IMPROVED ARCHITECTURES FOR COMPUTER GO

Tristan Cazenave
Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE
75016 PARIS, FRANCE
Tristan.Cazenave@dauphine.fr

ABSTRACT

AlphaGo trains policy networks with both supervised and reinforcement learning and makes different policy networks play millions of games so as to train a value network. The reinforcement learning part requires massive amount of computation. We propose to train networks for computer Go so that given accuracy is reached with much less examples. We modify the architecture of the networks in order to train them faster and to have better accuracy in the end.

1 INTRODUCTION

Deep Learning for the game of Go with convolutional neural networks has been addressed by Clark & Storkey (2015). It has been further improved using larger networks Maddison et al. (2014); Tian & Zhu (2015). AlphaGo Silver et al. (2016) combines Monte Carlo Tree Search with a policy and a value network.

Deep neural networks are good at recognizing shapes in the game of Go. However they have weaknesses at tactical search such as ladders and life and death. The way it is handled in AlphaGo is to give as input to the network the results of ladders. Reading ladders is not enough to understand more complex problems that require search. So AlphaGo combines deep networks with MCTS Coulom (2006). It trains a value network in order to evaluate positions. When playing, it combines the evaluation of a leaf of the Monte Carlo tree by the value network with the result of the playout that starts at this leaf. The value network is an important innovation due to AlphaGo. It has helped improving a lot the level of play.

One of the problems about training a value network is that millions of games have to be played by the policy network against different versions of itself in order to create the data used to train the value network. It is therefore interesting to find a way to learn with less training examples so as to reduce the bottleneck of playing millions of games. Learning with less examples also often implies that in the end the accuracy of the network on the training set is greater.

Residual Networks improve the training of very deep networks He et al. (2015). These networks can gain accuracy from considerably increased depth. On the ImageNet dataset a 152 layers networks achieves 3.57% error. It won the 1st place on the ILSVRC 2015 classification task. The principle of residual nets is to add the input of the layer to the output of each layer. With this simple modification training is faster and enables deeper networks.

Residual networks were recently successfully adapted to computer Go Cazenave (2016a). As a follow up to this paper, we propose improved architectures for residual networks. We use enriched inputs, more comprehensive training and testing examples and experiment with deeper networks that give better accuracy and stronger play.

The second section details different layer architectures for computer Go, the third section gives experimental results, and the last section concludes.

2 DIFFERENT LAYERS FOR COMPUTER GO

The usual layer used in computer Go program such as AlphaGo Maddison et al. (2014) and DarkForest Tian & Zhu (2015) is composed of a convolutional layer and of a ReLU layer as shown in figure 1.

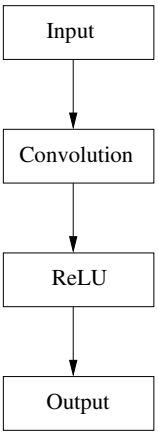

Figure 1: A usual layer.

The residual layer used for image classification adds the input of the layer to the output of the layer using addition and identity. It is shown in figure 2.

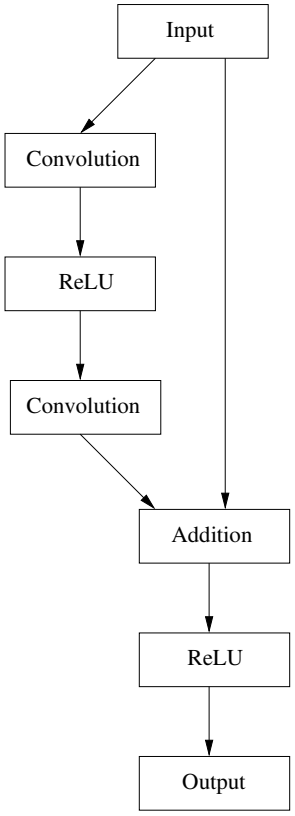

Figure 2: A residual layer for Image Classification.

The residual layer we use for Computer Go is shown in figure 3 Cazenave (2016a). It simply adds the input of a layer to the output of the 3×3 convolutional layer. It then uses a ReLU layer before the output. The output of a residual layer is the input of the next residual layer.

In the code of the open source DarkForest Go program, Spatial Batch Normalization Ioffe & Szegedy (2015) is used after the ReLU layer as shown in figure 4.

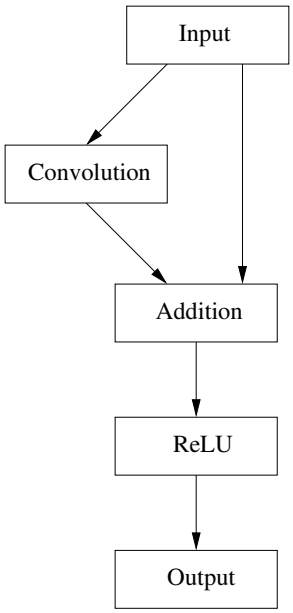

Figure 3: A residual layer for computer Go.

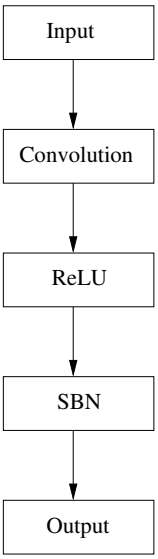

Figure 4: A layer of DarkForest.

Inspired by DarkForest we tried to add Spatial Batch Normalization after the ReLU layer in our residual layer. The layer that gave the best results is given in figure 5. It simply adds a Spatial Batch Normalization after the ReLU layer and outside of the residual block. This is a new architecture that we propose and test in this paper.

The Torch Collobert et al. (2011) code used for the hidden layers is simply:

```
convnet:add (nn.ConcatTable ()
:add (cudnn.SpatialConvolution (nplanes, nplanes,3, 3, 1, 1, 1, 1))
:add (nn.Identity ()))
:add (nn.CAddTable (true))
:add (cudnn.ReLU ())
:add (cudnn.SpatialBatchNormalization (nplanes))
```

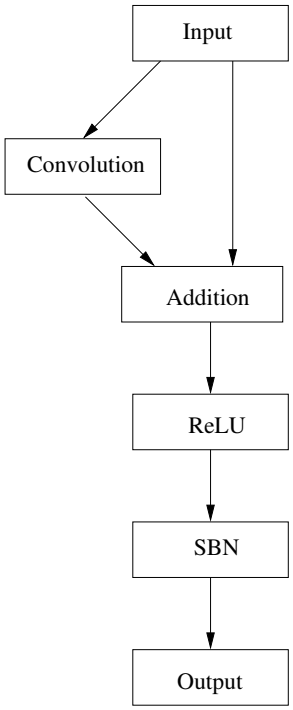

Figure 5: A residual layer with Spatial Batch Normalization.

The input layer of our network is also residual. It uses a 5×5 convolutional layer in parallel to a 1×1 convolutional layer and adds the outputs of the two layers before the ReLU layer. It is depicted in figure 6.

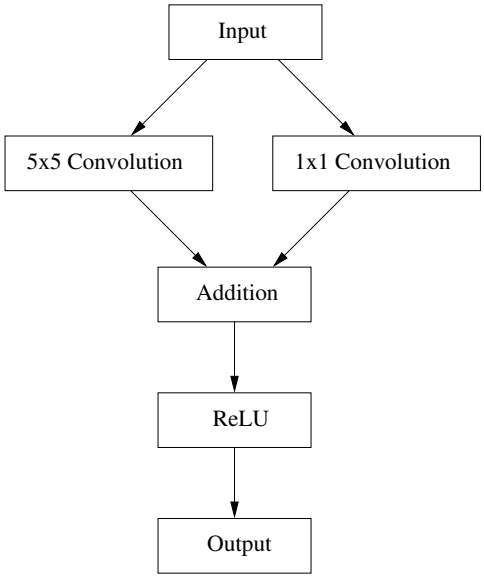

Figure 6: The first residual layer of the network for computer Go.

The output layer of the network is a 3×3 convolutional layer with one output plane followed by a SoftMax. All the hidden layers use 256 feature planes and 3×3 filters.

3 EXPERIMENTAL RESULTS

In this section we will explain how we conducted the experiments evaluating deep residual networks. We first present the data that was used for training and testing. We then describe the input planes of the networks and the training and testing phases with results given as percentages on the test set. We finish the section describing our Go playing program Golois.

3.1 DATA

Our training set consists of games played between 2000 and 2014 on the Kiseido Go Server (KGS) by players being 6 dan or more. We exclude handicap games. Each position is rotated and mirrored to its eight possible symmetric positions. It results in 160 000 000 positions in the training set. When training reaches the last position of the training set it starts again with the first one. The test set contains the games played in 2015. The positions in the test set are not mirrored and there are 100 000 different positions in the test set.

The dataset is similar to the AlphaGo and the DarkForest datasets, all the games we have used for training are part of these two other datasets. AlphaGo also uses games by weaker players in its dataset Maddison et al. (2014), instead we only use games by 6 dan or more, it probably makes the dataset more difficult and more meaningful. The AlphaGo dataset is not available, also it would help to have the 30 000 000 games played by AlphaGo against itself so as to train a value network but this dataset is not available either.

3.2 INPUT AND OUTPUT PLANES

The networks use 42 19×19 input planes: three planes for the colors of the intersections, one plane for the third line, one plane filled with one if there is a ko, one plane with a one for the ko move, one plane with the ownership of each intersection computed after one hundred random playouts, one plane for the criticality Coulom (2009), one plane for the AMAF values, ten planes for the liberties of the friend and of the enemy colors (1, 2, 3, 4, ≥ 5 liberties), twelve planes for the liberties of the friend and of the enemy colors if a move of the color is played on the intersection (1, 2, 3, 4, 5, ≥ 6 liberties), one plane to tell if a friend move on the intersection is captured in a ladder, one plane to tell is an enemy move on the intersection is captured in a ladder, one plane to tell if a string can be captured in a ladder, one plane to tell if a string can escape a ladder, one plane to tell if a friend move threatens to capture in a ladder, one plane to tell if an enemy move threatens to capture in a ladder, and five planes for each of the last five moves.

The output of a network is a 19×19 plane and the target is also a 19×19 plane with a one for the move played and zeros elsewhere.

3.3 TRAINING

Networks were trained with a minibatch of either 50 or 100 and an initial learning rate of $0.2 \times minibatch = 20.0$. The error criterion is the mean square error and the training algorithm is SGD with a momentum of 0.9. All networks use 256 feature planes, a 5×5 convolutional layer for the input layer and then only 3×3 convolutional layers.

The different network architectures we experimented with are:

- net_{α}, the usual 13 layers convolutional network as used in AlphaGo Maddison et al. (2014); Silver et al. (2016),
- net_{dark}, with 13 layers and convolutional layers followed by ReLU and Spatial Batch Normalization as used in DarkForest Tian & Zhu (2015),
- net_{13}, a residual 13 layers network,
- net_{20}, a residual 20 layers network,
- $net_{13/sbn}$, a residual 13 layers network with Spatial Batch Normalization.

We used Torch Collobert et al. (2011) for training the networks. Training a 13 layers network on 5,000,000 examples with Torch, CUDA 8.0 and CUDNN takes approximately three hours on a GTX

1080 GPU. This is 9 times faster than our previous experiments with a K40 Cazenave (2016a). The evolution of the percentage on the test set is given in table 1. Each line corresponds to 5,000,000 more training examples.

We see that all proposed layers improve much on the usual layer. The DarkForest layer is slightly better than the residual layer with the same number of layers but slightly worse than the residual network using 20 layers. The layer combining residuals and Spatial Batch Normalization is the best overall.

The 20 layers residual network was trained longer than the other networks. It used a 0.2 learning rate until 210,000,000 examples and then halved the learning rate every 35,000,000 examples until 340,000,000 examples (line 68 of the table). It reached a 58.001% accuracy on the test set. It is greater than previous results reaching either 57.0% with 128 planes Silver et al. (2016), or 57.3% with 512 planes Tian & Zhu (2015). The main difference with previous work is the architecture, we use less training examples as previous work since we only use KGS games without handicap by players greater than 6 dan.

In order to further enhance accuracy we used bagging. The input board is mirrored to its 8 possible symmetries and the same 20 layers network is run on all 8 boards. The outputs of the 8 networks are then summed. Bagging improves the accuracy up to 58.485%. The use of symmetries is similar to AlphaGo.

Table 1: Evolution of the accuracy on the test set

	net_{α}	net_{dark}	net_{13}	net_{20}	$net_{13/sbn}$
1	0.896	47.132	46.978	47.389	47.819
2	0.675	49.179	48.555	48.821	49.493
3	1.121	50.260	50.302	50.601	50.826
4	1.454	51.076	51.186	51.449	51.471
5	43.719	51.230	51.088	51.722	51.689
6	46.334	51.832	51.602	51.797	52.219
7	47.875	52.188	52.258	52.375	52.611
8	48.629	52.384	52.384	52.618	52.756
9	49.308	52.705	52.697	53.029	53.085
10	49.698	52.748	52.856	53.157	53.145
11	50.196	53.244	53.189	53.566	53.441
12	50.367	53.114	53.201	53.514	53.718
13	50.954	53.471	53.442	53.794	53.708
14	51.337	53.661	53.720	53.827	53.985
15	51.489	53.969	53.844	54.063	53.984
16	51.572	53.983	53.635	54.282	54.021
17	51.981	54.145	54.009	54.438	54.349
18	51.922	54.134	54.051	54.539	54.314
19	52.056	54.183	54.164	54.631	54.485
20	52.294	54.408	54.226	54.541	54.522
...				...	
68				58.001	

Encouraged by these results and in order to address the reviewers comments we ran additional tests. Instead of generating files for training as in the previous experiments we use dynamic minibatches of size 50 that take 50 random states in the training set, each randomly mirrored to one of its eight symmetric states. As preparing the minibatch can take a significant time compared to the training time, we optimized the ladder code and we removed the Monte Carlo related input features.

The updating of the learning rate is performed using algorithm 1. A step corresponds to 5000 training examples. Every 1000 steps the algorithm computes the average error over the last 1000 steps and the average error over the step minus 2000 and the step minus 1000. If the decrease in the average error is less than 0.05% then the learning rate is divided by 2. The initial learning rate is set to 0.2. The algorithm stays at least 4000 steps with the same learning rate before dividing it by 2.

Algorithm 1 The algorithm used to update the training rate

if nbExamples % 5000 == 0 **then**
  $step \leftarrow step + 1$
  if step % 1000 == 0 **then**
    $error1000 \leftarrow \sum_{last\ 1000\ steps}(errorStep)$
    $error2000 \leftarrow \sum_{step\ -\ 2000\ to\ step\ -\ 1000}(errorStep)$
    if step - lastDecrease $>$ 3000 **then**
      if error2000 - error1000 $<$ 0.0005 * error2000 **then**
        $rate \leftarrow \frac{rate}{2}$
        $lastDecrease \leftarrow step$
      end if
    end if
  end if
end if

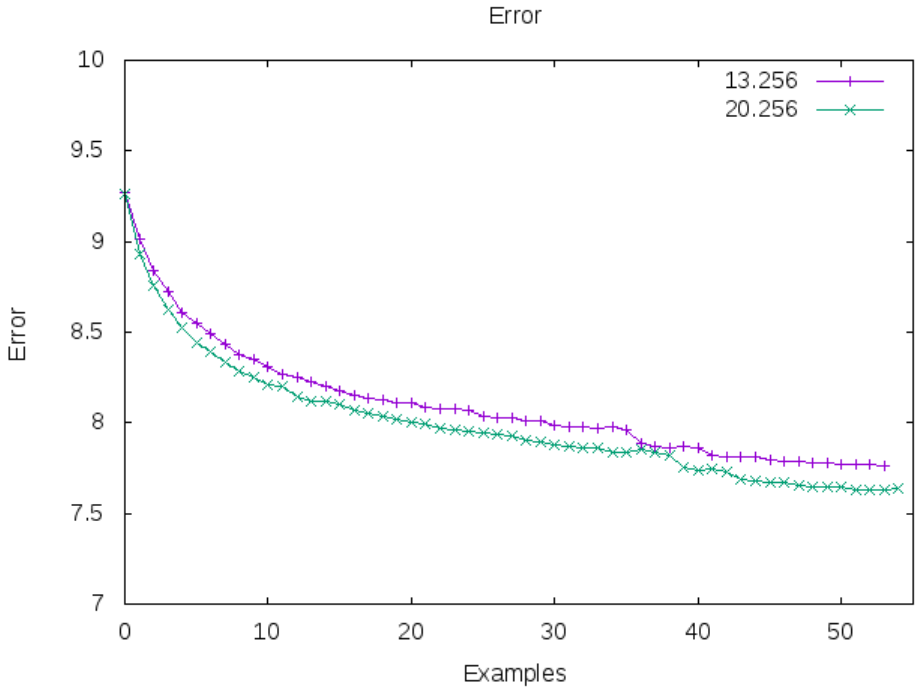

Figure 7: Evolution of the error on the test set.

The comparison of the evolution of the test error of the 13 layers network and of the 20 layers network is given in figure 7. These two networks use residual layers and spatial batch normalization. We observe that the 20 layers network is consistently better and ends with a significantly reduced error. This also results in a better accuracy as depicted in figure 8. The 20 layers network ends with an accuracy of 57.687% without bagging. The 13 layers networks ends with an accuracy of 57.024%. We also trained a 13 layers networks with 512 planes, it is slightly better than the 13 layers network with 256 planes but worse than the 20 layers network with 256 planes. In previous experiments networks with more than 13 layers were considered worse than 13 layers networks. Using our architecture enables to efficiently train networks deeper than 13 layers.

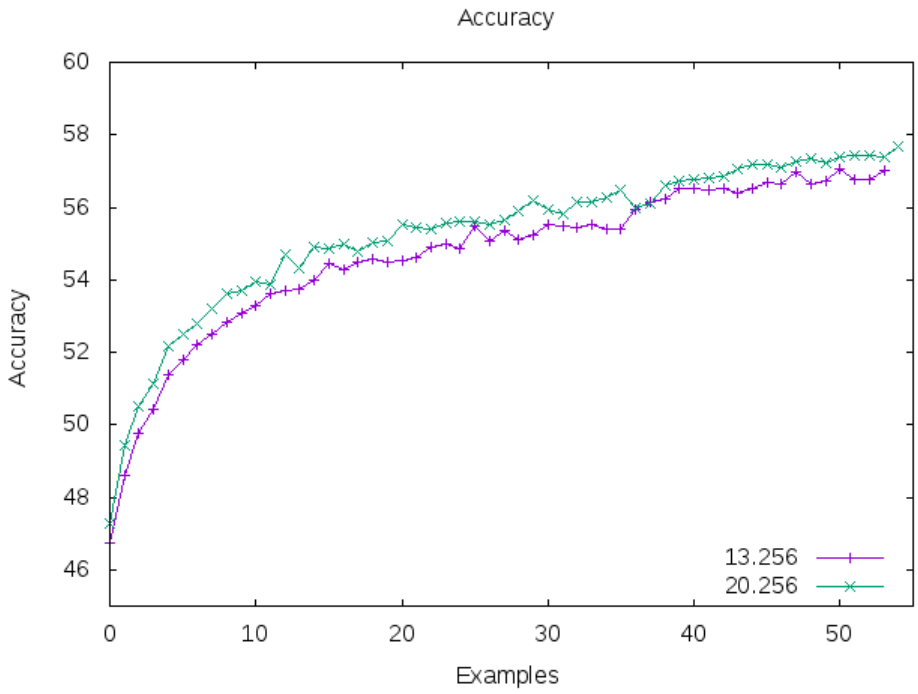

Figure 8: Evolution of the accuracy on the test set.

3.4 GOLOIS

We made the 20 layers network with bagging and a 58.485% accuracy play games on the KGS internet Go server. The program name is Golois3 and it is quite popular, playing 24 hours a day against various opponents. It is ranked 3 dan.

Playing on KGS is not easy for bots. Some players take advantage of the bot behaviors such as being deterministic, so we randomized play choosing randomly among moves that are evaluated greater than 0.95 times the evaluation of the best move and that have a greater evaluation than the best move when augmented by 0.05. Other players intentionally play the bot with a wrong handicap that disadvantages the bot if it loses and does not increase its rank if it wins. Golois plays on par with other 3 dan human players and can even occasionally win a game against a 5 dan. Golois3 plays its moves almost instantly thanks to its use of a K40 GPU. It gives five periods of 15 seconds per move to its human opponents.

In comparison, AlphaGo policy network and DarkForest policy network reached a 3 dan level using either reinforcement learning Silver et al. (2016) or multiple output planes giving the next moves to learn and 512 feature planes Tian & Zhu (2015).

Golois sometimes loses games due to the lack of tactical search. Especially against very strong players. In order to improve the level of play we plan to train a value network and to add the results of tactical searches as input features.

Two previous versions of Golois played before Golois3 on the KGS Go server. The first version is named Golois and it is described in Cazenave (2016b). Golois is ranked 1 kyu. The second version is named Golois2 and it is described in Cazenave (2016a). Golois2 is ranked 1 dan. The improvements due to changing the training set, the input features and the architecture described in this paper have enabled Golois3 to reach the 3 dan level.

The 20 layers network of the second set of experiments also plays on KGS under the name Golois4. It is ranked 3 dan.

4 CONCLUSION

The usual architecture of neural networks used in computer Go can be much improved. Adding Spatial Batch Normalization as in DarkForest enables to train the networks faster. Adapting residual networks also helps training the network faster. It also enables to successfully train deeper networks. A residual network with 20 layers scores 58.001% on the KGS test set. It is greater than previously reported accuracy. Using bagging of mirrored inputs it even reaches 58.485%. The 20 layers network with bagging plays online on KGS and reached a 3 dan level playing almost instantly. Combining residual networks with Spatial Batch Normalization enables to train networks faster and to efficiently train deeper networks.

Training deeper networks faster for better results is important for the next development phase of Golois, namely training a value network. We also plan to add the results of elaborate tactical searches as input to the network. Both for the policy and the value network.

ACKNOWLEDGMENTS

The author would like to thank Nvidia and Philippe Vandermersch for providing a K40 GPU that was used in some experiments. This work was also granted access to the HPC resources of MesoPSL financed by the Region Ile de France and the project Equip@Meso (reference ANR-10-EQPX-29-01) of the programme Investissements d'Avenir supervised by the Agence Nationale pour la Recherche.

REFERENCES

Tristan Cazenave. Residual networks for computer Go. *submitted to IEEE TCIAIG*, 2016a.

Tristan Cazenave. Combining tactical search and deep learning in the game of Go. In *IJCAI 2016 Workshop on Deep Learning for Artificial Intelligence (DLAI), NYC, USA*, 2016b.

Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play go. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1766–1774, 2015.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers (eds.), *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*, pp. 72–83. Springer, 2006.

Rémi Coulom. Criticality: a monte-carlo heuristic for go programs. *Invited talk at the University of Electro-Communications, Tokyo, Japan*, pp. 198–203, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015.

Chris J Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in go using deep convolutional neural networks. *arXiv preprint arXiv:1412.6564*, 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Yuandong Tian and Yan Zhu. Better computer go player with neural network and long-term prediction. *arXiv preprint arXiv:1511.06410*, 2015.