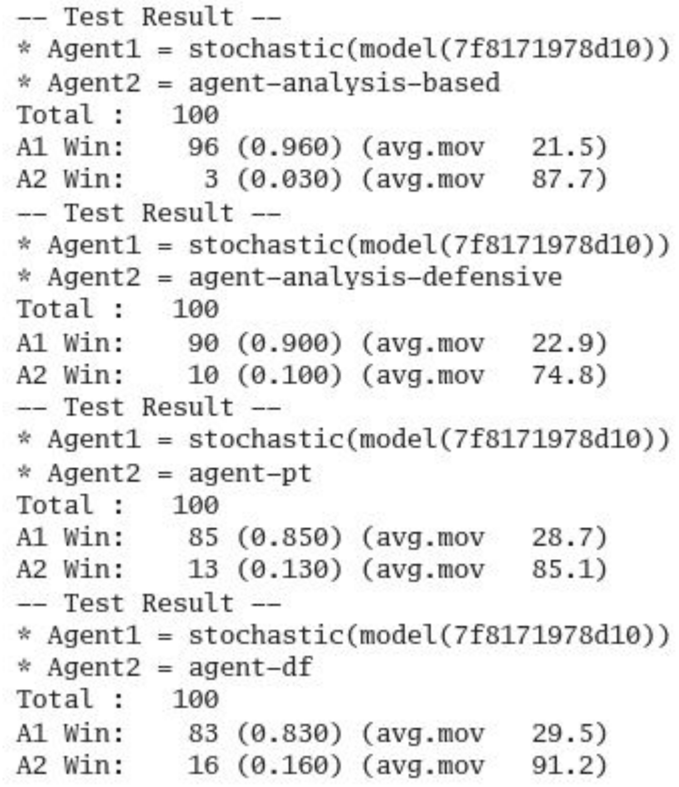
- I used REINFORCE algorithm too.
- The training is done with 100k games in 10 hours.
- After 43.5k games, the policy network are nearly equal to the algorithm.
- After 87.5k games, the policy network achieve 80% win rate against to the algorithm, but no dramatic improvements after that point.



#Ep	Win Rate
1000	3.2000%
2500	12.4000%
6000	23.0000%
13000	30.6000%
35500	40.7000%
43500	**50.5000%**
58000	61.9000%
65000	70.1000%
87500	80.5000%
91500	*83.3000%*

Policy-based Approach for Omok 3 (Cont.d)

- After 100k games, the policy network can win 9 of 10 games against to algorithms.
- When the policy wins, it finish the game about 25 movements.
- And it endure about 80 movements even if it loses.



```
-- Test Result --
* Agent1 = stochastic(model(7f8171978d10))
* Agent2 = agent-analysis-based
Total :   100
A1 Win:    96 (0.960) (avg.mov   21.5)
A2 Win:     3 (0.030) (avg.mov   87.7)
-- Test Result --
* Agent1 = stochastic(model(7f8171978d10))
* Agent2 = agent-analysis-defensive
Total :   100
A1 Win:    90 (0.900) (avg.mov   22.9)
A2 Win:    10 (0.100) (avg.mov   74.8)
-- Test Result --
* Agent1 = stochastic(model(7f8171978d10))
* Agent2 = agent-pt
Total :   100
A1 Win:    85 (0.850) (avg.mov   28.7)
A2 Win:    13 (0.130) (avg.mov   85.1)
-- Test Result --
* Agent1 = stochastic(model(7f8171978d10))
* Agent2 = agent-df
Total :   100
A1 Win:    83 (0.830) (avg.mov   29.5)
A2 Win:    16 (0.160) (avg.mov   91.2)
```

Other Policy-based Approaches for Omok

- There are another algorithms to train policy network!
- Actor-Critic and A2C train both of value and policy networks.
- Instead of using the the return at the end of game, it uses the estimation by value network for training.
- However, since Omok episodes are very short and always finite, value network is not necessary.
- Because of no significant improvement, I threw it away.

Other Policy-based Approaches for Omok

- I tried to implement AlphaZero-like MCTS.
- But for each play, it must run the neural network a lot of times. (AlphaGo may run each networks 1600 times FOR EACH MOVEMENT.) Compared to the previous approach, which run a policy network once for each movement, MCTS takes x100~x10000 more times to generate episode.
- Using Google Colab in 10 hours, the previous approach can play and learn 100k games, but MCTS only played 1.7k games.
- MCTS is better by help of better decision algorithm, but the number of episodes is more important at the first step of learning.

Conclusion about Training Omok

Tips for Training Omok by Experiences

- “The number of episodes is most important“.
- Even if you equipped powerful approximation algorithm, you may not reduce the number of plays to obtain a result.
- In other words, you should consider distributed/parallel computing and off-policy learning to generate lots of episodes, before think about how to improve algorithms.

Tips for Training Omok by Experiences (Cont.d)

- “The quality of episodes is important“.
- Because RL, policy gradient more specifically, just increase the probability of ‘good actions’ obtained in episode, the trained result become better when each episode contains better actions.
- If the opponent is powerful, agents rapidly try to mimic the powerful actions during training.
- If the opponent is very weak, such as random movements, agents may not learn anything from episode.
- This is why self-play learning is hard when the policies are poor.
- MCTS is useful in this sense, because it finds out movements better than ones memorized in neural nets by Tree Search and memorizes the better movements after the episode.

Tips for Training Omok by Experiences (Cont.d)

- If you don’t know which RL algorithm is the best fit, apply policy gradient first.
- Stochastic policy with temperature softmax may be better and easier than deterministic policy with epsilon exploration. You may not worry about exploration with stochastic policy.
- In policy gradient objective function, if we choose very rare action and the reward absolutely large, the gradient will be explode. You may need gradient clipping to prevent this problem.

Tips for Training Omok by Experiences (Cont.d)

- If the action decided by local state, try to use Fully CNN. It may give faster and better approximation.
- You may stack over 10 layers in neural networks for Omok. But you many not need more than 15 layers.
- However, the feature dimension is very important. Less than 64 feature dimension lost too many information, and the result was not be good.
- USE GELU. RL requires parametric functions to wander off in the loss space, and too many ReLU may be dead during training.
- Dropout is not helpful because underfit is more frequent than overfit in RL.
- Batch normalization may not be very helpful because the state-action space is too large.

Tips for Training Omok by Experiences (Cont.d)

- Distinction between on- and off-policy is important for math, but may not need for engineering. Without importance sampling, it can be trained anyway.
- For RL, it may be better to jumble parameters rather than make a stable convergence of parametric functions. Therefore, rather Adam, use SGD with large learning rate and large momentum.
- Also, large batch may indicate stable direction and it may be harmful to find global optimum. Thus, use smaller batch.

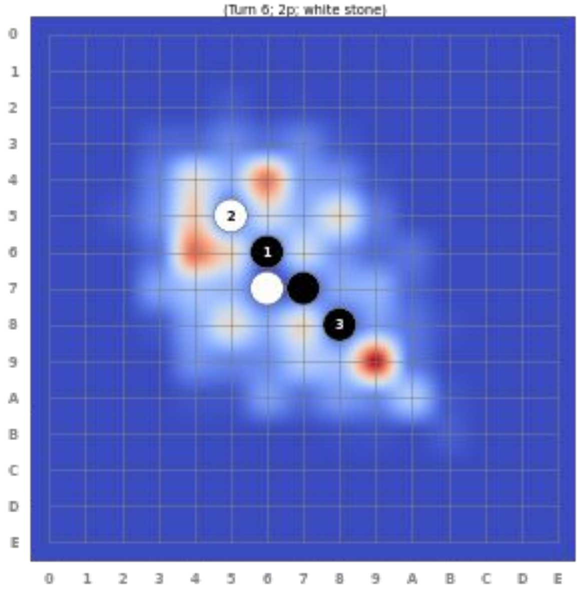
How Does the Policy Play Omok?

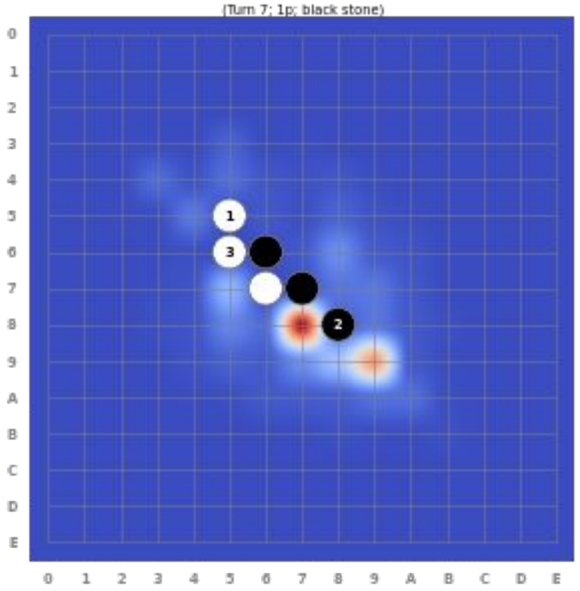
Visualize Policy Network

- In my experiments, the policy network is just a function p:S->[0,1]^A, which takes a board state and return a board with probabilities for each cells.
- For the fixed state, the policy network just give us an image!
- I wanted to see how it looks like.

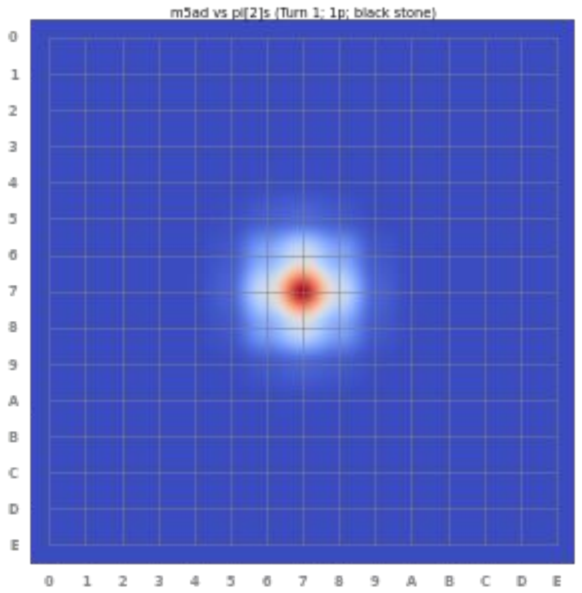
Visualize Policy Network (Cont.d)

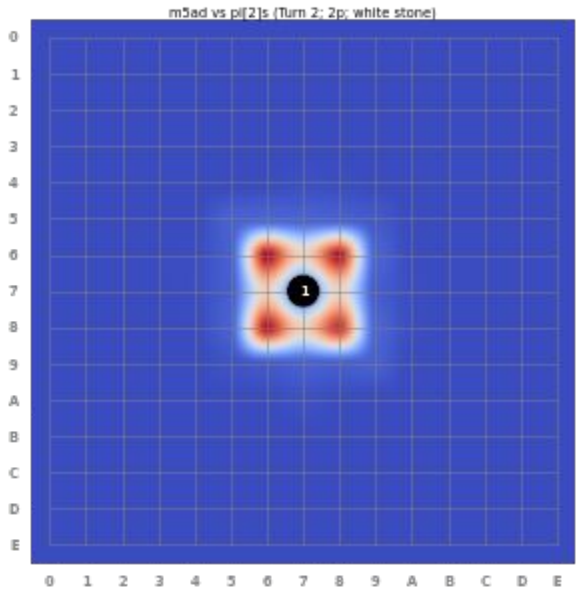
- The color is the normalized probabilities power of 0.3.
- Blue means 0, red means 1, white means 0.5^(10/3)~= 0.1

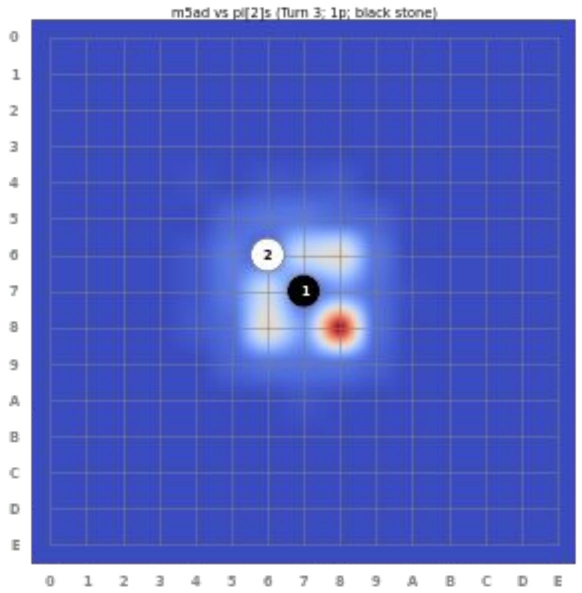




Visualize Policy in a Game v.s. Algorithm







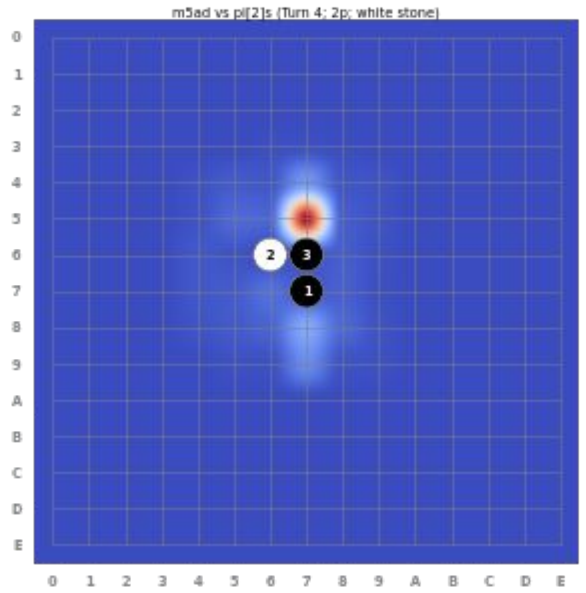
Center is preferred!

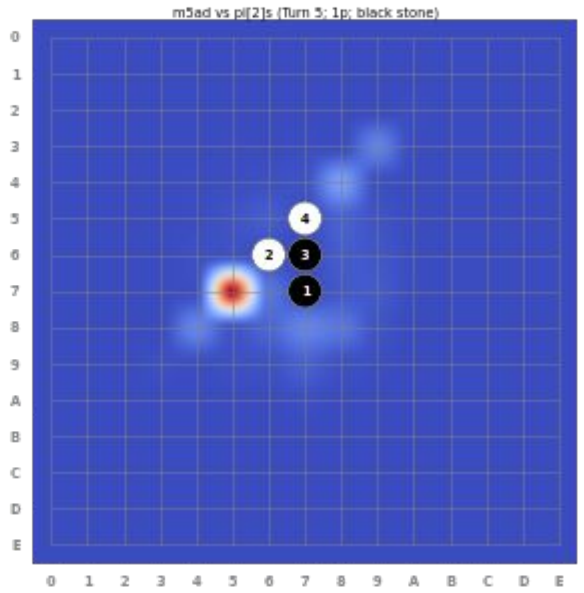
Indirect walk is preferred!

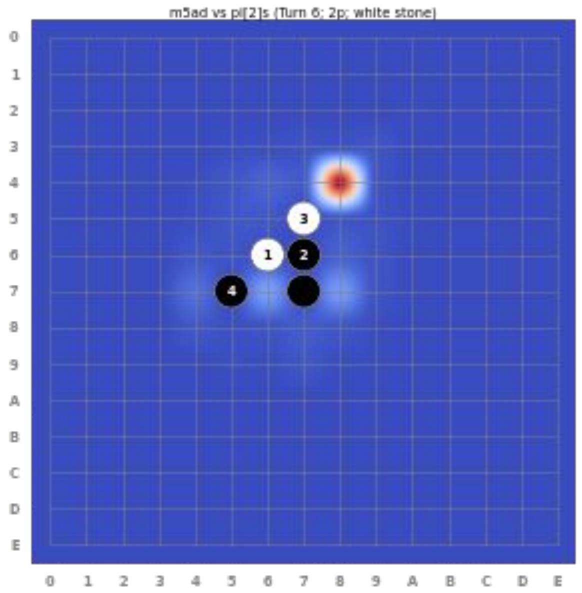
간접 사월 주형

Black stone = algorithm, White stone = trained policy network

Visualize Policy in a Game v.s. Algorithm

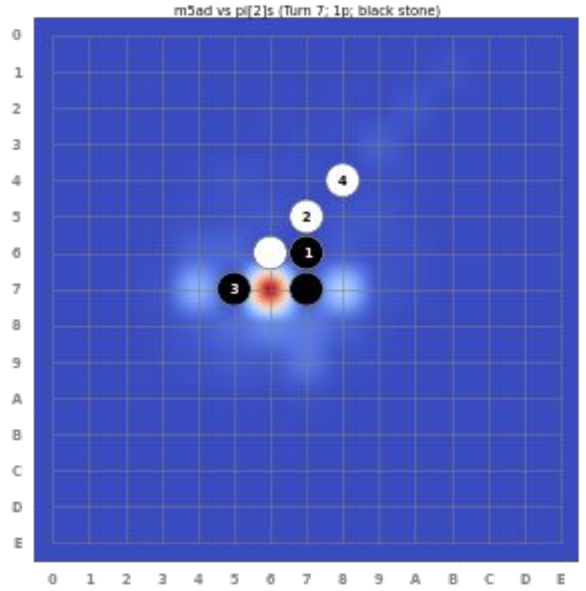


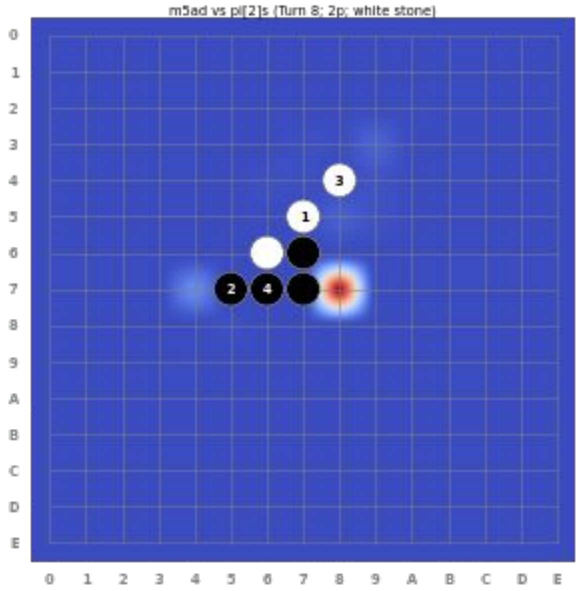


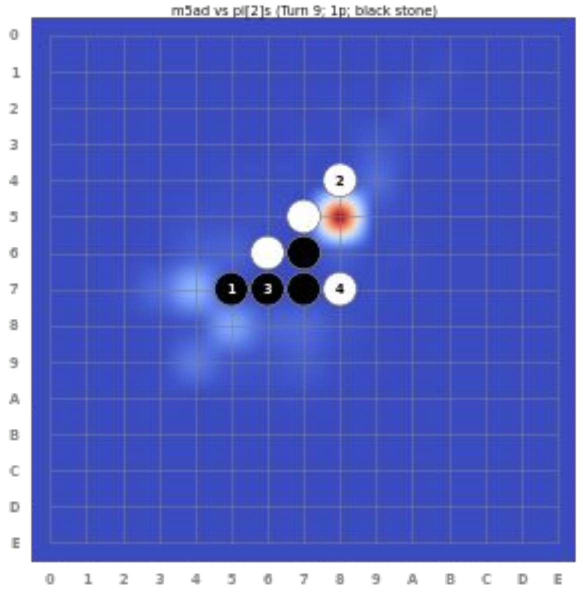


You can see the movement can make a connection is preferred

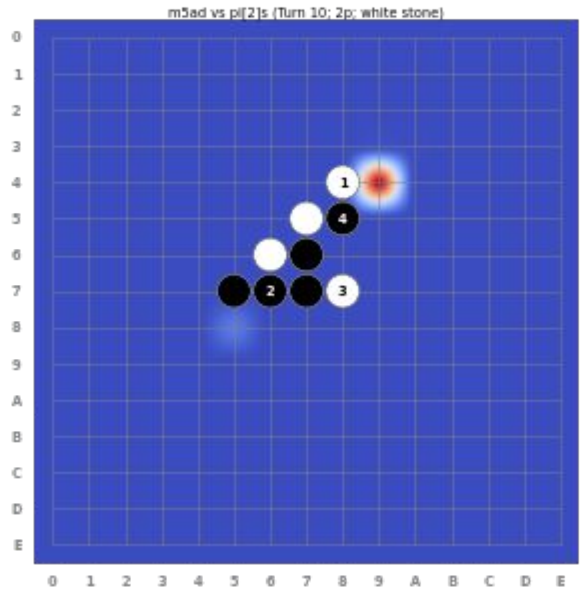
Visualize Policy in a Game v.s. Algorithm

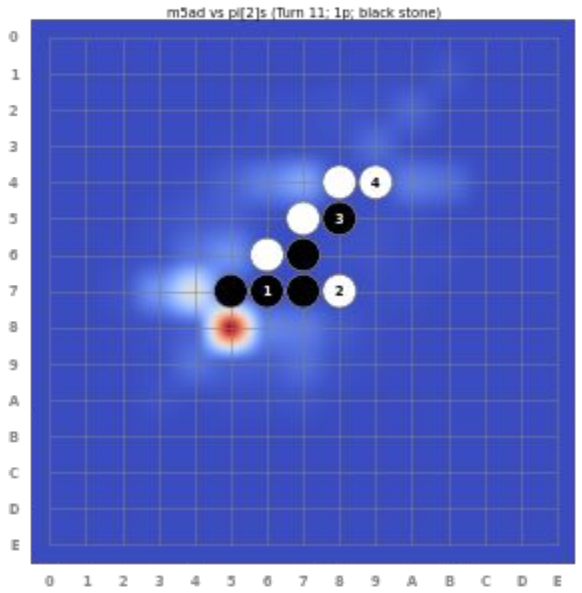


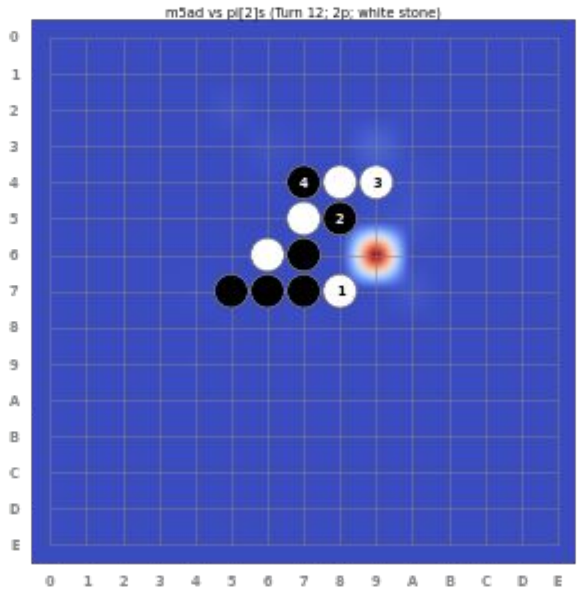




Visualize Policy in a Game v.s. Algorithm

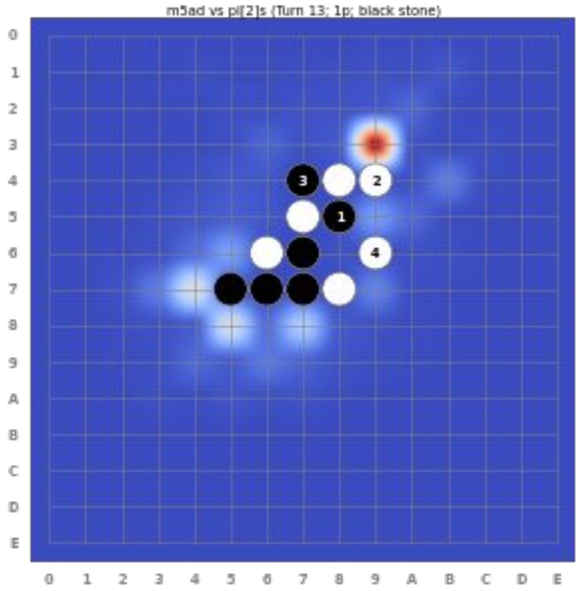


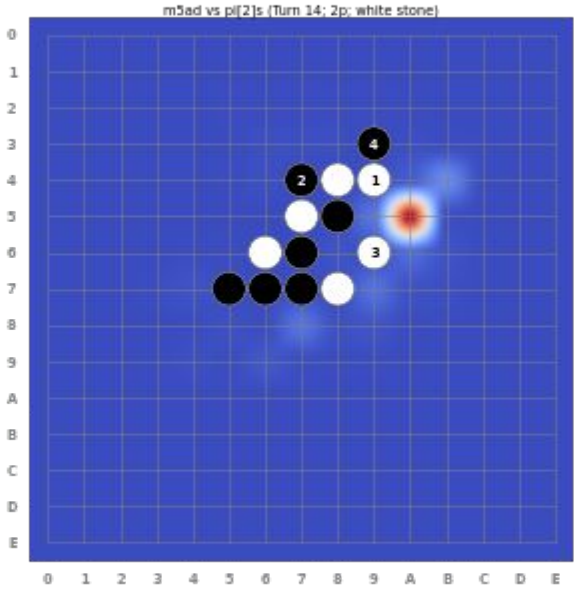


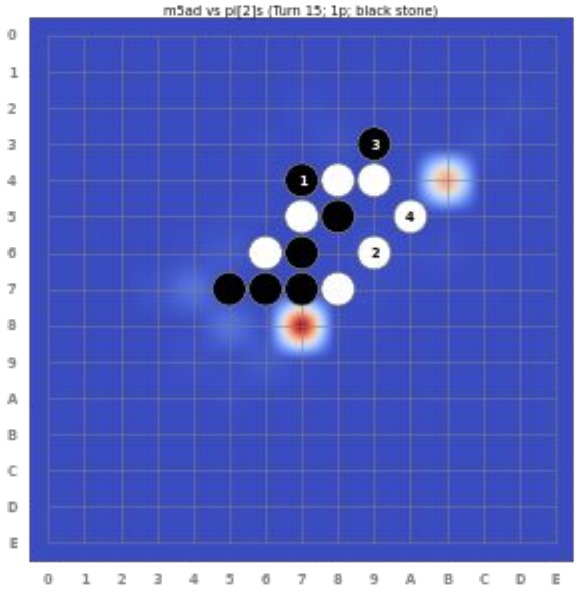


Block open-3

Visualize Policy in a Game v.s. Algorithm

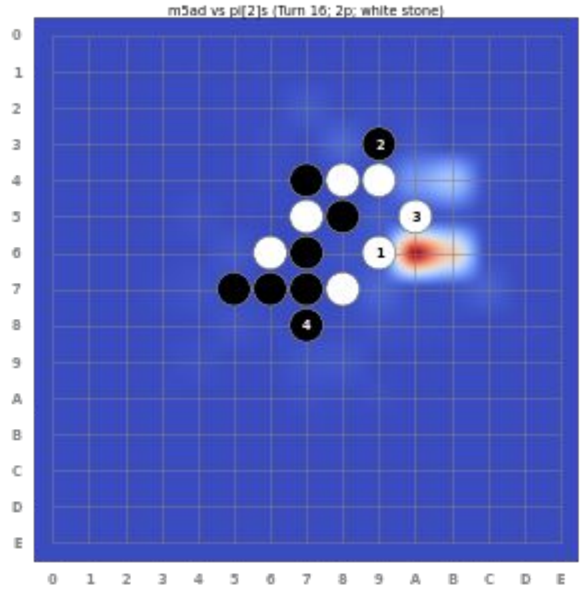


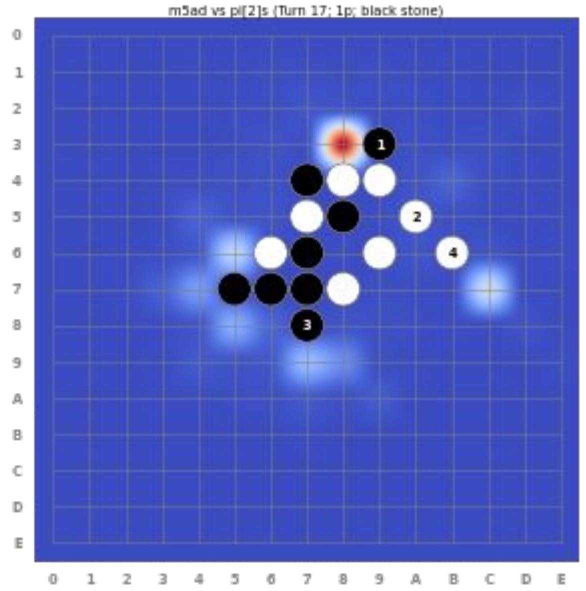


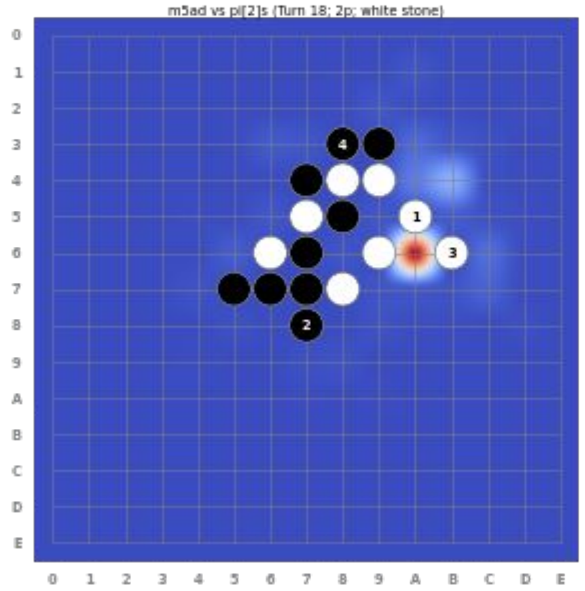


Open-3 must beeee blocked.
Policy think below one is better
because tha action can block and extend connection.

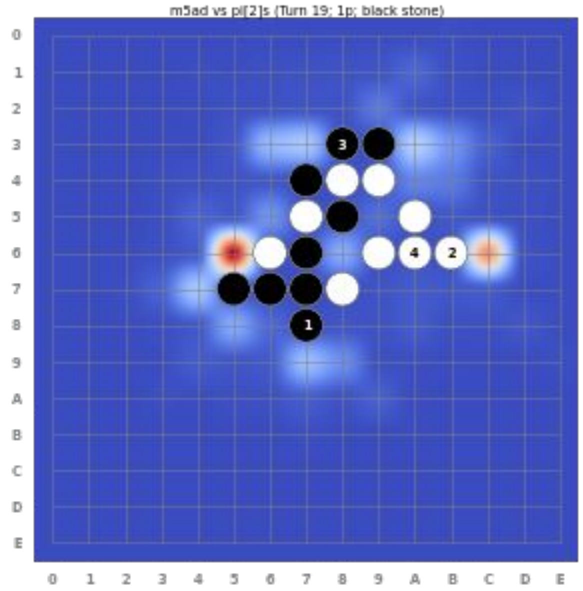
Visualize Policy in a Game v.s. Algorithm

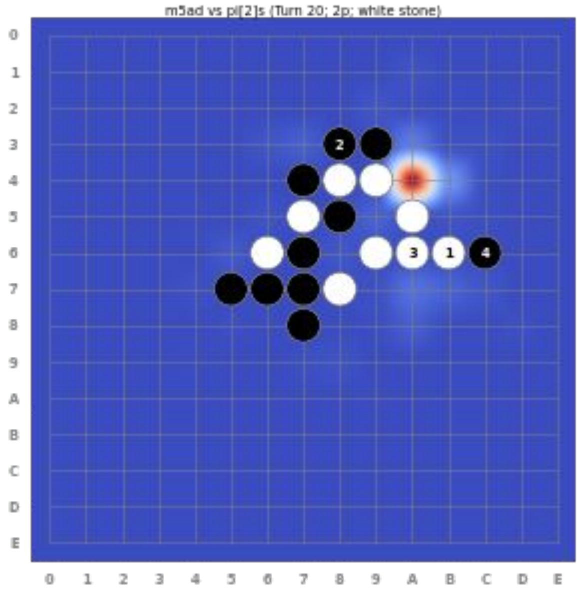


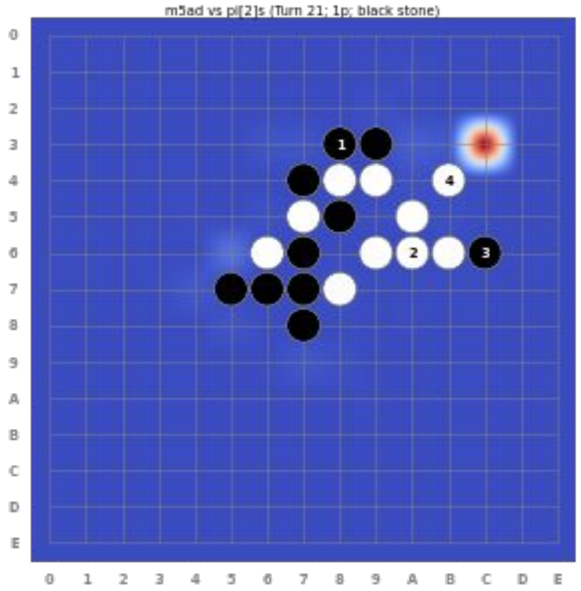




Visualize Policy in a Game v.s. Algorithm

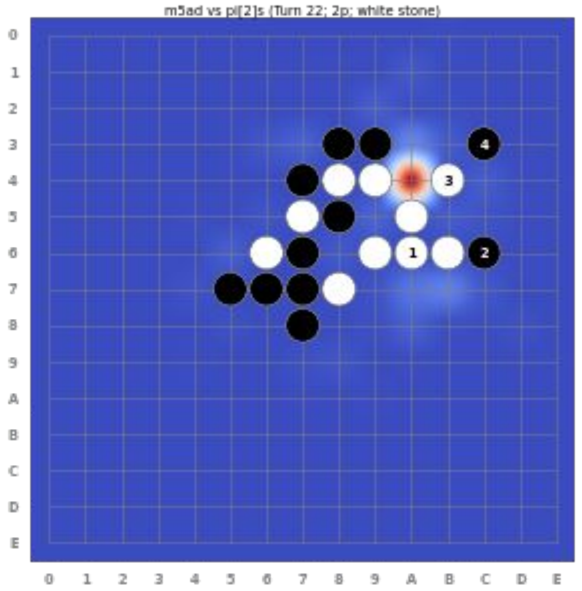


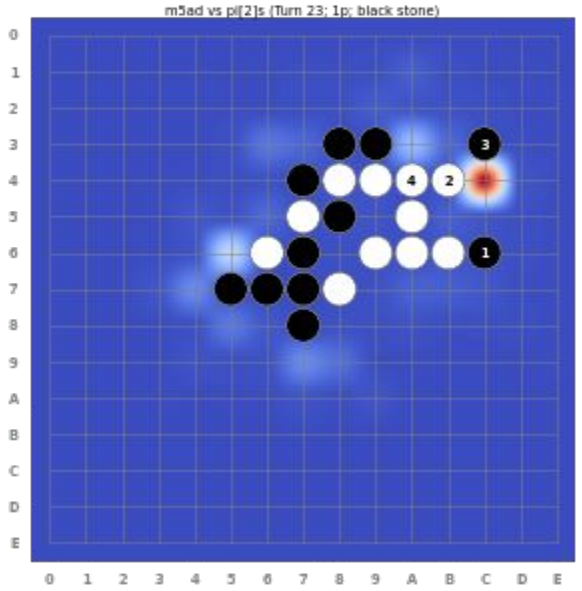


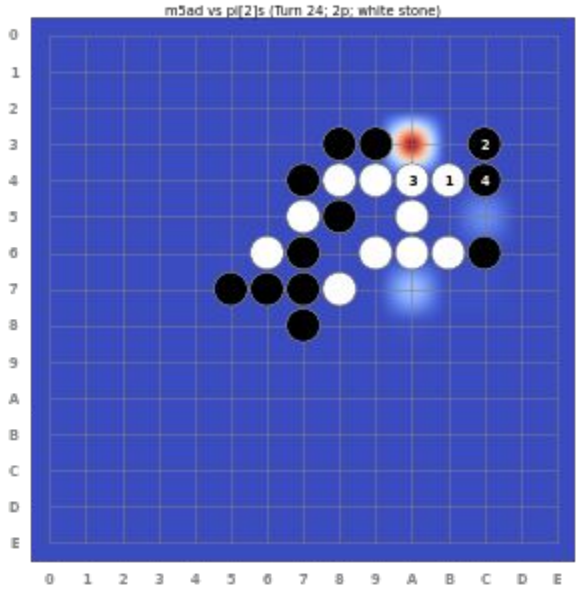


This is wrong prediction,
If it did not block the right white stones,
game will be end

Visualize Policy in a Game v.s. Algorithm

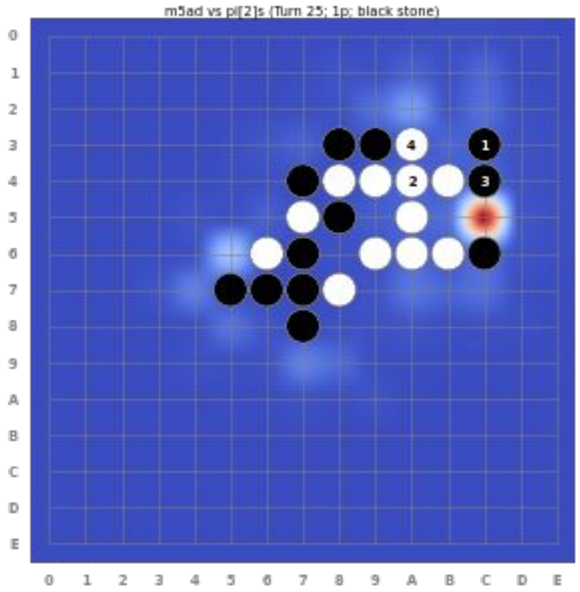


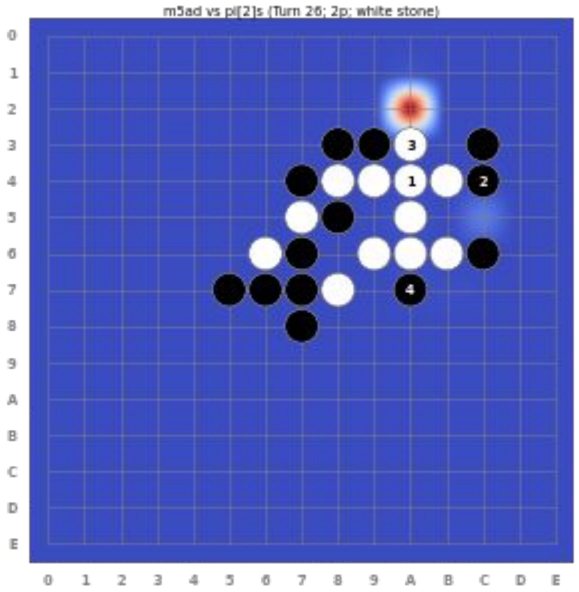




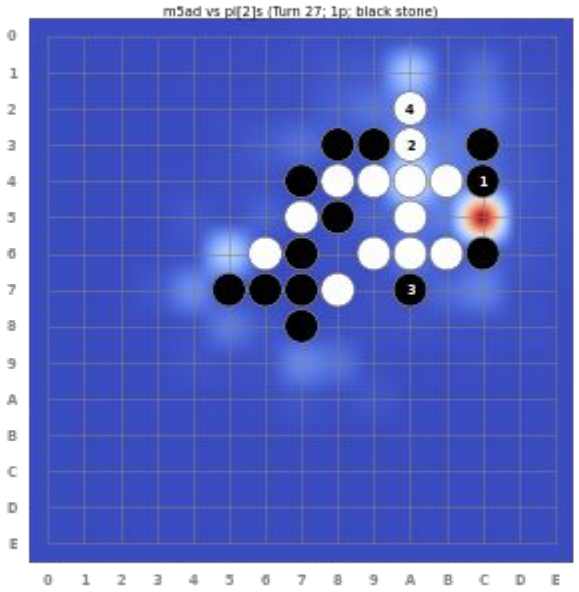
4-3 attack

Visualize Policy in a Game v.s. Algorithm





Finish the game



White stone (policy network) win!

Visualize Policy Network (Cont.d)

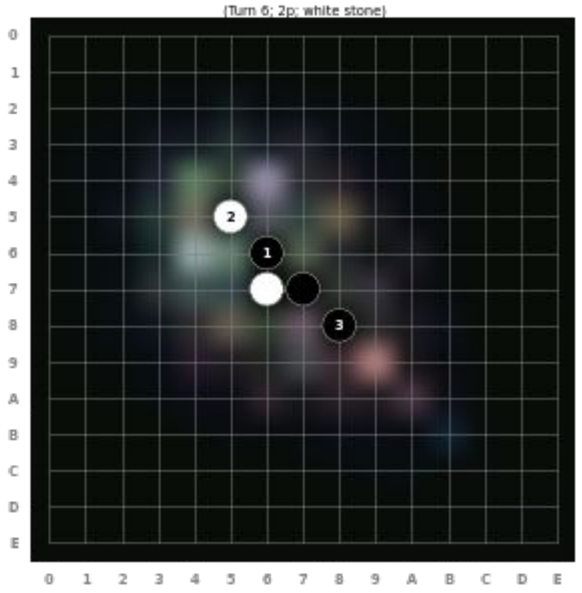
- It's interesting that it prefers the center of board at the first state! (because the policy is full CNN and the first state of board is just zero array.)
- The probability of actions not directly connecting stones is almost zero. It means the agent looks like understanding what is omok, but it will not try knight's move strategy (날일진)
- It may make mistake when both of the agent and the opponent can attack. Even if the opponent's attack is faster, it may attack rather than defense.

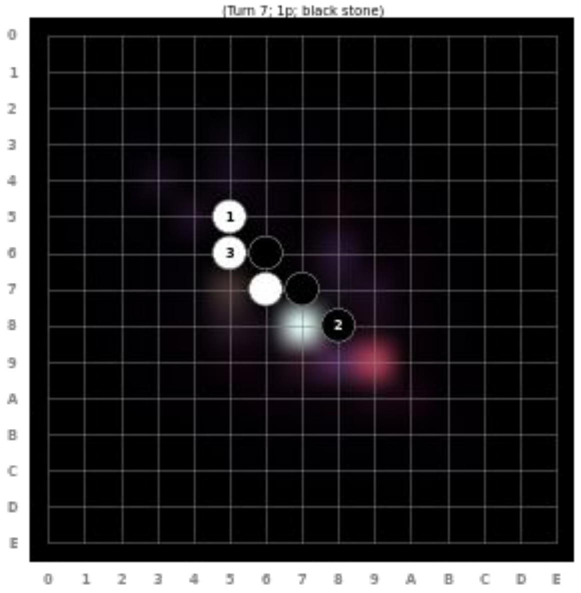
Visualize Multiple Policies

- After train an agent with 100k games, I duplicated the policies and incite a fight between them and algorithms to achieve 100% win rate against my algorithms.
- In this training, I keep 5 best policies, and if the new one training with the policies and algorithms is better, replace one of the policy to new one.
- In evaluation step, I saw that the policies are not very improved. But also their win rate against to each other was not 0.5. It means they have slightly different strategies.
- Therefore, I thought how about to visualize multiple policies at once.

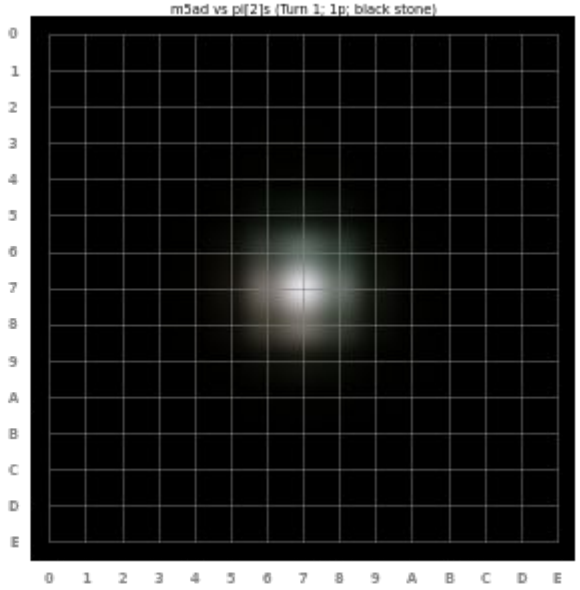
Visualize Multiple Policies (Cont.d)

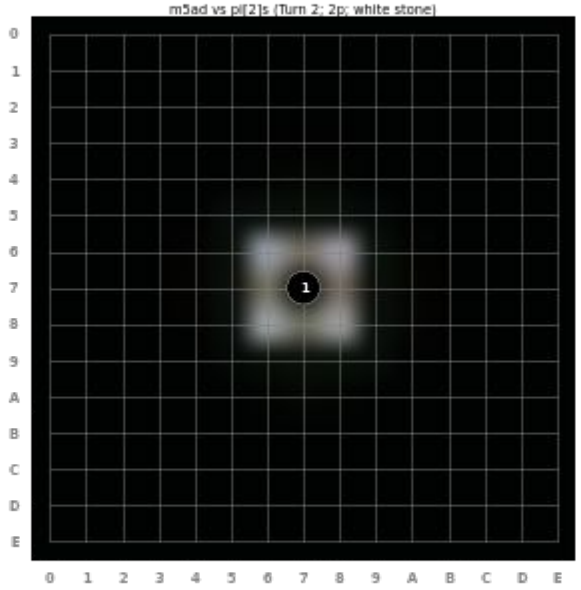
- I chose and fix 3 of the 5 policies arbitrarily.
- Each RGB channel represents the probability of the 1st, 2nd and 3rd policy.
- e.g. red=only 1st prefers, cyan=2nd and 3rd prefer, white=everyone prefer

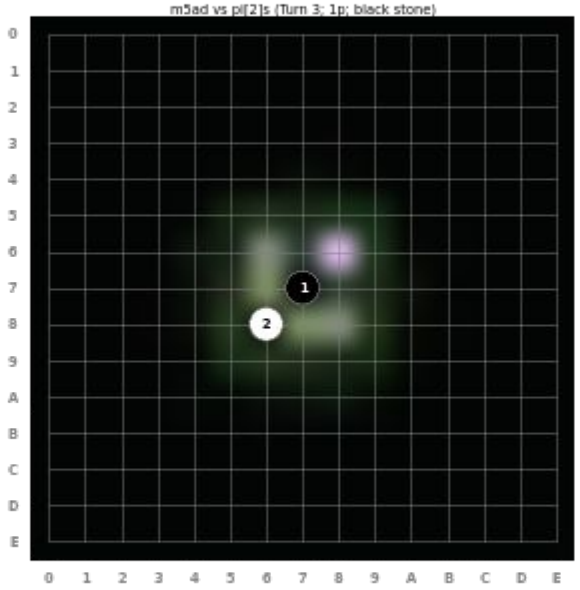




Visualize Policies in a Game v.s. Algorithm



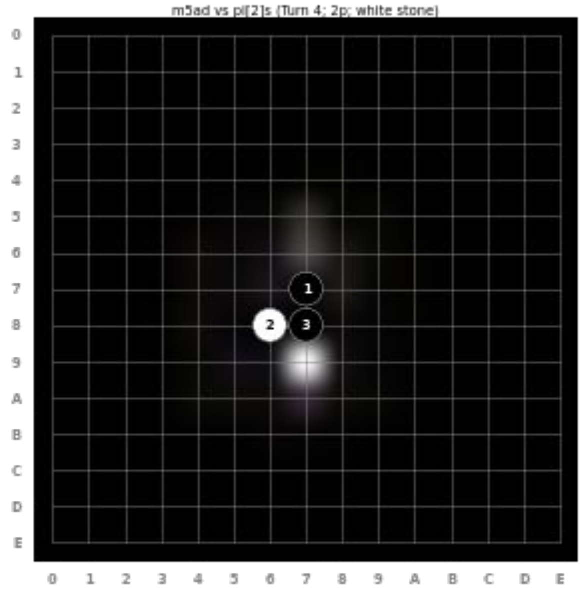


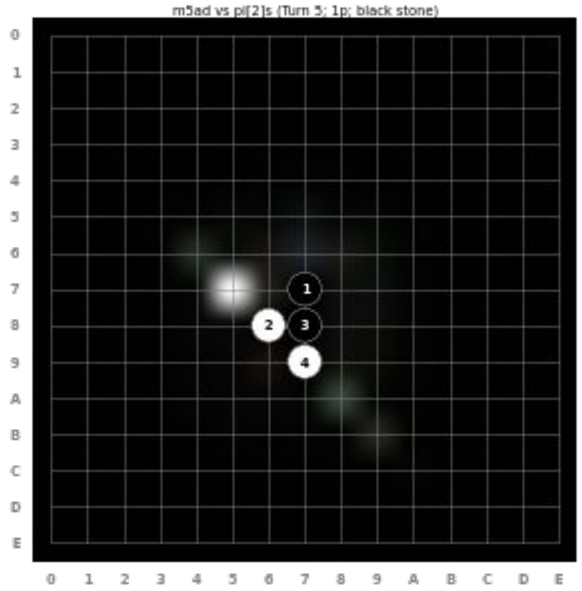


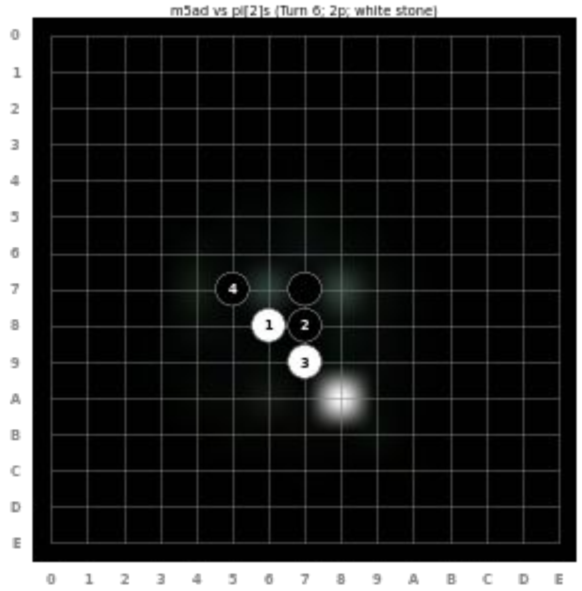
From the start, the prob.distr. is slightly different.

Black stone = algorithm, White stone = trained policy network(blue)

Visualize Policies in a Game v.s. Algorithm

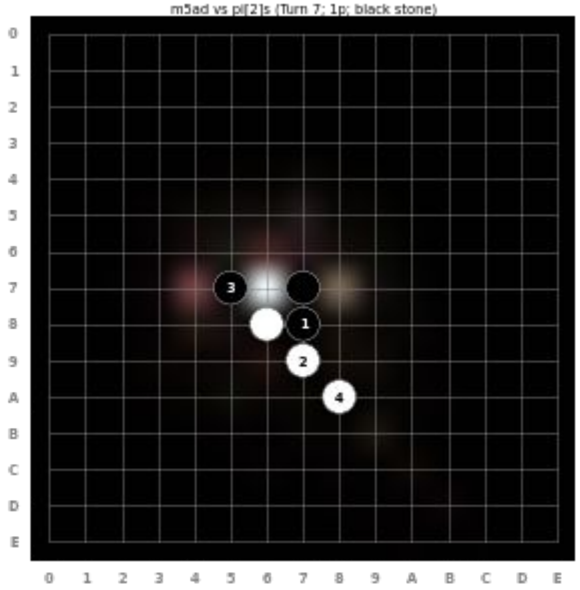


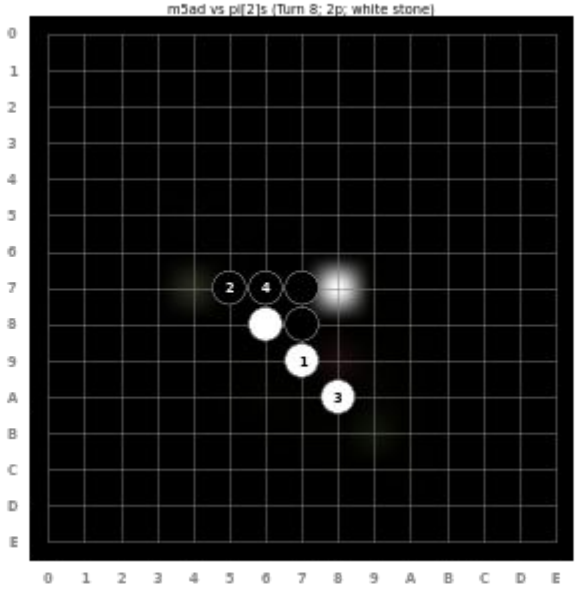


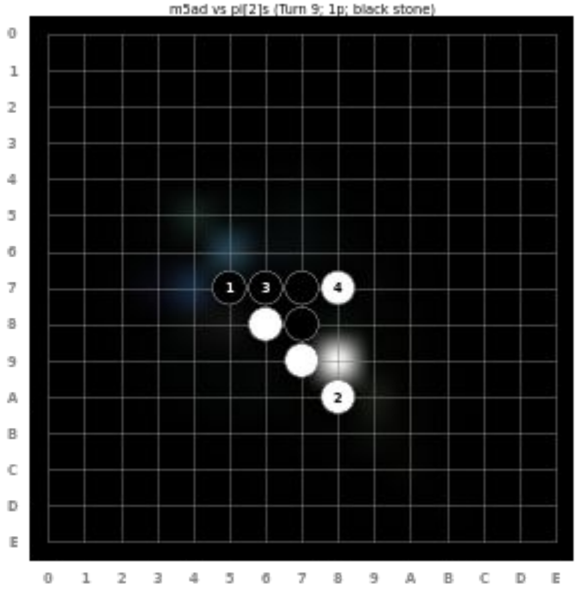


Connecting/blocking actions is prefered for everyone,
but the 2nd (green)'s action is slightly distrubuted.

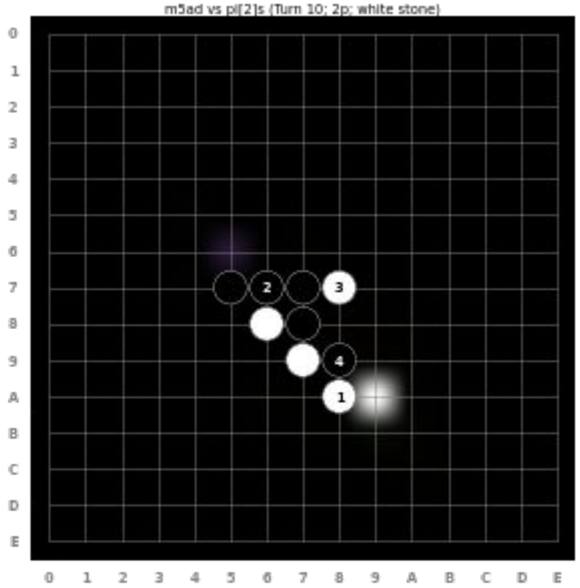
Visualize Policies in a Game v.s. Algorithm

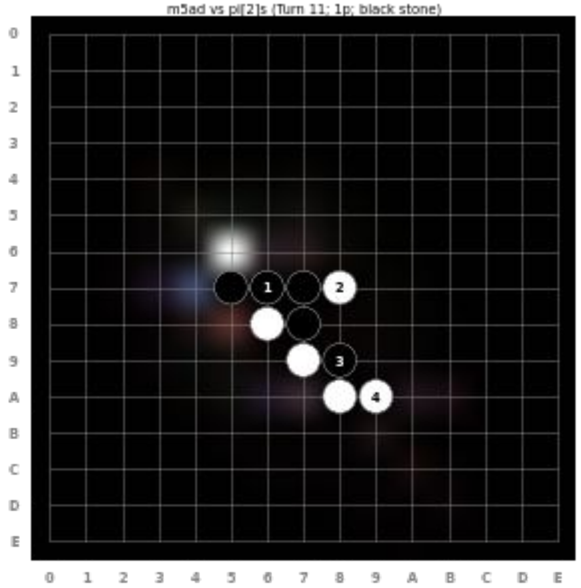


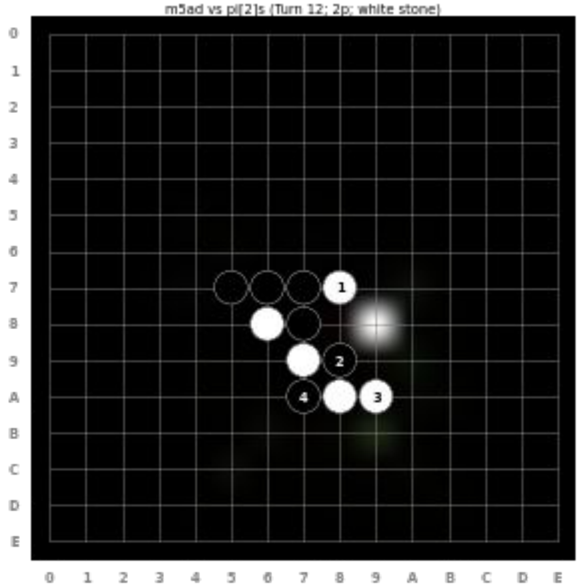




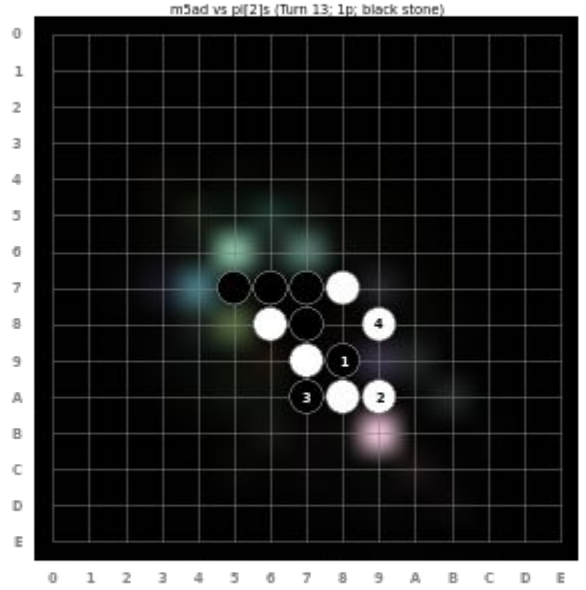
Visualize Policies in a Game v.s. Algorithm

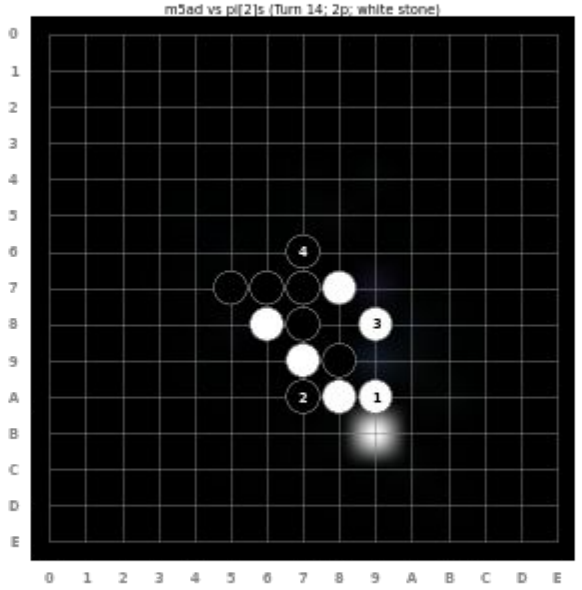


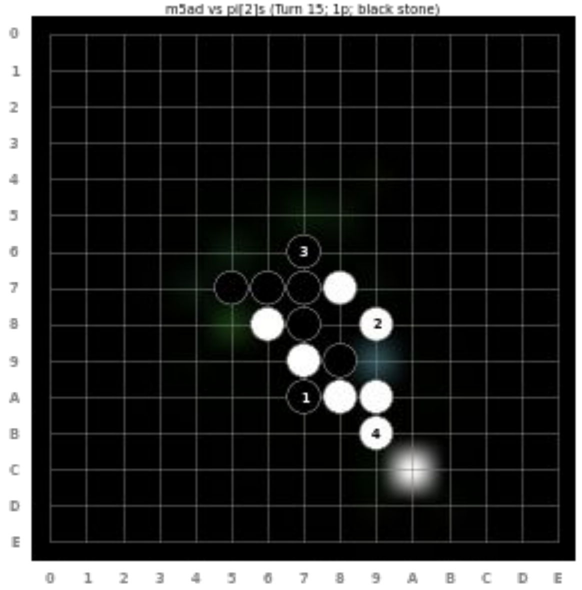




Visualize Policies in a Game v.s. Algorithm

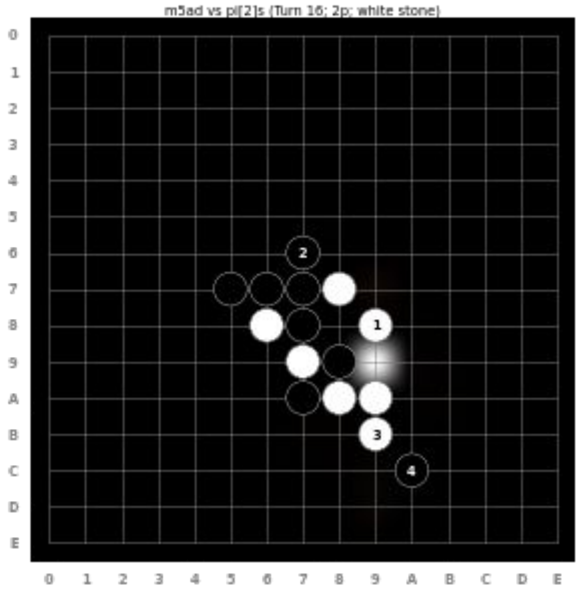


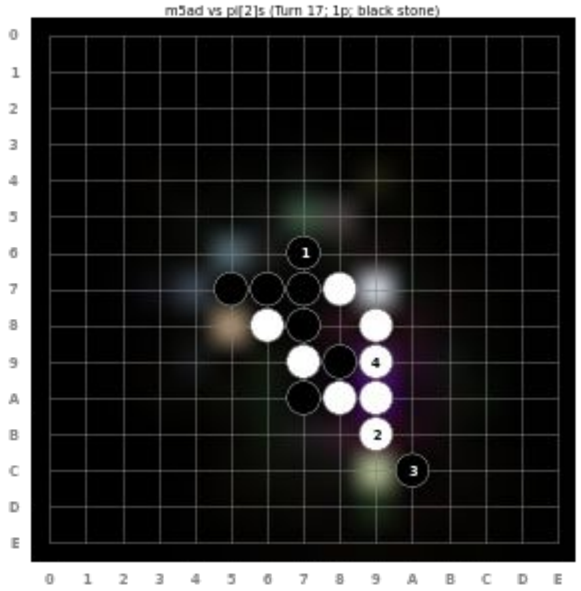


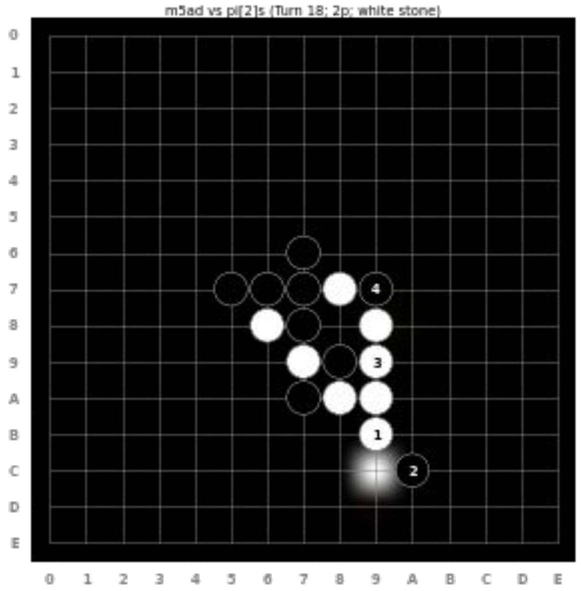


The difference is noticeable in this situation.
Everyone agrees the right bottom must be blocked which can be 3-4.
However, the 2nd (green) and 3rd (blue) think the left top position (extend black connection) is good.

Visualize Policies in a Game v.s. Algorithm



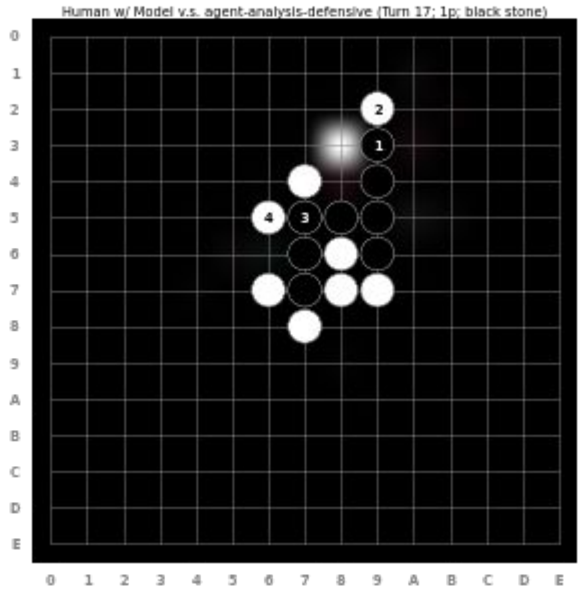




When the game can be finished,
some of them consider other strange movements.

Visualize Multiple Policies (Cont.d)

- We can see the differences between policies, but they are not different extremely.
- In most situations, there is a movement everyone agrees. It means even if we aggregate their choices (as ensemble method does), no great change may exist.
- But since they shows various movements worth to consider, it can be used for hints to human!


Human w/ Model v.s. agent-analysis-defensive (Turn 17; 1p; black stone)
0 1 2 3 4 5 6 7 8 9 A B C D E
1
2
3
4

Repo.

- https://github.com/lumiknit/journey-to-learn-omok