Combining tactical search and deep learning in the game of Go

Tristan Cazenave
PSL-Université Paris-Dauphine, LAMSADE CNRS UMR 7243, Paris, France
Tristan.Cazenave@dauphine.fr

Abstract

In this paper we experiment with a Deep Convolutional Neural Network for the game of Go. We show that even if it leads to strong play, it has weaknesses at tactical search. We propose to combine tactical search with Deep Learning to improve Golois, the resulting Go program. A related work is AlphaGo, it combines tactical search with Deep Learning giving as input to the network the results of ladders. We propose to extend this further to other kind of tactical search such as life and death search.

1 Introduction

Deep Learning has been recently used with a lot of success in multiple different artificial intelligence tasks. The range of applications go from image classification [Krizhevsky *et al.*, 2012] where deep convolutional neural networks have better results than specialized image vision algorithms with a more simple algorithm, to writing stories with recurrent neural networks [Roemmele, 2016].

Deep Learning for the game of Go with convolutional neural networks has been addressed first by Clark and Storkey [Clark and Storkey, 2015]. It has been further improved by using larger networks [Maddison *et al.*, 2014]. Learning multiple moves in a row instead of only one move has also been shown to improve the playing strength of Go playing programs that choose moves according to a deep neural network [Tian and Zhu, 2015].

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 [Silver *et al.*, 2016] 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 Monte Carlo Tree Search [Coulom, 2006]. It learns a value network with reinforcement learning to learn 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.

Elaborated search algorithms have been developed to solve tactical problems in the game of Go such as capture problems [Cazenave, 2003] or life and death problems [Kishimoto and Müller, 2005]. In this paper we propose to combine tactical search algorithms with deep learning.

Other recent works combine symbolic and deep learning approaches. For example in image surveillance systems [Maynord *et al.*, 2016] or in systems that combine reasoning with visual processing [Aditya *et al.*, 2015].

The next section presents our deep learning architecture. The third section presents tactical search in the game of Go. The fourth section details experimental results.

2 Deep Learning

In the design of our network we follow previous work [Maddison *et al.*, 2014; Tian and Zhu, 2015]. Our network is fully convolutional. It has twelve convolutional layers each followed by a rectified linear unit (ReLU) layer [Nair and Hinton, 2010] except for the last one. It uses eleven binary 19×19 input planes: three planes for the colors of the intersections, six planes for the liberties of the friend and of the enemy colors (1, 2, $\geq$ 3 liberties), two planes for the last moves of the friend and of the enemy: as in darkforest [Tian and Zhu, 2015] the value decreases exponentially with the recency of the last moves. The last move gets $e^{-0.1}$ for the first input plane, the penultimate move gets $e^{-0.2}$ for the second input plane, the move before that gets $e^{-0.3}$ for the first input plane and so on.

Each convolutional layer has 256 feature planes. The size of the filter for the first layer is 5×5 and the following layers use 3×3 filters. Figure 1 gives the architecture of the network.

On the contrary of previous work we have found that learning was faster when not using a softmax layer as a final layer. We also do not use a minibatch. We just use SGD on one example at a time.

We also believe that learning ko threats disturbs the learning algorithm. Ko threats are often moves that do not work in normal play, so in order to simplify learning we set them apart. In our learning and test sets we do not include moves from positions containing a ko.

Our training set consists of games played on the KGS Go server by players being 6d or more between 2000 and 2014. We exclude handicap games. Each position is rotated and mirrored to its eight possible symmetric positions. It results

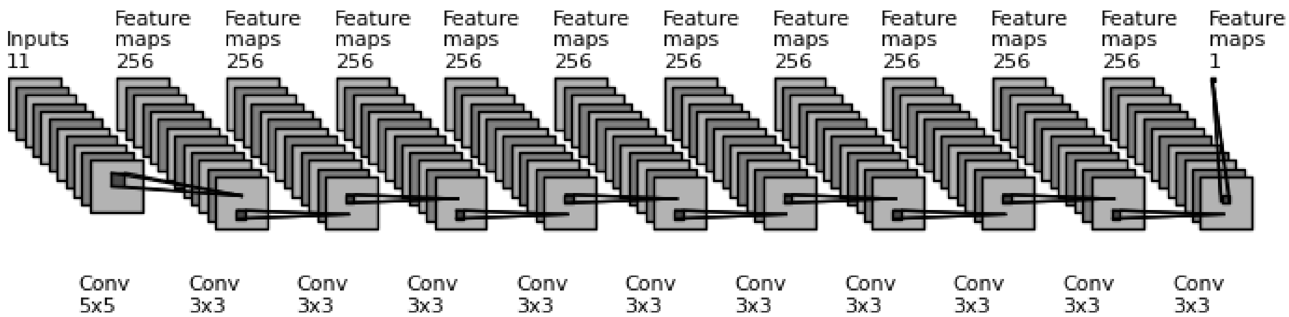

Figure 1: Architecture of the network.

in 160 000 000 positions in the learning set. 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.

3 Tactical Search

We made our DCNN play the game of Go on the KGS Go server. The following examples are taken from games played against human players. Let first define some important Go terms. A string is a set of stones of the same colors that are connected together. An important concept in the game of Go is the number of liberties of s string. The number of liberties is the number of empty intersections next to the stones. A particular kind of useful tactical search in the game of Go is ladders. A ladder is a consecutive serie of ataris that results in the capture of a string. In figure 2, Golois is Black and it fails to see a ladder that captures five black stones and makes White alive. The sequence of moves is obvious (W[C9], B[D8], W[B8], B[D7], W[D6], B[E7], W[F6]). However Golois fails to see it. These types of errors can cause Golois to lose a game it would otherwise win.

Another unlikely behavior of Golois is given in figure 3. We can see that it pushes through a lost ladder, giving Black some free points. We also want to prevent such bad behavior.

Besides from ladders, DCNN also have weaknesses for life and death problems. A string is alive if it cannot be captured. A string is dead if it cannot avoid being captured. Figure 4 shows a White move by Golois that fails to make an important group alive even though the living move is obvious. Such bad behavior could be avoided with a simple life and death search.

Other more complicated problems such as Seki are also out of scope of the DCNN as can be seen in figure 5. A move at J1 would have given White life by Seki, and it could easily have been found with a life and death search algorithm.

Another kind of life and death problems that are difficult to handle even with Monte Carlo Tree Search are problems involving double kos. In the last November 2015 Mylin Valley computer Go tournament, Dolbaram the winner of the tournament failed to understand a life and death fight involving a double ko when playing an exhibition match against a strong

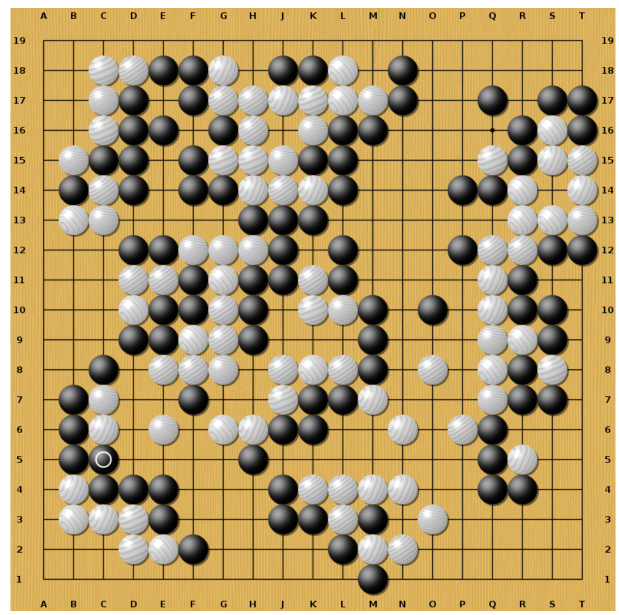

Figure 2: An unseen ladder.

professional player. This kind of problem can be solved by a life and death search algorithm.

The life and death problem is not specific to Golois. For example Darkforest, the Facebook AI Research bot also played a lot of games on KGS with a deep neural network. In many games it lost the game due to the inability to handle well a simple life and death problem. Other deep learning Go programs could be improved by incorporating simple life and death knowledge and search.

The ladder algorithms we use are given in algorithms 1 and 2.

The captureLadder algorithm searches for the capturing moves. The *inter* variable is the intersection of a stone of the string to capture. The *depth* variable is the depth of the search, it is initially called with a zero value. If the string of the stone is captured the algorithm sends back true as it suc-

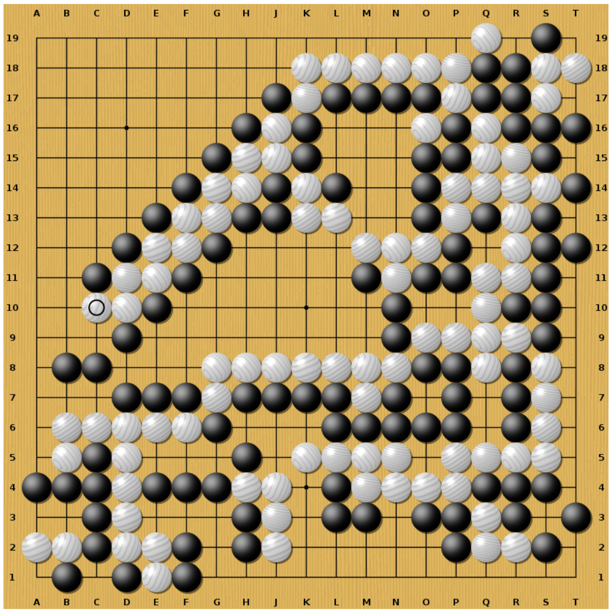

Figure 3: A lost ladder.

ceeded. if the string gets more than two liberties, then it is considered saved. The algorithm could be extended to handle more complex captures of strings that have more than two liberties by modifying this threshold.

The isCapturedLadder algorithm verifies that all possible moves that can save a string do not work and that the string is captured. It is called by the captureLadder algorithm and it also recursively calls the captureLadder algorithm.

The way we integrate ladders with the neural network is that we modify the result of the output of the network according to ladders. If a move is in a ladder and results in more than four stones, its value is decreased by the number of stones. If a move captures strictly more than four stones in a ladder, its value is increased by the number of captured stones. If a move saves strictly more than four stones in a ladder, its value is increased by the number of saved stones. Using these simple rules occasionally wastes a move as it is not always the best move to play in a ladder even if it has move than four stones. However it often saves a game when the DCNN fails to see a ladder.

4 Experimental Results

Tables 1 and table 2 give the learning curve of the DCNN. The last column gives the percentage of move prediction on the test set. These moves are the ones played by players ranked better than 6d on KGS, so these are the kind of moves we want the network to replicate. As we use SGD with no minibatch we could use high learning rates such as 0.2 to start with. Then in the end the network was fine tune with a 0.025 learning rate. We get 55.56% on the test set which is comparable to other approaches. AlphaGo gets 57.0% on the test set for its policy network. When all the examples in the training set have been used, the learning algorithm starts again from the first examples.

Algorithm 1 The capture ladder algorithm.

```
captureLadder (inter, depth)
if depth > 100 then
  return false
end if
nbNodesLadder++
if nbNodesLadder > 1000 then
  return false
end if
if board [inter] == Empty then
  return true
end if
n = nbLiberties (inter, liberties, stones)
if n > 2 then
  return false
end if
if n == 1 then
  if capture on liberty is legal then
    return true
  end if
end if
res = false
if n == 2 then
  for m in liberties do
    if m is legal then
      play (m)
      if isCapturedLadder (inter, depth + 1) then
        res = true
      end if
      undo (m)
      if res == true then
        return true
      end if
    end if
  end for
end if
return res
```

Algorithm 2 The captured ladder algorithm.

```
isCapturedLadder (inter, depth)
if depth > 100 then
  return false
end if
nbNodesLadder++
if nbNodesLadder > 1000 then
  return false
end if
if board [inter] == Empty then
  return true
end if
n = nbLiberties (inter, liberties, stones)
if n == 0 then
  return true
end if
if n > 1 then
  return false
end if
res = true
if n == 1 then
  for m in strings adjacent to inter do
    if the adjacent string has one liberty then
      if the liberty is a legal move then
        play (liberty)
        if not captureLadder (inter, depth + 1) then
          res = false
        end if
        undo (liberty)
      end if
    end if
  end for
  for m in liberties do
    if m is legal then
      play (m)
      if not captureLadder (inter, depth + 1) then
        res = false
      end if
      undo (m)
    end if
  end for
end if
return res
```

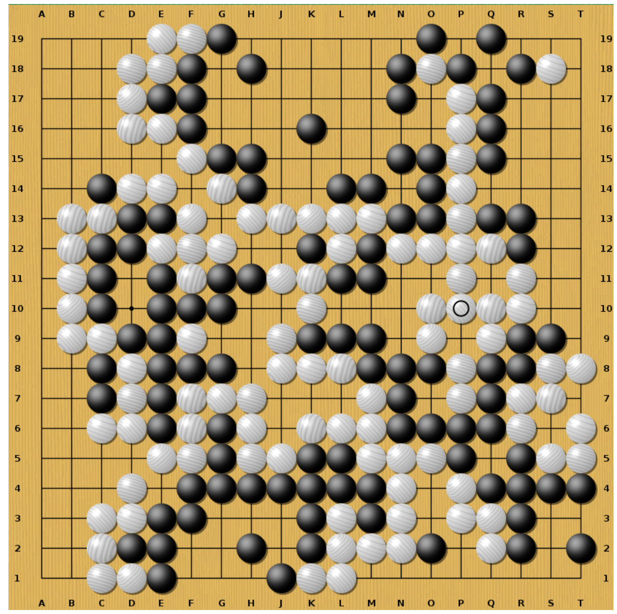

Figure 4: Missing the living move.

A simple improvement that improves the prediction accuracy is to use bagging with the same network applied to the eight possible symmetries of a Go board. For each move, the value of a move is the sum of the values of the symmetric move on the reflected boards. Bagging enables to improve the prediction accuracy from 55.809% to 56.398% for four symmetries, and to 56.513% for eight symmetries.

Golois played a lot of games on the KGS Go server. Its level of play is currently first kyu. It occasionally wins games against 2d and loses some games to 2k but rarely to players less than 2k. Games against other first kyu are balanced. Reaching the first kyu level for a deep network which is not combined with Monte Carlo Tree Search is a nice achievement and it competes well with the other best programs using a similar architecture. Moreover it plays moves very fast for a first kyu program.

5 Conclusion

We have presented a combination of tactical search and deep learning for the game of Go. Deep convolutional neural networks have difficulties at tactical search. Combining them with specialized tactical searches such as capture search or life and death search improves their level of play.

The combination with tactical search results in improved policy network that can also be used for programs that combine Monte Carlo Tree Search and Deep Learning.

Future work will be to use more elaborate tactical search algorithms for capture and life and death.

Our current use of the results of tactical searches is rather crude since it consists in always following the tactical move if it is considered important by an heuristic. In future work we will use the results of tactical search on more complex capture and life and death as inputs to the neural network.

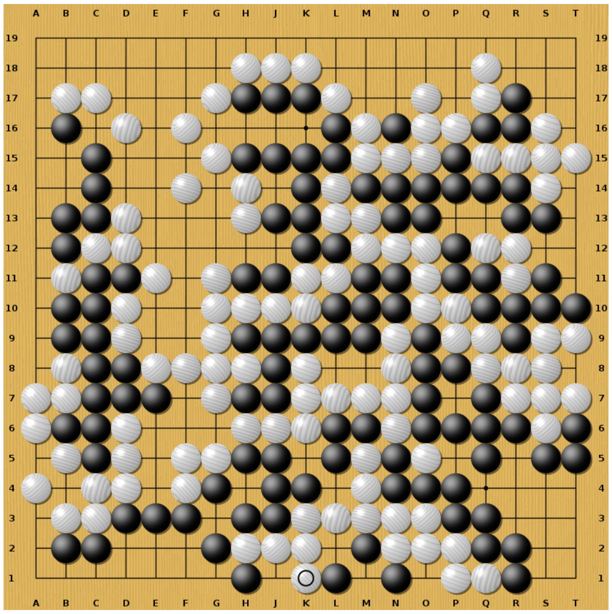

Figure 5: Missing the seki move.

Acknowledgments

This work was 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

[Aditya *et al.*, 2015] S. Aditya, Y. Yang, C. Baral, C. Fermuller, and Y. Aloimonos. Visual common-sense for scene understanding using perception, semantic parsing and reasoning. In *L. Morgenstern, T. Patkos, and R. Sloan (Eds.) Logical Formalizations of Commonsense Reasoning (Technical Report SS-15-04). Stanford, CA: AAAI Press*, 2015.

[Cazenave, 2003] Tristan Cazenave. A generalized threats search algorithm. In *Computers and Games*, volume 2883 of *Lecture Notes in Computer Science*, pages 75–87. Springer, 2003.

[Clark and Storkey, 2015] 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)*, pages 1766–1774, 2015.

[Coulom, 2006] 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, editors, *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2006.

[Kishimoto and Müller, 2005] Akihiro Kishimoto and Martin Müller. Search versus knowledge for solving life and death problems in go. In *AAAI*, pages 1374–1379, 2005.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[Maddison *et al.*, 2014] 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.

[Maynord *et al.*, 2016] M. Maynord, S. Bhattacharya, and D. W. Aha. Image surveillance assistant. In *Computer Vision Applications in Surveillance and Transportation: Papers from the WACV-16 Workshop. Lake Placid, NY*, 2016.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[Roemmele, 2016] Melissa Roemmele. Writing stories with help from recurrent neural networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 4311–4342, 2016.

[Silver *et al.*, 2016] 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.

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

Table 1: Evolution of the score on the test set with learning.

Examples learned	Learning rate	Test set percentage
5 000 000	0.2	44.033
10 000 000	0.2	46.548
15 000 000	0.2	48.2
20 000 000	0.2	48.851
25 000 000	0.2	49.084
30 000 000	0.2	49.595
35 000 000	0.2	50.042
40 000 000	0.2	50.457
45 000 000	0.2	50.734
50 000 000	0.2	50.994
55 000 000	0.2	51.183
60 000 000	0.2	51.34
65 000 000	0.2	51.59
70 000 000	0.2	51.817
75 000 000	0.2	52.05
80 000 000	0.2	52.098
85 000 000	0.2	52.218
90 000 000	0.2	52.407
95 000 000	0.2	52.762
100 000 000	0.2	52.807
105 000 000	0.2	52.516
110 000 000	0.2	52.919
115 000 000	0.2	53.278
120 000 000	0.2	53.076
125 000 000	0.2	53.182
130 000 000	0.1	53.673
135 000 000	0.1	53.834
140 000 000	0.1	53.918
145 000 000	0.1	54.114
150 000 000	0.1	54.41
155 000 000	0.1	54.636
160 000 000	0.1	54.664
165 000 000	0.1	54.748
170 000 000	0.1	54.838
175 000 000	0.1	55.062
180 000 000	0.05	55.037
185 000 000	0.05	54.85
190 000 000	0.05	55.036
195 000 000	0.05	55.56

Table 2: Evolution of the score on the test set with learning.

Examples learned	Learning rate	Test set percentage
200 000 000	0.025	55.228
205 000 000	0.025	55.059
210 000 000	0.025	55.155
215 000 000	0.025	55.15
220 000 000	0.025	55.177
225 000 000	0.025	55.159
230 000 000	0.025	55.21
235 000 000	0.025	55.276
240 000 000	0.025	55.285
245 000 000	0.025	55.283
250 000 000	0.025	55.282
255 000 000	0.025	55.17
260 000 000	0.025	55.149
265 000 000	0.025	55.139
270 000 000	0.025	55.217
275 000 000	0.025	55.187
280 000 000	0.025	55.12
285 000 000	0.025	55.282
290 000 000	0.025	55.549
295 000 000	0.025	55.449
300 000 000	0.025	55.579
305 000 000	0.025	55.532
310 000 000	0.025	55.749
315 000 000	0.025	55.692
320 000 000	0.025	55.784
325 000 000	0.025	55.809