DeepShuai: a Deep RL Agent for Chinese Chess

Zihua (James) Liu, Kedao (Keven) Wang, Chengshu (Eric) Li
Project Mentor: Steven Hansen, Department of Psychology

Introduction

We used Reinforcement Learning to train a neural network to play Xiangqi (interchangeably Chinese Chess). Xiangqi is a traditional chess game much like chess itself. In terms of game tree complexity, Xiangqi is more complex than chess with a larger board and action space. To evaluate the performance of our agent, we will play our agent against a feature based, open source Xiangqi agent called Elephant Eye as did many relevant literature.

Environment

We implemented a chinese chess environment that will be the core of our agent's interaction with the opponent. The use of this environment is as followed.

- Generate dataset from Xiangqi Notation of 7,000,000 independent board positions for supervised learning
- Generate all legal moves given the current state
- Integrate with Elephant Eye, a commercial Xiangqi agent
- Allows the network to self-play to learn the end-game scenarios.

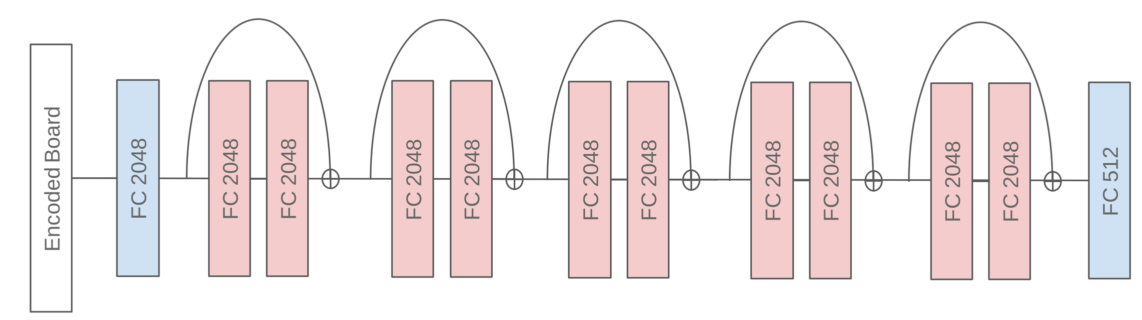
Data

- **70,000** complete expert Xiangqi games from an online Xiangqi database,
- On Average **100** moves per game.
- Total of **5,595,966** unique game positions with drawed games.
- **80% - 20%** of train-validation data split.
- Augmented dataset by including both player perspective with opposite win/loss label.

Results	Percentage
Red win	37.78%
Black win	27.90%
Draw	34.32%

Approach

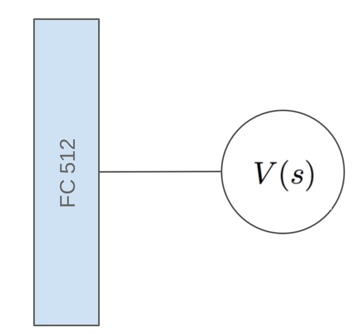
Like AlphaGo, we first used supervised learning to bootstrap RL. We represent each board state with an one-hot tensor and feed them into a 5 layer ResNet. We trained a value network and a policy network separately, using the same architecture, except the final layer.



Shared Architecture: 5 Layer ResNet

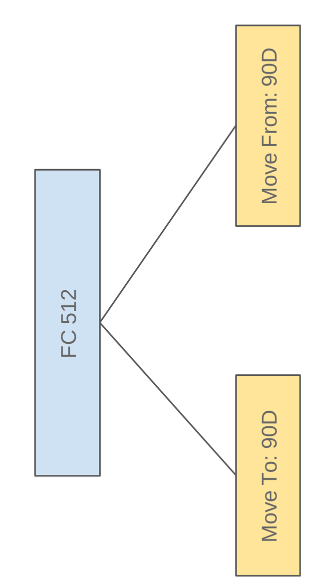
SL: Value Network

Output an estimated value of this state - a real number between -1 to 1, exponentially decayed by discount rate gamma.



SL: Policy Network

- Output a stochastic action, i,e, the distribution over the start position and the end position, represented by two 90-dimensional vectors
- Final action is chosen by taking the argmax of the element-wise product among all legal moves



$$L = -\log(\max(P_{start} * P_{end}))$$

RL: Value TD Learning

We used value network learned in supervised learning to extract a policy: choose the next state that corresponds to the highest value. We trained our network against Elephant Eye, a commercial Chinese Chess AI. Double Q learning is employed

$$L = (r + \gamma V(s', w^-) - V(s, w))^2$$

RL: Policy Gradient

We also used REINFORCE algorithm to train our agent. Moves agents made that lead to a winning game is treated as reward 1 and -1 otherwise. REINFORCE algorithm allows us to maximize probabilities for more positive actions and decreasing the probability of taking negative actions.

Result

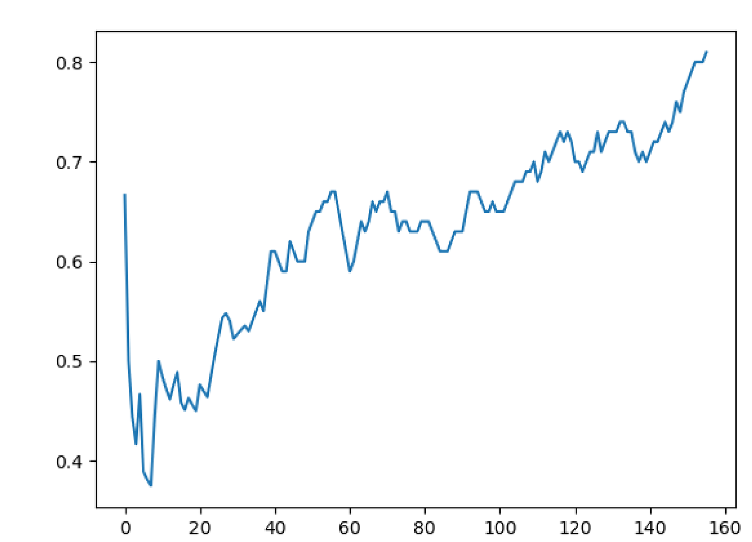
Supervised Learning:

- Value Network is traing on 7,000,000 expert move evaluated via RMSE. THis method achieved a **validation loss of 0.1877** with **gamma = 0.98**.
- Policy Network is trained on the same data evaluated via joint softmax loss between 'move-from' and 'move-to' position. This method achieved a **19.28%** accuracy on move-from position and **27.57%** on move-to position.

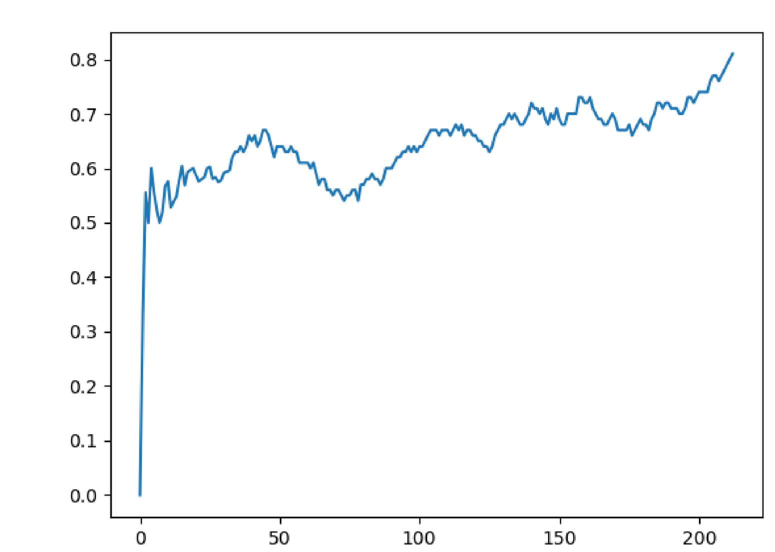
Reinforcement Learning:

To evaluate the game-play, we played our agent against a previous self, or vs. Elephant Eye. We use the winning percentage against Elephant Eye, a popular Chinese Chess AI used in prior work, to evaluate our network's performance.

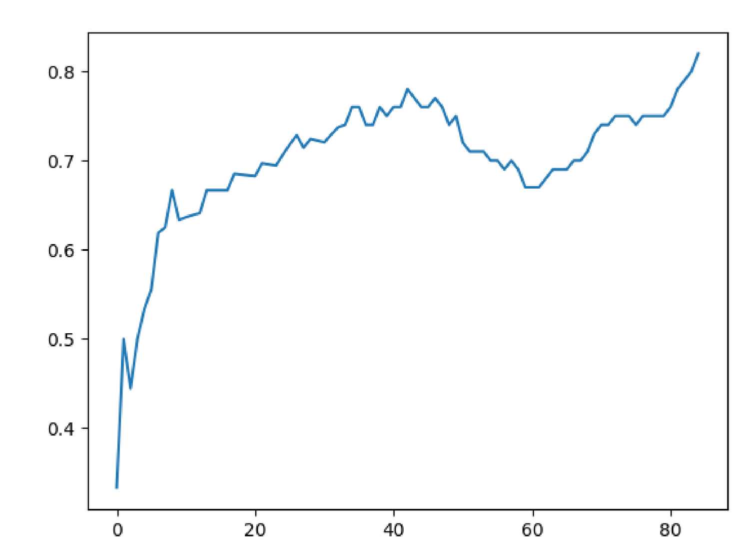
Particularly for self-play, we fixed weights for one agent and update the other. When the updated agent has over 80% win rate over the old agent, we update the old agent with the new weights, i.e. a new generation. Belows are some win rate graphs for different generations



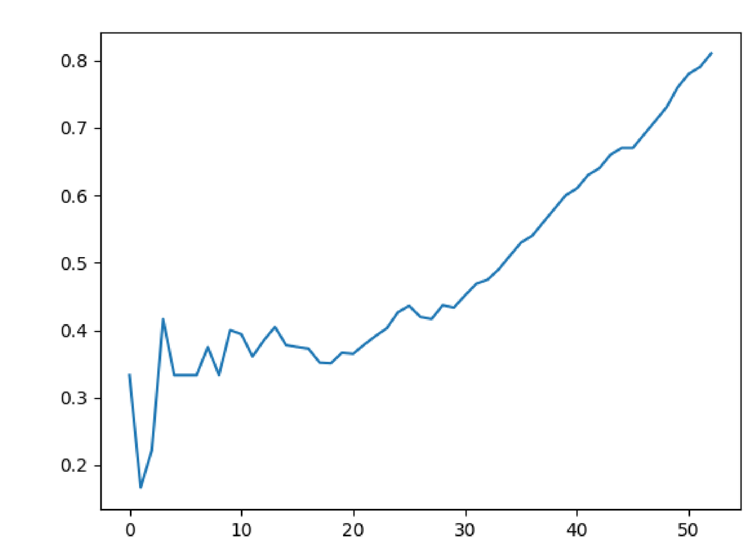
Win rate v. time on Gen 1



Win rate v. time on Gen 2



Win rate v. time on Gen 3



Win rate v. time on Gen 4