Using AlphaZero to Play Chinese Chess AlphaXiangqi (Final Presentation)

Yifeng Liu
Jiahua You

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.

Challenge

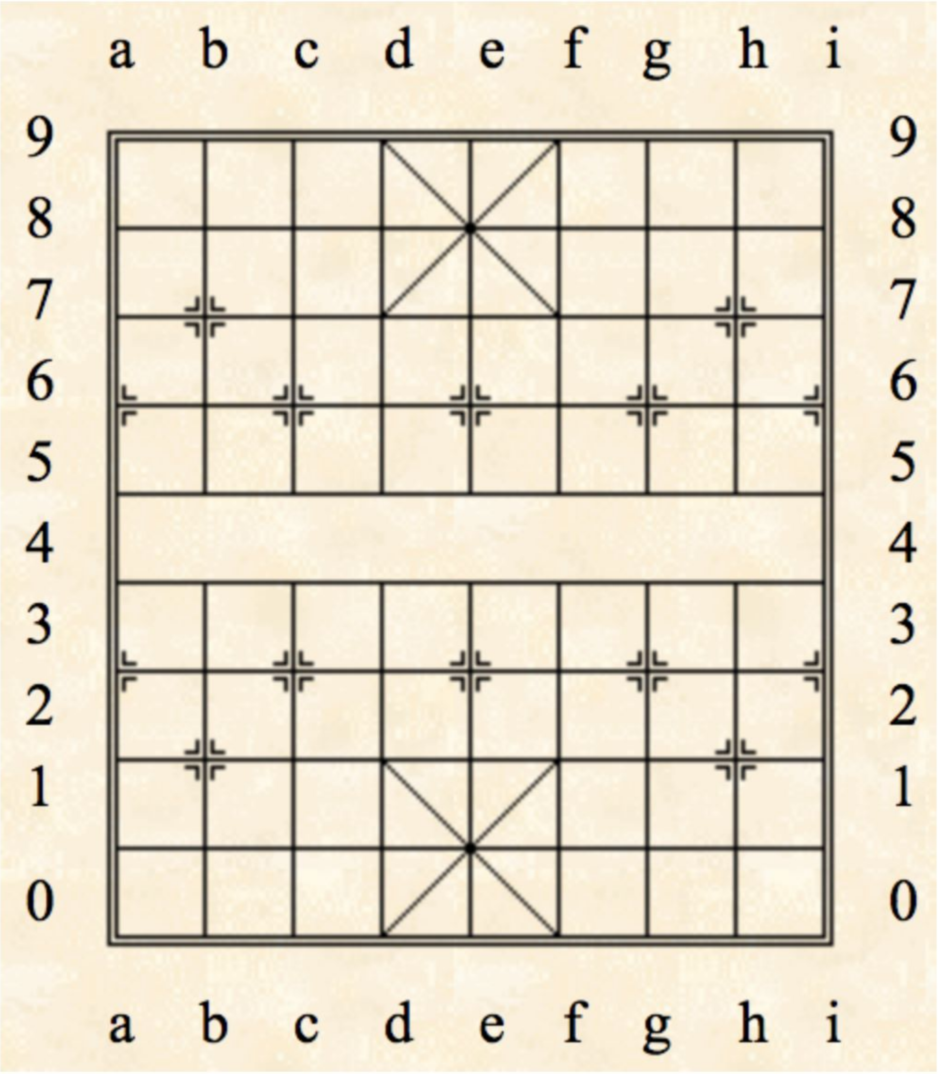
Design input feature

- 

- 

- 7 piece, so we have 7 character-plane

- 2 players, we have 14 character-plane



Papers study apply to AlphaXiangqi

- Mastering the Game of Go without Human Knowledge
    - MCTS
    - Neural network
- Masting Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm
    - generate chess manual
    - chess manual = input training neural network
    - neural network (finish training) will predict success rate

AlphaGo Zero(Xiangqi): learning from first principles

- No human data

- learns solely by self-play reinforcement learning, starting from random

- No human features

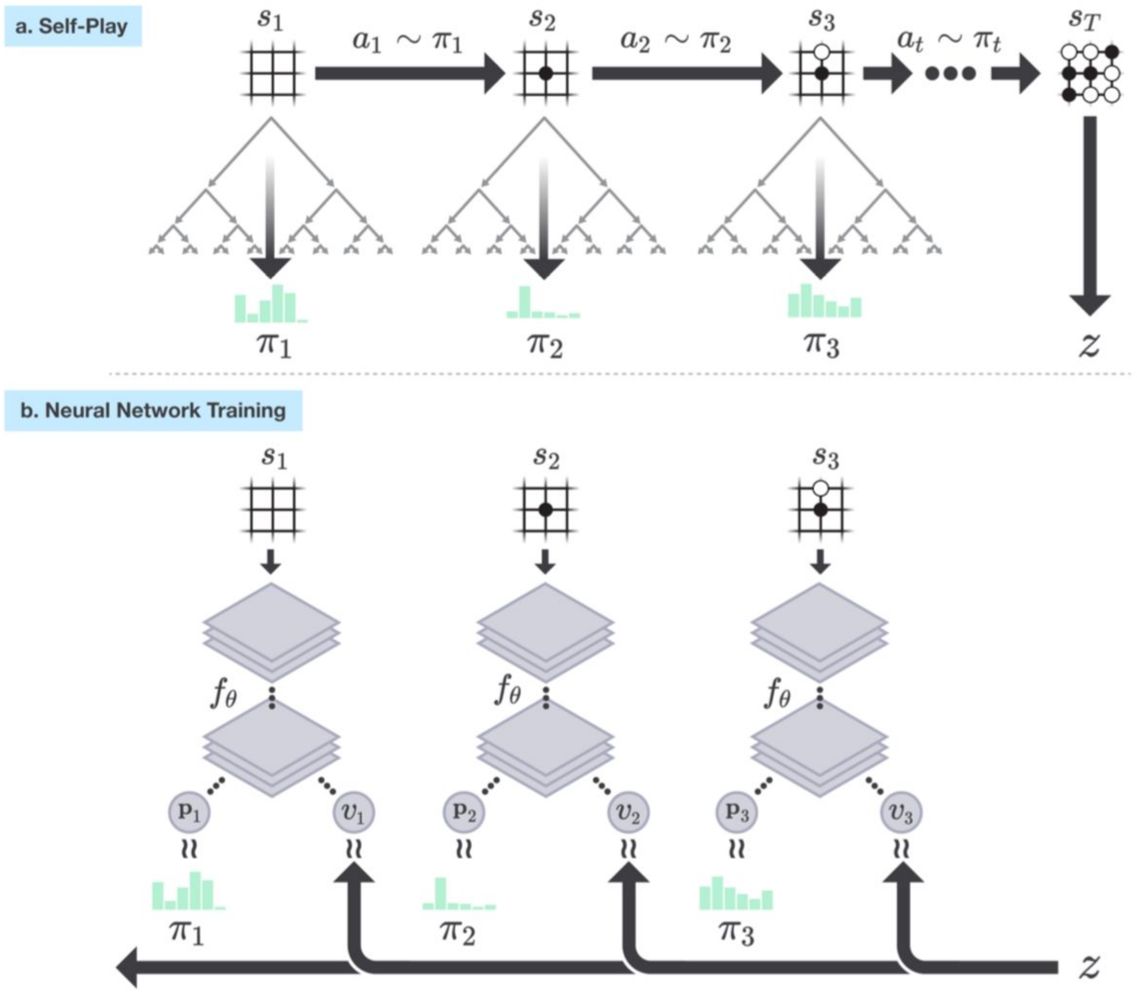
-only takes raw board as an input

- Single neural network

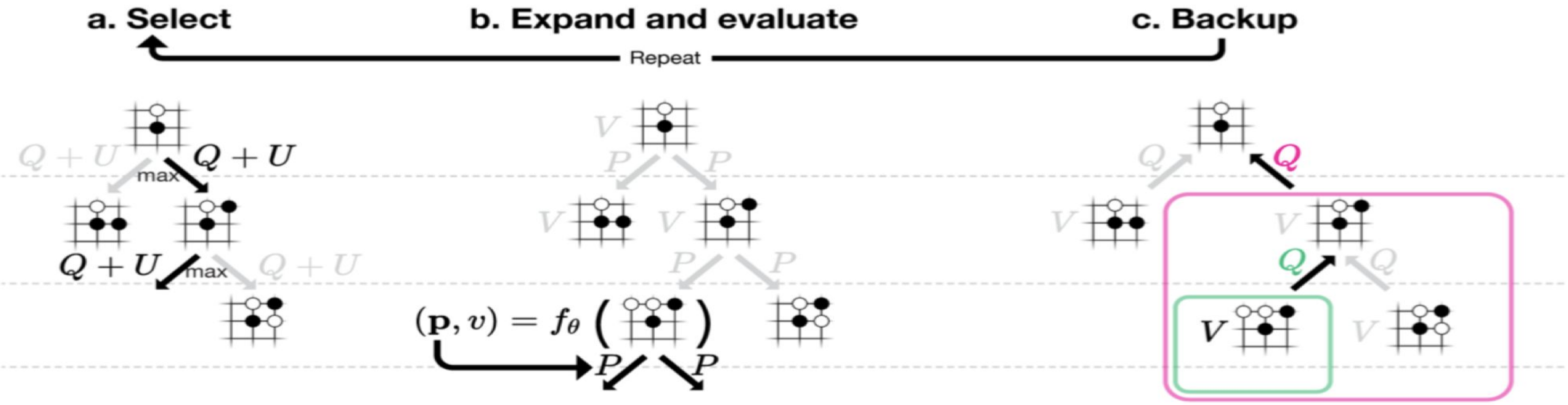
- Policy and value networks are combined into one neural network (resnet)

- Simple search

-MCTS, only use neural network to evaluate



a. Self-Play
s_1
$a_1 \sim \pi_1$
s_2
$a_2 \sim \pi_2$
s_3
$a_t \sim \pi_t$
s_T
π_1
π_2
π_3
z
b. Neural Network Training
s_1
s_2
s_3
f_θ
$\mathbf{p}_1$
v_1
$\mathbf{p}_2$
v_2
$\mathbf{p}_3$
v_3
π_1
π_2
π_3
z

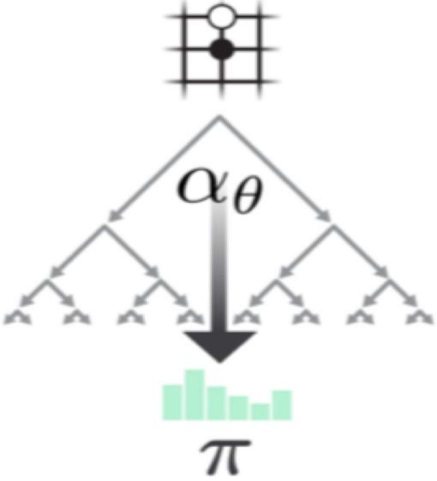


a. Each simulation traverses the tree by selecting the edge with maximum action-value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed).

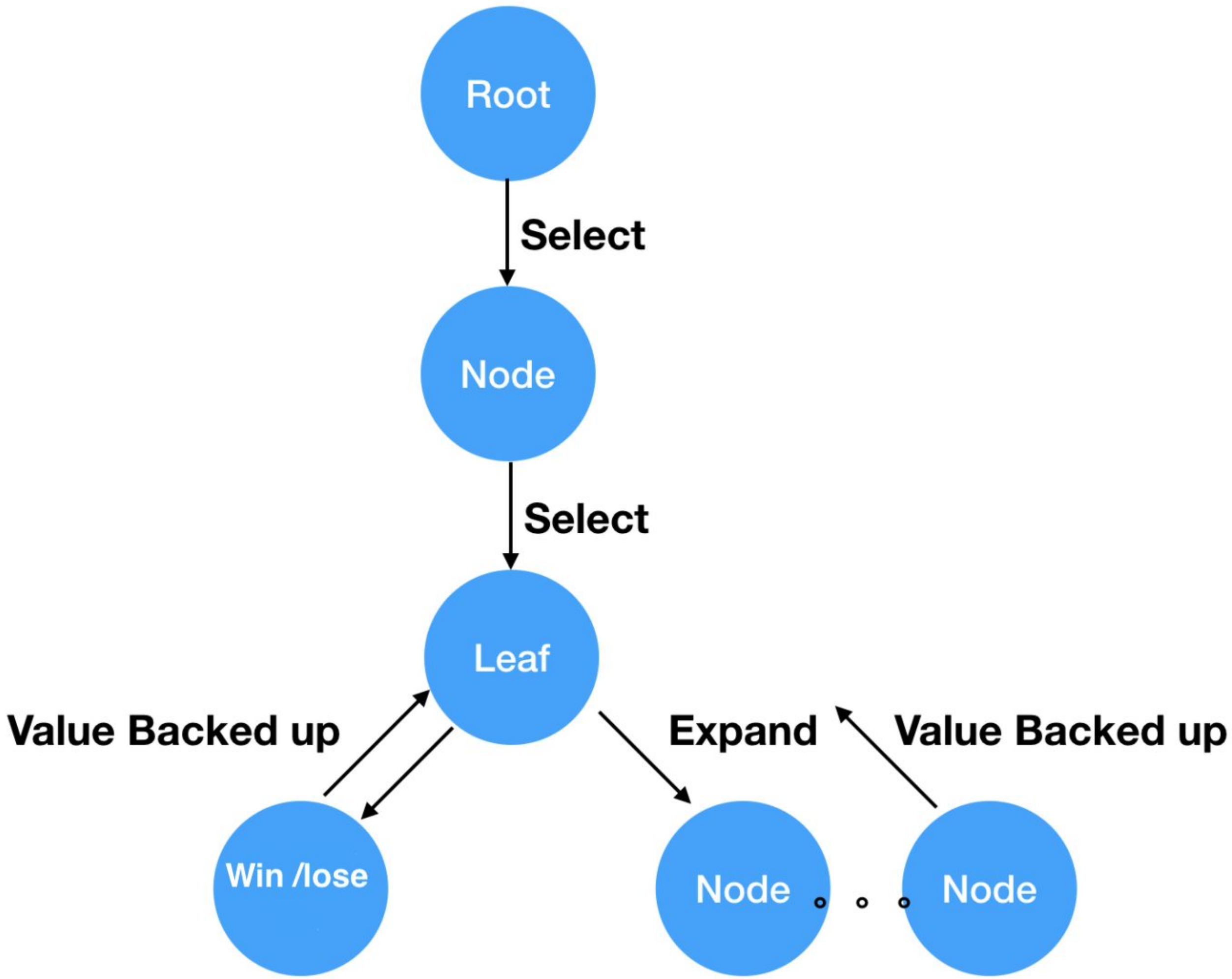
b. The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s,\cdot), V(s)) = f_\theta(s)$; the vector of P values is stored in the outgoing edges from s.

c. Action-values Q are updated to track the mean of all evaluations V in the subtree below that action.

d. Play

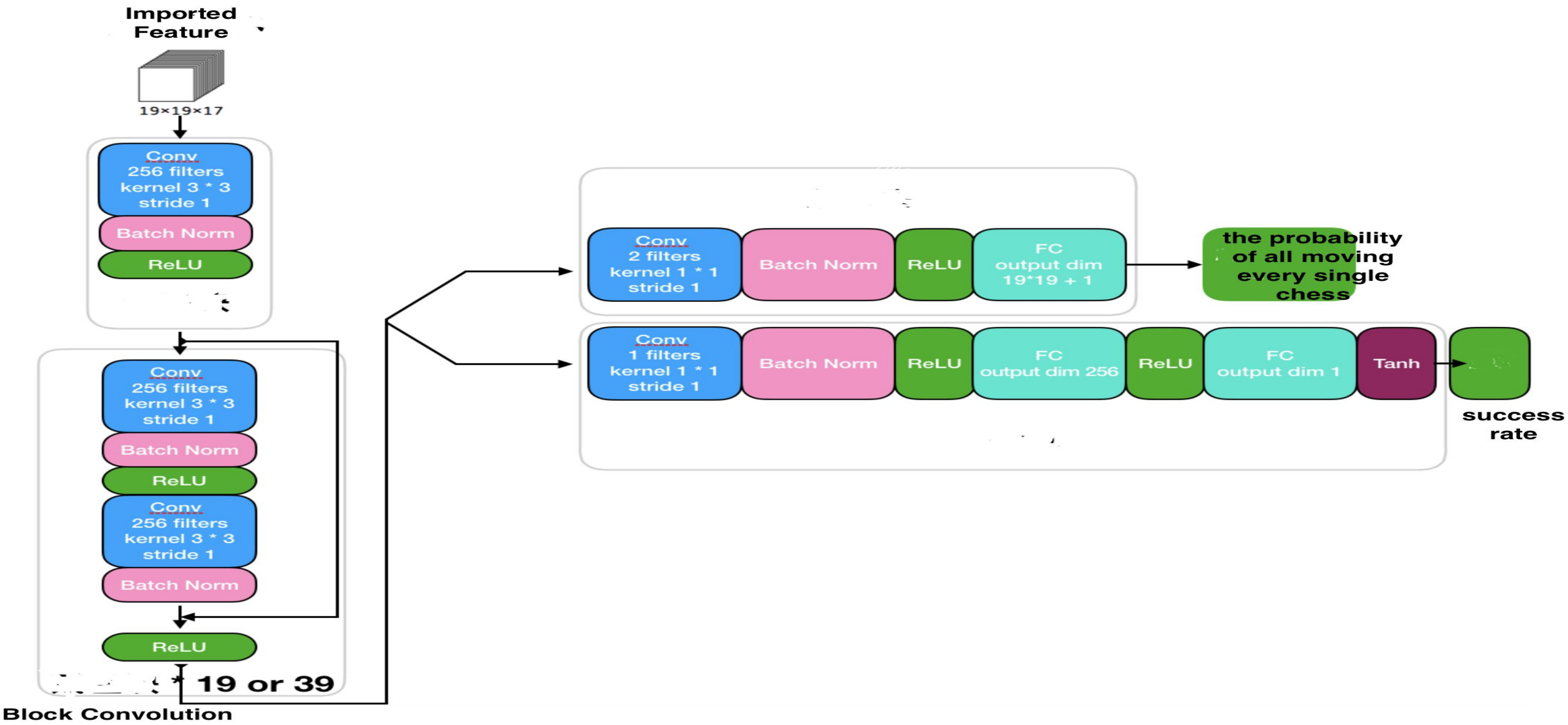


d. Once the search is complete, search probabilities π are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.



Root
Select
Node
Select
Leaf
Value Backed up
Expand
Value Backed up
Win /lose
Node
Node

The structure of Neural Network



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

Table S1: 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.

Expand and evaluate (Figure 2b). The leaf node s_L is added to a queue for neural network evaluation, $(d_i(p), v) = f_\theta(d_i(s_L))$, where d_i is a dihedral reflection or rotation selected uniformly at random from $i \in [1..8]$.

Positions in the queue are evaluated by the neural network using a mini-batch size of 8; the search thread is locked until evaluation completes. The leaf node is expanded and each edge (s_L, a) is initialised to $\{N(s_L, a) = 0, W(s_L, a) = 0, Q(s_L, a) = 0, P(s_L, a) = p_a\}$; the value v is then backed up.

Backup (Figure 2c). The edge statistics are updated in a backward pass through each step $t \leq L$. The visit counts are incremented, $N(s_t, a_t) = N(s_t, a_t) + 1$, and the action-value is updated to the mean value, $W(s_t, a_t) = W(s_t, a_t) + v, Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$. We use virtual loss to ensure each thread evaluates different nodes [69].

A: root

$$U(s,a) = c_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

B:leaf:expand

C:backup

```python
def back_up_value(self, value):
    self.N += 1
    self.W += value
    self.v = value
    self.Q = self.W / self.N
    self.U = c_PUCT * self.P * np.sqrt(self.parent.N) / ( 1 + self.N)
```

```
  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)
```