Mastering the game of Go from scratch

Michael Painter, Luke Johnston

Stanford University

Introduction

In the landmark paper "Mastering the game of Go with deep neural networks and tree search" [1], superhuman performance on the game of Go was achieved with a combination of supervised learning (from professional Go games) and reinforcement learning (from self-play). However, traditional pure RL approaches haven't yielded satisfactory results on the game of Go on a 19x19 board because its state space is so large and its optimal value function so complex that learning from self-play is infeasible. However, these limitations do not apply to smaller board sizes. In this project, we investigate an entirely unsupervised, reinforcement-learning based approach to playing Go, by learning how to play on a smaller board and using a form of transfer learning to learn to play on larger board sizes. If successful, this suggests an effective strategy using RL for approaching large state space problems, for which we, as humans, are not experts (and cannot generate any good dataset).

Reinforcement Learning Details

A policy $p_\rho(s)$ parameterised by ρ, is learned, by optimising over the expected reward:

$$U(p_\rho) = \sum_\tau \Pr(\tau|\rho) R(\tau)$$

where τ is a path of states. By the policy gradient theorem [2] and using an empirical estimate, we can arrive at an update rule for ρ of

$$\Delta\rho = \frac{\alpha}{n} \sum_{i=1}^{n} \sum_{t=1}^{T_i} \frac{\partial \log(p_\rho(s_t^i, a_t^i))}{\partial \rho} v_t^i$$

where for each episode i, s_t, a_t and v_t are the t'th state, action and value.

To train the network we play games against an old opponents. Where we play against policy ρ^-, chosen uniformly from the pool of old opponents. The current policy is copied every 500 episodes into the opponent pool.

To evaluate the policy learned we play against the Pachi AI [4], provided by OpenAI's Go Environment [5]. We track two metrics, average game length (to see if the game was 'competitive') and the average value of the game from our agent's perspective (the win ratio).

A number of different design choices were made compared to Alpha Go:

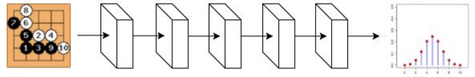
- Removal of variance reduction term from update rule, as it would significantly increase the computation time.
- No use of Monte Carlo Tree Search (MCTS) in training, allowing for the self play to run some orders of magnitudes quicker.
- We still learn a policy (softmax output) rather than a Q function, as many actions could have a large Q-values, leading to a large branching factor when run with MCTS (for competitive play).

5x5 Board Architecture

The architecture used for the 5x5 board, the 'base case', is a 5 layer convolutional neural network. Padding is used so that each of the intermediate layers maintains the same shape of the input.

- The first layer uses a 5x5 convolution, with _ filters and __ activation.
- The second to fourth layers use a 3x3 convolution, with _ filters and a ReLU activation.
- The final, fifth layer uses a 1x1 convolution, with 1 filter, and a softmax activation to provide a probability distribution as output.
- Each layer includes bias terms.

To encode the input, we use 3 channels, one for the white pieces, one for the black pieces and one for free spaces.



Example of Transfer Learning to 9x9 Board

Black box:

- A network trained to play on a 5x5 board is used as a black box (weights frozen),
- The black box is used like an oracle to query about 'global' information and 'local' information.

Transfer method 1 - convolution, or 'local' use:

- The 5x5 network is applied to every possible window of the 9x9 board (as if the 5x5 network were a convolutional filter).
- Each application yields a 5x5 output of 'policy values' for actions at each board location:
  - Which is padded with zeros to match original 9x9 board;
  - All outputs concatenated to form 9x9x25 tensor.

Transfer method 2 - scaling, or 'global' use:

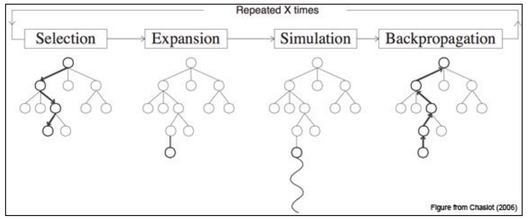
- A convolution is applied to the 9x9 board, such that the output is 5x5, by using a 5x5 convolution, and no padding
- The convolutions output is run through the black box
- The transpose of the *same* filter is applied to the 5x5 output from the black box, to mimic an 'inverse', and yielding a 9x9 output.

MCTS

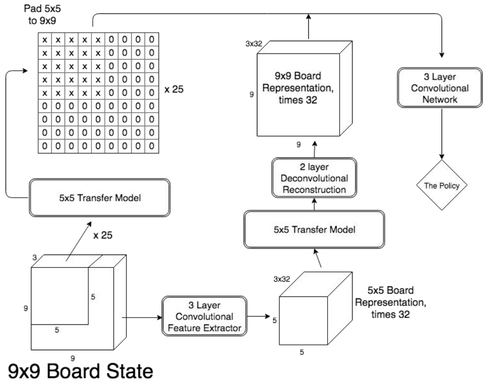
For competitive play we use the Monte Carlo Tree Search (MCTS) algorithm, which uses a stochastic policy to guide a tree search for a zero sum game. The algorithm maintains a partial search tree, adding one node to it on every iteration. Internally, every node maintains a 'value'.

- Selection: traverse the partial tree (probabilistically, according to the search policy), until a node not in the tree is reached.
- Expansion: add the node to the tree.
- Simulation: use a policy to (greedily) pick actions, until a terminal state. The value of this run of the game is then known.
- Backpropagation: update the value of all nodes traversed in this iteration using the value obtained from the simulation.

After some fixed number of iterations or time, the MCTS agent selects the action the leads to the child with maximum value from the root of the tree.



Composite Network Architecture

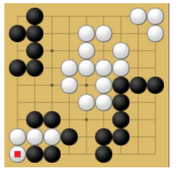


Gameplay

```
To play: black
Move:  12  Komi: 0.0
    A B C D E
  +-----------+
5 | . . . . . |
4 | X X X X . |
3 | O O O X . |
2 | . X O)O O |
1 | . . . . . |
  +-----------+
```

```
To play: black
Move:  20  Komi: 0.0
    A B C D E
  +-----------+
5 | . X . X . |
4 | X X X X O)|
3 | O O O X O |
2 | O X O O O |
1 | . X O X . |
  +-----------+
```

- 5x5 board: learns to contest middle 3x3 squares first
  - Does not learn optimal first move (center square)
- Forms an "eye" structure on top border of board
- Makes a couple obvious mistakes - 2B in first image, 1D in second image



- Above diagram illustrates a couple Go concepts

References

[1] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

[2] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." NIPS. Vol. 99. 1999.

[3] Greensmith, Evan, Peter L. Bartlett, and Jonathan Baxter. "Variance reduction techniques for gradient estimates in reinforcement learning." Journal of Machine Learning Research 5.Nov (2004): 1471-1530.

[4] Baudiš, Petr, and Jean-loup Gailly. "Pachi: State of the art open source Go program." Advances in computer games. Springer Berlin Heidelberg, 2011.

[5] Brockman, Greg, et al. "OpenAI gym." arXiv preprint arXiv:1606.01540 (2016).