Improving Neural Network on the Game of Renju

Naxin Chen NC352@CORNELL.EDU
Yiwei Ni YN254@CORNELL.EDU
Guandao Yang GY46@CORNELL.EDU
Christopher De Sa CDESA@CORNELL.EDU

Abstract

The effort in AlphaGo and AlphaGoZero has shown that neural network could be used to learn a good heuristics in game playing, but there remains two challenges in using deep learning as heuristic in the game of Renju. First, a successful Renju heuristic needs to be trained in millions of game states, which takes a long time. Second, during game playing, the inference speed of deep neural network becomes a bottle-neck. In this report, we presents our attempts to leverage data parallelism, low-precision training, and deep knowledge distillation to address the above-mentioned two challenges.We provide detailed experiments both in training and game-playing to evaluate the effectiveness of these three approaches.Results show that both half-precision training and distillation are unstable during the training stage. On the other hand, data parallelism has shown to be an effective drop-in improvement on both the training and inference stage.

Introduction

Renju is a board game for two players to take turns in placing black/white stones on a 15x15 board. The goal is to achieve an unbroken row of five or more stones horizontally, vertically, or diagonally. In contrast to simple five-in-a-row, additional rules are used to control first-hand advantages. First, Renju starts with a curated set of opening to ensure the fairness of the game. Second, the first-hand player has a series of forbidden move as illustrated in Figure 1. With these rules, Renju becomes a much more sophisticated games, and has gained interests from AI developers for years (gom, 2017).

However, most top performing AIs are based on tree search or heuristic algorithms, which requires a lot of domain knowledge. Inspired by the success of AlphaGo (Silver et al., 2017; 2016), we would like to fill in the gap by training a neural network to learn the Renju heuristic from human expert data.

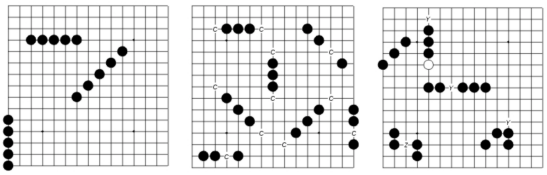

Figure 1. left: Renju possible winning lines; middle: possible open fours with move c (typical winning moves); right: forbidden move example, marked as Y; Different from five-in-a-row, Renju forbids the first-hand player to play the following winning moves: 4-4, 3-3, or overline.

While reproducing techniques of AlphaGo in Renju, we realized that training and inference of deep neural networks are unrealistically time consuming. In this paper, we share three attempts to improve our neural network's training and inference speed. To improve the training speed, we utilize data-parallelism and half-precision training. In order to increase inference speed, we manually split the dataset into two parts manually and train a shallower network with the split dataset. We also borrow distillation technique from (Hinton et al., 2015), trying to use a deeper network to teach a smaller one.

Experiment results show that data parallelism is most effective in improving both the training and the inference of the dual-resnet. Both half-precision training and distillation show either instability during training or limited effect in the improvement of training or inference time.

Related Works

Renju AIs : Currently, there are 7 Renju AIs in the Renju tournament, such as Yixin. To the best of our knowledge, they are all based on tree search or heuristic algorithms. Our AI on the other hand train

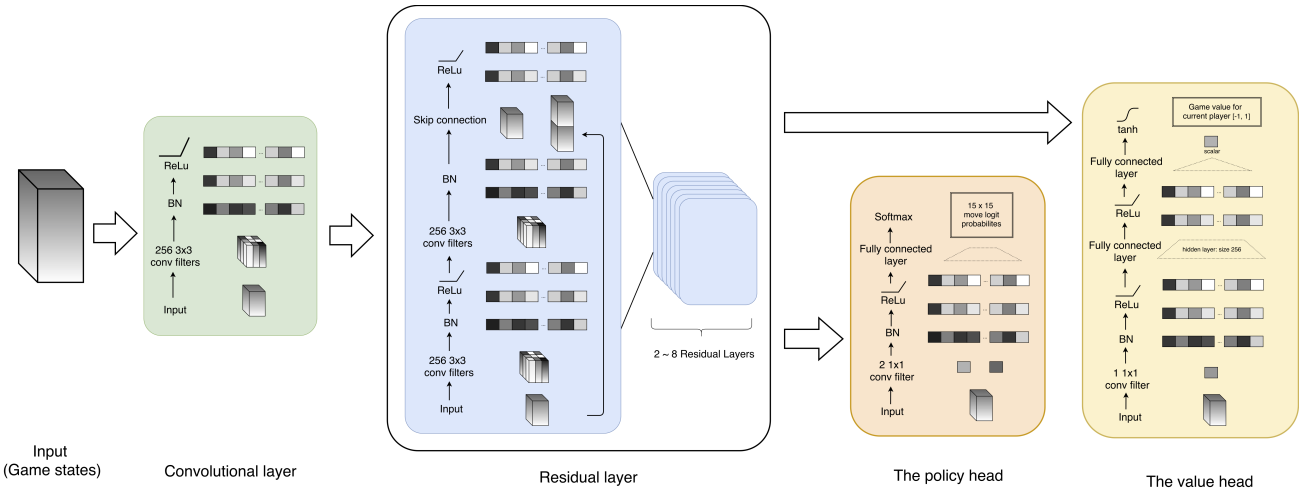

Figure 2. Neural network architecture. We use Dual-ResNet f_θ with parameters θ, with a single raw board s as the input. f_θ outputs the next move probability distribution $p = Pr(a|s)$ and a corresponding value v, where a is the next possible move from board s. The policy and value network, p and v, provide the prior probability, P, and the value function, V, for the tree search, respectively.

a deep neural network to replace the domain-specific heuristic. There has been a previous work on move prediction in Gomoku using deep learning (Shao et al., 2016). Several course projects from Stanford also use ConvNets and RNNs on similar tasks(Zhao). All of these projects have showed successful trainingr esults. Our project use more sophisticated network structure adapted from AlphaGoZero (Silver et al., 2017) and better data-set to achieve much better training results. There have been several similar attempts to reproduce AlphaGo Master or Zero in Renju or a simpler version of Renju called Gomoku(AI-Zone)(welkincrowns). None of those project show successful results competing with heuristic-base AIs.

Improving Techniques: Previous works on low-precision training inspired us to use such technique to try speeding up our training process(Gupta, 2015). To speedup network inference and training, various distillation techniques(Hinton et al., 2015) has been introduced where a student model would be training not only on the target label but also the logits from a pre-trained teacher model.

Methods

Architecture

We use the network architecture in AlphaGoZero (Silver et al., 2017). Detailed architecture is shown in figure 2. Our input feature is a single-channel image representing a game state at Renju. We simply put $+1$ at the position when there is a white stone, and -1 at the position of a black stone. We apply all eight of Dihedral transformations to augment the training dataset, which results in a dataset with more than 25 millions game states. With this network structure, even if we use only 8 residual blocks (few number compared to 20-40 used in AlphaGo), it takes hours to go through one epoch in a Titan Xp GPU. In order to obtain reasonable result or experiment different hyper-parameter, we need to speed up the training process.

Data Parallelism

Data Parallelism has been used as a drop-in speed up tools in many applications when there are multiple GPU available. We use two types of data-parallelism to improve both the training and inference time.

During the training process, we use PyTorch's torch.nn.DataParallel feature to distribute training batch across available CUDA devices. Using two Nvidia 1080Ti GPU we would expect a linear speedup on the training time.

Moreover, observed that there are significant difference in the strategy between the first-hand player (who will play black stone) and the second-hand player (who will play black stone), we might need to increase the model capacity significantly to capture these information. As a result, the model takes longer to inference. To address this challenge, we decided to double not the depth of the model but the width of it. We manually create data-parallelism by splitting the dataset into boards to be played by second-hand player (i.e, white dataset) and boards to be played by the first-

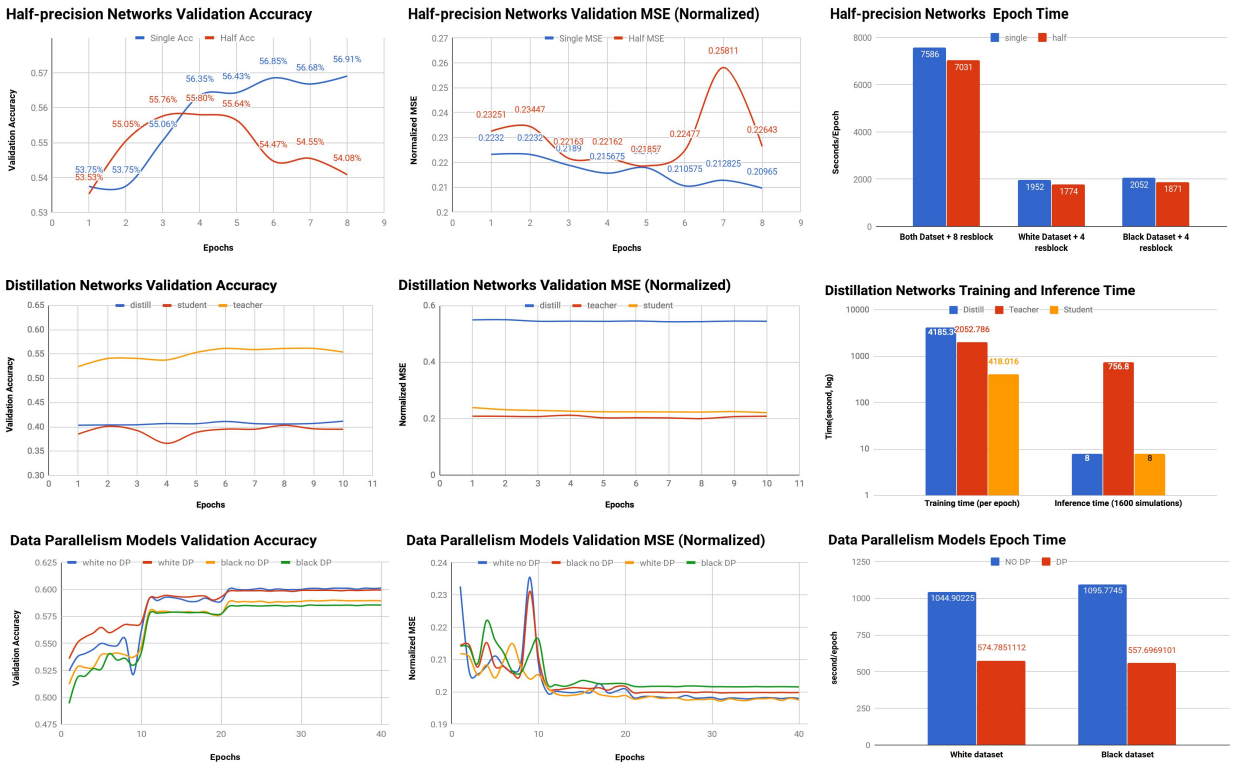

Figure 3. This figure contains our experiment results. The first column shows the comparison of policy network's validation accuracy. Second column shows the normalized mean square error. Third column presents wall-clock time comparison. The first rows compares models trained with or without half-precision. The second rows compare the teacher, student, and distilled models. The third column shows results with and without data parallelism.

hand player (i.e. black dataset). We will train each of these two data-sets with a dual-ResNet. As a result, during inference time, we only need to run a model with half the depth but contain the same capability.

Distillation and Low-precision

Recent works on speeding up training using low precision float point numbers(Gupta, 2015) motivated us to try using half precision trianing for faster training. For this particular experiment we are going to use PyTorch's half precision float tensor on Nvidia Titan Xp GPU.

In order for our AI to be competitive against other tree search based Renju AIs, we need to speedup the inference cost without sacrificing our network's accuracy. To avoid the trade off between faster network inference time and higher network accuracy, we decided to implement neural network knowledge distillation techniques (Hinton et al., 2015) to transfer the knowledge from a bigger network to a smaller one. Our experiment implementation is based on a teacher-student architecture where the student network is trained both on the correct label prediction and the teacher network's logit.

Experiments

Preprocessing and Dataset

We use two datasets: RenjuNet(Ren) and RenjuOffline(Ren, 2017). RenjuNet contains 50,064 games while RenjuOffline contains 104,948 games. RenjuNet's games are played by human experts across the world in real time. The games in RenjuOffline are slow paced games played online without time restriction, which is known to have better quality. Due to the special opening restriction applied to Renju, we removed the first 6 moves from all the games, and generate game states from all the game records. With this method, we obtain a data-set with 3,146,736 game states, all of which annotated with a human expert's move and its game results (+1 if wins and -1 if lose).

Half Precision Training

We trained our largest network (with 8/2 residual blocks) with the same parameter using half-precision tensors in a Titan X GPU to compared with our model trained with single precision. Both models use stochastic gradient descent as optimizer, and scheduled a learning rate drop from 0.1 to 0.01 after 5 epochs. We killed the model after 10 epochs due to the deteriorated performance. The results are posted in the first row of figure 3.

As we could see from the figure, when half-precision training experienced instability after the dropping of the learning rate at epoch 5. It appears in the figure that both the validation accuracy and the mean square error is suffered from such affect. Our hypothesis is that the batch-normalization might have interfered with the half-precision training, resulting in several zero gradient. We also tried using Adam, but it's hard to not getting NaN without turning up the epsilon to affect the convergence. Moreover, the time chart shows that the speed-up half-precision training brought is highly marginal (less than 10%). As a result, we conclude that the drop-in half-precision training isn't suitable in our setting.

Distillation

In the distillation experiment, we try to use a student network without any residual-blocks to learn the policy of a teacher network with 4 residual-blocks. The value of the student network is still learned by itself. We used significantly higher weight on the loss on the soft-target. We show the results of models with the best hyper-parameter in figure 3.

As we could see that distillation is hard to get it working. The validation accuracy shows that distillation only helps to improve the policy accuracy in a very marginal way. Yet due to the additional target, the value-network finds it even harder to learn the correct value function. Nonetheless, the wall-clock time profiling shows that it's highly desirable if we could get the distillation to work since it could provide roughly 100 folds improvement in inference time.

Data Parallelism

In our attempt to incorporate data parallelism into the training process, we utilized PyTorch's built-in data parallel feature which evenly distributes the batch to available CUDA devcies. The To evaluate the effectiveness of data parallelism, we trained a data-parallel network with 2 residual blocks on separate black and white dataset. The baseline we test against is the same model trained without data parallelism. The model with data parallelism is trained with 2 NVIDIA 1080Ti GPUs and the other model is trained with only 1 1080Ti GPU. These two networks are trained using SGD with learning rate dropping by 10 folds every 10 epochs. The training batch size is the same for the two models, thus for the data-parallel model each GPU would be working with half of the training batch-size. Results are shown in figure 3.

We could see from both the graph for the validation accuracy and the normalized MSE that there are only minimal difference between the model trained with data parallelism and the one without. Data parallelism provide expected linear speed up as the time taken per-epoch is reduced by half.

Model	Estimated Elo Score
4 rblocks + separate colors	**1898**
Pela (0.01s)	1835
2 rblocks + separate colors + DP	1737
2 rblocks + seperate colors	1727
4 rblocks + both colors	1712
2 rblocks + both colors	1483
0 rblocks + distillation	1453
4 rblocks + half precision	1319
0 rblocks + both data	1299

Table 1. The statistics of an internal tournament among all policy networks. We could see that training with data parallelism has little influence in performance. Models trained with separated data-sets in general perform better.

Elo Score

To evaluate the quality of policy networks, we run an internal tournaments among all the policy networks mentioned in previous sections and compute the Elo score. We also add PELA(0.01s), one of the AI ranked in GomokuCup(gom, 2017), into the tournament as a reference point to estimate Elo scores. The results are shown in table 1. The results shows that manual data parallelism increases performance significantly. Adding data parallelism during training boosts the performance marginally. Adding distillation does have positive effect in the performance compare to networks with the same size, but it's far from the performance of the teacher network and takes even longer to train. Half-precision training seems to have significant adverse effect in game-playing. As a result, we conclude that splitting the data-sets manually and applying data parallelism are two techniques we found beneficial in the setting of training Renju heuristics using deep learning.

Future Work and Acknowledgements

Due to the limitation of PyTorch implementation, we haven't tried to use model compression techniques, which will be the main direction for improvement. We could also improve our results by leveraging reinforcement learning. We want to acknowledge that this project is based on the project Guandao Yang, Naxin Chen, Ransen Niu, Vaidehi Patel, and Hanqing Jiang have been working for CS 6784.

References

The renju international federation. URL http://www.renju.net/.

Renju offline, 2017. URL http://renjuoffline.com/main.php.

Gomocup the gomoku ai tournament, 2017. URL http://gomocup.org/.

AI-Zone. Alphagobang - implement alphazero in gobang game. URL https://github.com/AI-zone/AlphaGoBang.

Gupta, Suyog; Agrawal, Ankur; Gopalakrishnan Kailash; Narayanan Pritish. Deep learning with limited numerical precision. 2015.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Shao, Kun, Zhao, Dongbin, Tang, Zhentao, and Zhu, Yuanheng. Move prediction in gomoku using deep learning. In *Chinese Association of Automation (YAC), Youth Academic Annual Conference of*, pp. 292–297. IEEE, 2016.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Silver, David, Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, Hubert, Thomas, Baker, Lucas, Lai, Matthew, Bolton, Adrian, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

welkincrowns. Alphago-unlimitedgomokuworks. URL https://github.com/welkincrowns/AlphaGo-UnlimitedGomokuWorks.

Zhao, Rongxiao. Convolutional and recurrent neural network for gomoku.