CS230 - Learning to Play Minichess Without Human Knowledge

Karthik selvakumar Bhuvaneswaran (karthik0@stanford.edu)

Abstract

- Implementing a self play based algorithm using neural nets has become popular after the huge success of Alpha Zero by Deep Mind.
- Replicating the results for games with larger search space like chess requires scaling.
- We develop a scaled up version of alpha-zero-general for the game of Minichess and evaluate our learning algorithm with various baselines.

Keywords: *CNN, Reinforcement Learning, Distributed Computing, Monte Carlo search*

Introduction

- Self play and improve without any human knowledge.
- MCTS provides provides ground truth to compare and learn.
- 5X5 chess board with Gardner layout will be used for our training.

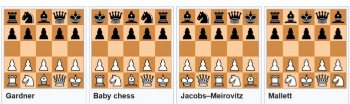


Figure 1:Popular Minichess Board Layouts

- Single Neural network used for both Policy and Value evaluation
- We will use the following Loss function

$$l = -\Sigma(v_\theta(s_t) - z_t)^2 + \vec{\pi}_t log(\vec{p}_\theta(s_t))$$

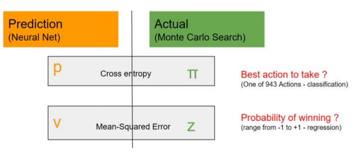


Figure 2:Parameters in network and choice of loss functions

Distributed Architecture

Three major components in self play and learn are:

- *Training Data Generator* - Plays games and generates data for training.
- *Trainer* - Consumes the data from Training Data Generator, compares against MCTS and learns.
- *Pitter* - Compares two models and publishes a winner model.

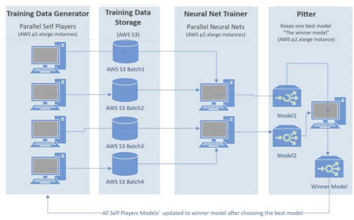


Figure 3:Scaling up the training using Distributed Architecture

Neural Network Model

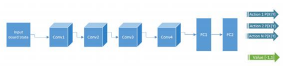


Figure 4:CNN Layers with both Policy and Value output

Hyperparameter	Value
MCTS Simulations	200
Exploration (cpuct)	1
Learning Rate	0.0005
Update Threshold	0.5
Arena Compare	20
Iterations	100
Episodes	100
Data Augmentation	Two way Symmetry

Hyperparameter	Value
Value Activation	tanh
Policy Activation	Softmax
Batch Size	128
Number of Layers	7 (4 CONV, 3 FC)
Regularization	Dropout (0.2)
Optimizer	Adam
Normalization	Batch Normalization

Figure 5:Hyperparams used for Pre-processing and Training

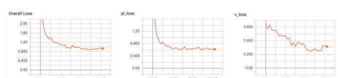


Figure 6:Loss values after each epoch

Training Performance

- 2.5 times improvement in training speed with distributed setup.

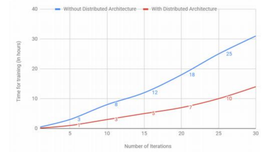


Figure 7:Single CPU vs Distributed Architecture

Baseline Comparison

Baseline	Color	Won	Lost	Draw
Random Player	White	10	0	0
	Black	10	0	0
Greedy Player	White	10	0	0
	Black	3	0	7
Model Version 1	White	10	0	0
	Black	7	0	3
Model Version 5	White	10	0	0
	Black	0	0	10
Model Version 15	White	0	0	10
	Black	0	0	10

Figure 8:Results of pitting Best Model (V21) with other players

After 21 iterations of training, when evaluated:

- Defeats random player 100%.
- Defeats greedy player 100% when Neural Net takes first turn (White)

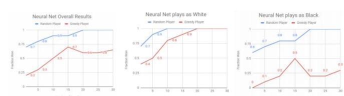
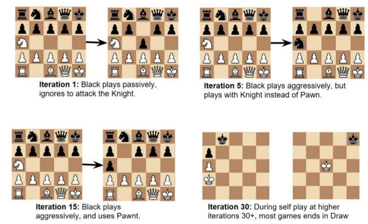


Figure 9:Performance of Neural Net over other baselines

Chess Layout	Baseline	Color	Won	Lost	Draw
BabyChess	Random	White	10	0	0
		Black	9	1	0
	Greedy	White	10	0	0
		Black	0	10	0
Mallot	Random	White	10	0	0
		Black	9	1	0
	Greedy	White	10	0	0
		Black	10	0	0

Figure 10:Trained on Gardner and transferred to other layouts

Observations



Conclusion

- Trained model beats the random, greedy baselines and performs decently on other layouts.
- Monte Carlo Tree Search and CNN can approximate search space as large as $9 * 10^{18}$ as we seen in Minichess.
- Parallelizing self play, training and pitter by leveraging cloud services improves the performance substantially

References

- Gardner's Minichess Variant is solved. Mehdi Mhalla et al. *arXiv e-print (arXiv:1307.7118)*
- Learning to Play Othello Without Human Knowledge Surag Nair et al
- Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. Silver et al. 2017a

Contact Information

- Web: https://github.com/karthikselva/alpha-zero-general/tree/minichess
- Demo Video: https://youtu.be/NxzABCCzYCE
- Mentor: patcho@stanford.edu