Giraffe!

Ayoub Mokhtari, M. Nassim Aliousalah, Mohammed D. Belgoumri, Oussama Dje

December 29, 2021

Contents

Chapter 1

Introduction

Chess has been an extensively-studied problem in artificial intelligence since the 1950s. In a chess problem, we basically try to find a move that maximizes our chance of winning. In high level chess, we usually model the opponent to be the same as ourselves. An alternative formulation of the problem more often used is : given a chess position, we want to assign a score to it that corresponds to the chance of winning for the side to move, and for any given position, the score for one side is the negated score for the opposite side, because in chess scores are centered around 0.

Chapter 2

Conventional Chess Engines

2.1 Minimax

The minimax algorithm is a simple recursive algorithm to score a position based on the assumption that the opponent thinks like we do, and also wants to win the game.In chess, this algorithm works in theory but not in practice, the most common approach is a fixed-depth search, we go down in the tree and when we are at the end of the sub-tree we want to search, we call a static evaluation function that assigns a score to the position. The method generates a new problem, the horizon effect,the solution to it is called “quiescent search” (q-searche).

2.2 alpha-beta pruning

This algorithm can be optimized by introducing a lowerbound/uperbound for each call to minimax(), we call this : the alpha-beta pruning. In alpha-beta pruning we only explore nodes that can potentially be ”useful”, we can stop searching a node as soon as we prove that the result will be outside the window.In the best case scenario, it reduces the effective branching factor to the square root of the original value, which would allow the search to go about twice as far in the same amount of time. In conventional chess engines, there are a few common heuristics to improve move ordering :

1. If we had previously searched the same position, the move that ended up being the best is probably still the best (even if the previous search was done to a shallower depth).

2. If a move proved to be good in a sibling node, there is a good chance it will be good for the node under consideration as well.

3. Capturing high-valued pieces with low-valued pieces is usually good, and capturing defended low-valued pieces with high-valued pieces is usually bad.

4. Queen promotions are usually good.

2.3 Evaluation

The job of the evaluation functions is to assign scores to positions statically. For instance, we can take a look at the evaluation function of Stockfish, which consists of 9 parts: Material, Piece-Square Tables, Pawn Structure, Piece-specific Evaluation, Mobility, King Safety, Threat, Space and Draw-ish-ness.

Chapter 3

Machine Learning in Zero-Sum Games

3.1 Mate Solvers

There have been many attempts at using machine learning to solve chess endgames. We note that these situations are much simpler than average chess positions. Approaches using genetic programming do not scale to much more complicated positions seen in midgames and openings.

3.2 Learning Based on Hand-Selected Features

In a typical chess evaluation function, there are hundreds of parameters that must be tuned together. NeuroChess and The Falcon project are two attempts to tune evaluation parameters using generic programming with the help of a mentor, and a more recent attempt is the Meep project by Veness et al, whose evaluation function is a linear combination of hand-designed features, and where the weights are trained using reinforcement learning.

3.3 Temporal-Difference Learning

Temporal-difference reinforcement learning is an approach to evaluate the performance of models, where systems are tested to predict their own evaluation in the near future. This approach has several advantages over using game results, the most important one being that evaluations used to make moves in the same game can receive different error signals. In temporal-difference learning, only the blunder (which creates a temporal inconsistency) will get a high error signal. This

significantly improves the signal-to-noise ratio in error signals.The most famous study in using temporal-difference reinforcement learning to play zero-sum games is probably Tesauro's backgammon program TD-Gammon [19] from 1995.

3.4 Deep Learning

To go beyond weight tuning with hand-designed features we need a powerful and highly non-linear universal function approximator, where the input-output mapping is inherently hierarchical, the models would have to be made extremely large. In this project, we apply deep learning to chess. We use deep networks to evaluate positions, decide which branches to search, and order moves.