Playing Draughts using Neural Networks and Genetic Algorithms

Thien Nguyen

Department of Computer Science
Durham University

January 29, 2018

Problem Description

A problem in Computer Science (..before AlphaZero existed)

Presently, A variety of Draughts AI players tend to be designed to play at a fixed ability.

While it has produced very competitive and intelligent players, they require human intervention in order to improve their performance.

By combining Neural Networks and Genetic Algorithms, this issue could possibly be solved by creating a player that can grow in ability over time, without the dependency on move-banks.

Related Work

Similar works of art but no cigar

Samuel (59')

Uses Genetic Algorithms to improve coefficents of a set of heuristics to evaluate Draughts games. [3]

Blondie24 (97')

Uses an Evolutionary Algorithm and Neural Networks to evaluate Draughts games. (Quite similar!) [1]

Giraffe (15')

Uses contemporary machine learning techniques to train a Neural Network to evaluate Chess games. [2]

Uber (Last month)

Uses genetic algorithms to train convolutional neural networks to play Atari Games. [4]

Current Approach

How will I tackle this; in a nutshell

1. Evaluate a checkerboard state
2. Choose the best move for a given state
3. Make our checkerboard evaluator better

Evaluating Checkerboards

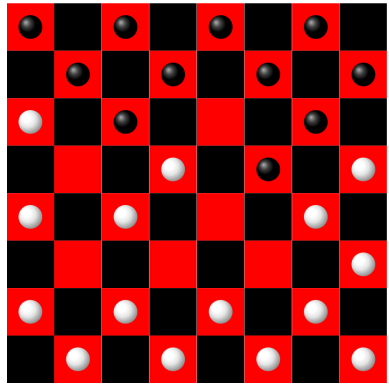
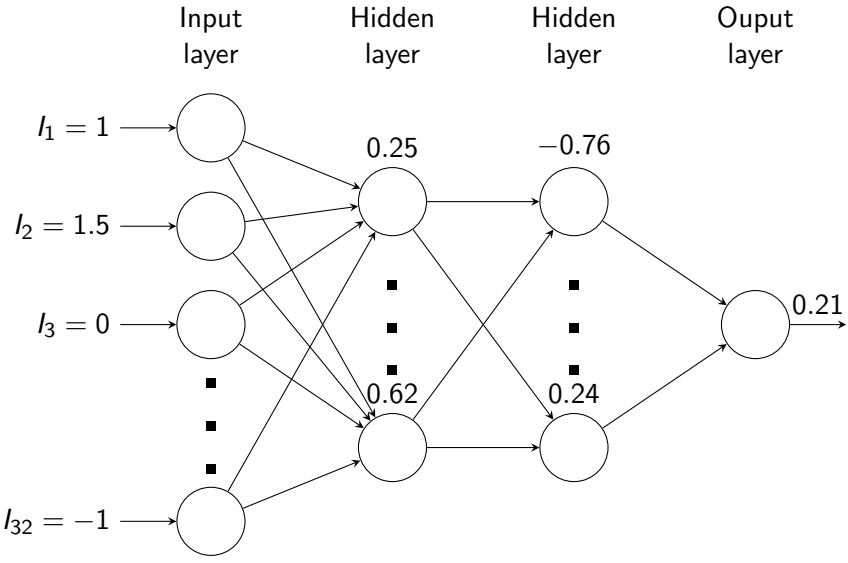


Figure: An example state of a checkerboard.

Neural Networks



Activation Function

Neural networks use activation functions to simulate an output of a node given a set of input(s).

$$O = f((Input * weight) + Bias)$$

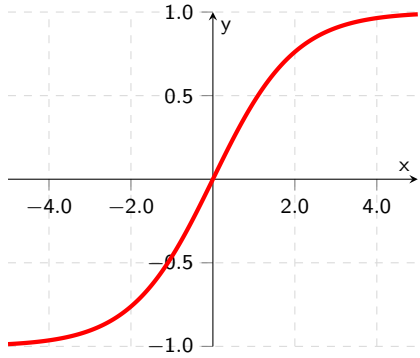
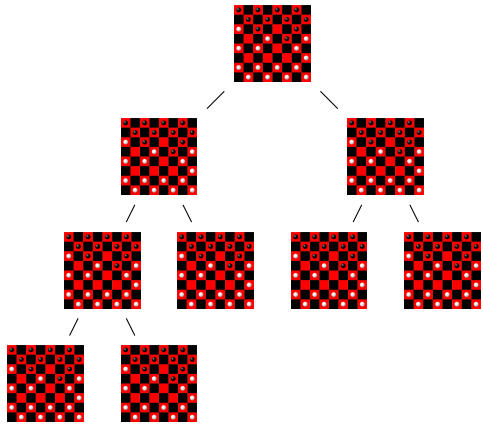


Figure: graph of tanh function
$f(x) = \frac{2}{1+e^{-x}} - 1$

Choosing moves



How do we choose a good move?

Monte Carlo Tree Search

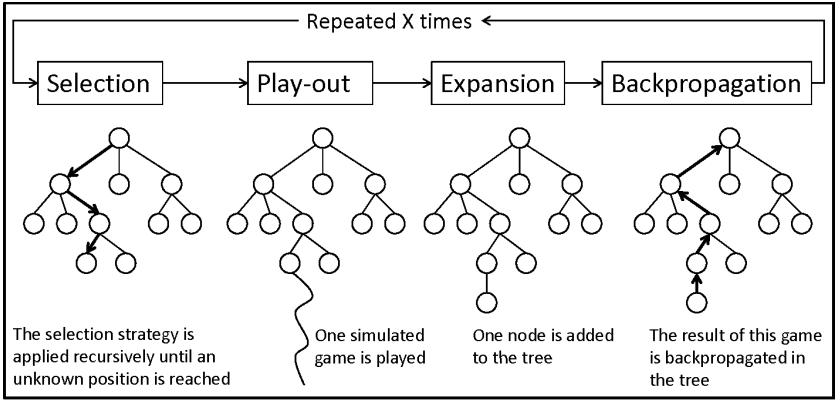


Figure: Diagram describing the MCTS process. Source: Winands et al. ([5])

Generating Agents

Agent

An agent is a generated set of weights and biases for a neural network.

- An initial agent would have a randomly generated set of weights and biases
- initial values range from -0.2 to 0.2.

Tournament

1. Generate a population of random agents (A population of 15 is used)
2. Make agents play each other
3. Order agents by the amount of points scored
4. The best few agents are chosen to stay on for the next tournament
5. Make new agents from them
6. The losers are destroyed
7. Repeat step 2-6 with the new agents until satisfied

Crossover Mechanism

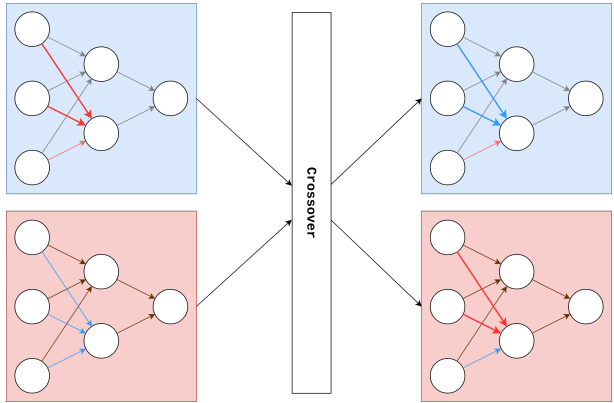


Figure: Offspring agents are created from a pair of parent agents. Each offspring is a reciprocal crossover of each other. A random hidden node is chosen, and influential inputs for that node are swapped between the two agents.

Mutation

WeightP is the current weight, K represents the number of weights and biases in the neural network, and m represents a random floating point in the range of [-1,1]:

$$WeightN = WeightP + \frac{m}{\sqrt{2 * \sqrt{K}}}$$

In reality, every weight and bias will be manipulated by some small manner.

Evaluation

How will I judge my outcome?

- **Play final agent against older agents to indicate whether it learns over time**
- Play against a randomly playing bot to see it is actually thinking
- Play against different move making algorithms (Vanilla MCTS, $\alpha\beta$ MiniMax)
- Also play against human players on popular checkers websites

Current Progress

What have I done already?

- The initial set up is ready*
- It has been trained*
- It plays relatively well (anecdotally)

Remaining Work

What do I still need to do?

- * Squash potential bugs that could impede performance/provide false positives
- Experiment with different crossover and mutation mechanisms
- Optimise training (enforce threefold repetition)
- Train the system for a bit longer
- Measure the system's performance against other players

Conclusion

TL;DR

- A Neural Network will evaluate a given state of the game
- Monte Carlo Tree Search chooses the best moves for a given state
- Genetic algorithms will train the Neural Network
- The AI is tested against agents from earlier generations to see whether it has learnt over time

Fin.

Any Questions?

References

Nanos gigantum humeris insidentes

Kumar Chellapilla and David Fogel.
Evolving Neural Networks to Play Checkers without Expert Knowledge.
IEEE Transactions on Neural Networks, 10(6):1382–1391, 1999.

Matthew Lai.
Giraffe: Using Deep Reinforcement Learning to Play Chess.
(September), 2015.

A L Samuel.
Some studies in machine learning using the game of checkers.
IBM Journal of Research and Development, 44(1.2):206–226, 2000.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune.
Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning.
arXiv:1712.06567 [cs], December 2017.
arXiv: 1712.06567.

M. H. M. Winands, Y. Bjornsson, and J. T. Saito.
Monte Carlo Tree Search in Lines of Action.
IEEE Transactions on Computational Intelligence and AI in Games, 2(4):239–250, December 2010.