Using AlphaZero to Play Chinese Chess

Jiahua You and Yifeng Liu

Link to proposal
Link to presentation

Abstract

Chinese Chess (Xiangqi), is a strategy board game for two players.The game represents a battle between two armies, with the object of capturing the enemy's general (king). Distinctive features of xiangqi include the cannon (*pao*), which must jump to capture; a rule prohibiting the generals from facing each other directly; areas on the board called the *river* and *palace*, which restrict the movement of some pieces (but enhance that of others); and placement of the pieces on the intersections of the board lines, rather than within the squares.[1]

Introduction and Rules

Xiangqi is played on a board nine lines wide and ten lines long. As in the game Go, the pieces are placed on the intersections, which are known as *points*. The vertical lines are known as *files*(*columns*), and the horizontal lines are known as *ranks* (*rows*).[1]

Centered at the first to third and eighth to tenth ranks of the board are two zones, each three points by three points, demarcated by two diagonal lines connecting opposite corners and intersecting at the center point. Each of these areas is known as, a castle.

Dividing the two opposing sides, between the fifth and sixth ranks, is, the "river". The river is often marked with the "River of the Chu ", and the "Border of the Han", a reference to the Chu-Han War. Although the river provides a visual division between the two sides, only two pieces are affected by its presence: soldiers have an enhanced move after crossing the river, and elephants cannot cross it. The starting points of the soldiers and cannons are usually, but not always, marked with small crosses.[1]

The pieces start in the position shown in the diagram above. Which player moves first has varied throughout history and from one part of China to another. Different xiangqi books advise either that the black or red side moves first.Some books refer to the two sides as north and south; which direction corresponds to which color also varies from source to source. Generally, Red moves first in most modern tournaments. [1]

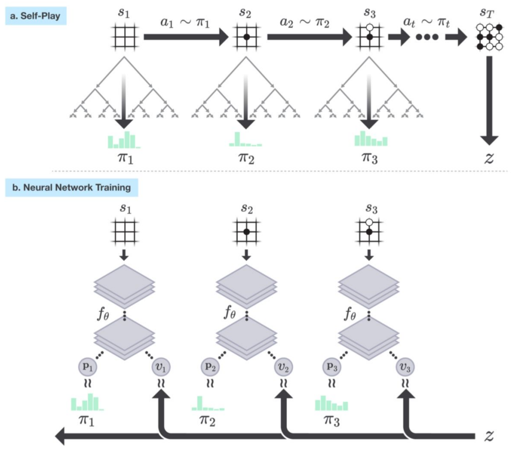
Each player in turn moves one piece from the point it occupies to another point. Pieces are generally not permitted to move through a point occupied by another piece. A piece can be moved onto a point occupied by an enemy piece, in which case the enemy piece is captured and removed from the board. A player cannot capture one of his own pieces. Pieces are never promoted (converted into other pieces), although the soldier is able to move sideways after it crosses the river.[1]

Related Work

In this project we study another technique and papers from AlphaGo, with is Mastering the Game of Go without Human Knowledge, and Mastering Chess and Shogi by Self-play with a General Reinforcement Learning Algorithm, to implement AlphaXiangqi.

The first and most important is that, it news no human data whatsoever so it learns solely by self play by playing a games against itself, and training by reinforcement learning starting from completely random games. So in other words, we initialize on the own network to random weights. And it plays games against itself starting with those random weights. So it knows nothing about the game and everything else is discovered from there onwards.

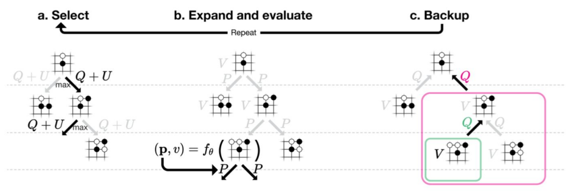
The difference is that AlphaGo Zero uses absolutely no handcrafted features there's nothing in there beyond the raw board. So, all it sees when it takes us out with the only thing that the neural network sees and received as an input is actually a representation of the raw board. Then, it justice has a plain saying you know whether there's chess in this position and plain saying whether there's another chess in this position. And that's it that's all it sees but there are some additional differences one of the most notable ones is that we actually unified the policy network and value network into a single neural network again based on a state-of-the-art residual network. And, this actually helped to regularize the network and make it much more robust against the kind of overfitting which is possible to see if you just have a value network trained by itself for example in addition AlphaGo zero actually uses a simpler search.



The above diagram is Self-play reinforcement learning in *AlphaGoZero*

a. The program plays a game s1,...,sT against itself. In each position st, a Monte-Carlo tree search (MCTS) αθ is executed using the latest neural network fθ. Moves are selected according to the search probabilities computed by the MCTS, at ~ πt. The terminal position sT is scored according to the rules of the game to compute the game winner z.

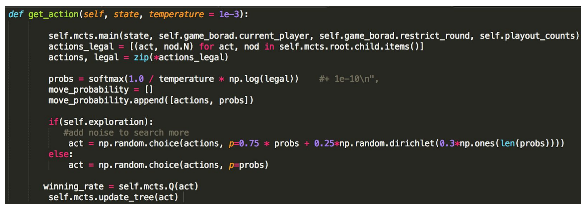
b. Neural network training in *AlphaGo Zero*. The neural network takes the raw board position st as its input, passes it through many convolutional layers with parameters θ, and outputs both a vector pt, representing a probability distribution over moves, and a scalar value vt, representing the probability of the current player winning in position st. The neural network parameters θ are updated so as to maximise the similarity of the policy vector pt to the search probabilities πt, and to minimise the error between the predicted winner vt and the game winner z. The new parameters are used in the next iteration of self-play.[2]



The MCTS algorithm proceeds in in three stages, in the first stage, the algorithm traverses the tree from root to leaf, picking each node it picks a child based on a kind of upper confidence rule to decide which action it should select.

So, this rule actually selects actions that have been evaluated most highly in pass simulations that this Q value from each node, and it also prefers to choose actions that the policy network likes that's the U term, the U term kind of is a bonus term that includes what the policy network thinks of the move.

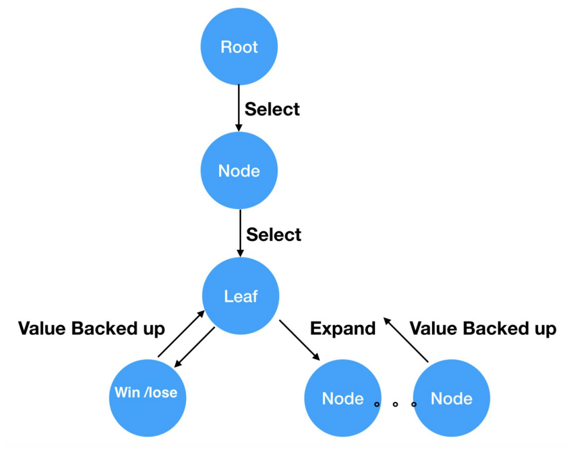
In the second stage, once we've actually traversed this this simulation down to a leaf node we expand the leaf node and evaluate this new node with both the policy and the value networks.
To simulate the playing. The mcts defines as:



```
def get_action(self, state, temperature = 1e-3):

        self.mcts.main(state, self.game_borad.current_player, self.game_borad.restrict_round, self.playout_counts)
        actions_legal = [(act, nod.N) for act, nod in self.mcts.root.child.items()]
        actions, legal = zip(*actions_legal)

        probs = softmax(1.0 / temperature * np.log(legal))    #+ 1e-10\n",
        move_probability = []
        move_probability.append([actions, probs])

        if(self.exploration):
           #add noise to search more
            act = np.random.choice(actions, p=0.75 * probs + 0.25*np.random.dirichlet(0.3*np.ones(len(probs))))
        else:
            act = np.random.choice(actions, p=probs)

       winning_rate = self.mcts.Q(act)
        self.mcts.update_tree(act)
```

And finally in the third stage, we back up the evaluation so whatever this evaluation was according to the value, Network we back that up through the search tree. So that every node we maintain the mean evaluation from each of those edges, so each Q value, there is basically storing the mean evaluation that it's seen from that point onwards in the search.

At this process this process Monte-Carlo research can very effectively expand a very large search tree by just considering that the most important parts of that search tree get expanded. And it systematically kind of gets deeper and deeper.



The diagram above can articulate the whole process.

Go Feature	Planes	Chess Feature	Planes	Shogi Feature	Planes
P1 stone	1	P1 piece	6	P1 piece	14
P2 stone	1	P2 piece	6	P2 piece	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P2 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 castling	2		
		P2 castling	2		
		No-progress count	1		
Total	17	Total	119	Total	362

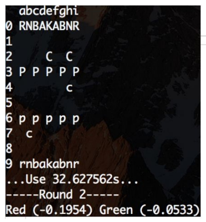
The diagram above shows the input features used by *AlphaZero* in Go, Chess and Shogi respectively.

The first set of features are repeated for each position in a T = 8-step history. Counts are represented by a single real-valued input; other input features are represented by a one-hot encoding using the specified number of binary input planes. The current player is denoted by P1 and the opponent by P2.

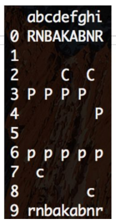
To make neural network works, we defined the input features. It is similar the chess, but with 7 kinds of pieces. So there are 14 input features. As for some other features mentioned in the paper like color, which round the player is in and records are not taken into account. So the input is just the current board with piece location information.

Results

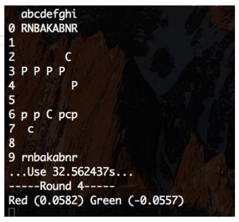
The program is modified from original alphazero program. The program plays start with no heuristic knowledge. And single GPU runs with very low efficiency. The initial data is very random with no right solution. It starts to show intelligence since the 500 times. It has some behavior of defending and attack.



```
  abcdefghi
0 RNBAKABNR
1
2     C  C
3 P P P P P
4        c
5
6 p p p p p
7  c
8
9 rnbakabnr
...Use 32.627562s...
-----Round 2-----
Red (-0.1954) Green (-0.0533)
```



```
  abcdefghi
0 RNBAKABNR
1
2     C  C
3 P P P P
4         P
5
6 p p p p p
7  c
8        c
9 rnbakabnr
```



```
  abcdefghi
0 RNBAKABNR
1
2        C
3 P P P P
4         P
5
6 p p C pcp
7  c
8
9 rnbakabnr
...Use 32.562437s...
-----Round 4-----
Red (0.0582) Green (-0.0557)
```

Challenge

1. Go from white board. It works.
2. Many input features.
3. Heavy calculation. It is hard to run the program in MAC and CPU only. So we hope minimize the calculation.
4. If it runs in multiple GPU, how to assign the data to GPU and guarantee that each GPU has same amount of data.

Future work

The Neural network is still a challenge. We are facing problems of how to deploy the program in aws. The self-playing can be multithreads, the data forward to neural network may not be able to devided into same amount to each GPU. If there are 3GPU, and data size is 4, there is gonna be a problem. The project will still focus on lightweight the program.

References

[1] Wikipedia:
https://en.wikipedia.org/wiki/Xiangqi

[2]Mastering the Game of Go without Human Knowledge

David Silver*, Julian Schrittwieser*, Karen Simonyan*, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis.

DeepMind, 5 New Street Square, London EC4A 3TW.

https://www.nature.com/articles/nature24270

[3] Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver,1∗ Thomas Hubert,1∗ Julian Schrittwieser,1∗ Ioannis Antonoglou,1 Matthew Lai,1 Arthur Guez,1 Marc Lanctot,1 Laurent Sifre,1 Dharshan Kumaran,1 Thore Graepel,1 Timothy Lillicrap,1 Karen Simonyan,1 Demis Hassabis1

1DeepMind, 6 Pancras Square, London N1C 4AG. ∗These authors contributed equally to this work.

https://arxiv.org/pdf/1712.01815.pdf