

Propuesta de Tesis de Licenciatura

Departamento de Computación - Facultad de Ciencias Exactas y
Naturales - Universidad de Buenos Aires

Título tentativo: Algo de ajedrez (?) DECIDIR

Alumno: Martín Emiliano Lombardo

LU: 49/20

Email alumno: mlombardo9@gmail.com

Plazo estipulado acordado con el tesista: 6 meses

Director: Agustín Sansone

Email director: agustinsansone7@gmail.com

Introducción

El desarrollo de engines de ajedrez es y ha sido un tema de interés en la comunidad de ajedrez y computación desde hace décadas. IBM DeepBlue [3] fue la primera máquina de ajedrez en ganarle a un campeón mundial —Garry Kasparov— en 1997. A partir de entonces, los engines han evolucionado en fuerza y complejidad.

Los engines tienen dos componentes principales: la búsqueda y la evaluación. La búsqueda es el proceso de explorar el árbol de posibles jugadas. La evaluación determina qué tan buena son esas posiciones para el que juega. Desde el origen de ajedrez por computadora en los años 50 hasta hace unos años, todos los engines han utilizado los algoritmos de búsqueda en árboles Minimax, Monte Carlo Tree Search [2] (MCTS) o alguna de sus variantes, con funciones de evaluación muy complejas y artesanales que se basan en conocimiento humano sobre el juego.

Hasta los 2010s, el desarrollo de engines avanzaba a un paso lento pero consistente hasta que en 2017, Google DeepMind publicó AlphaGo Zero [7] y su sucesor AlphaZero [6, 5] (2018), que mostró ser contundentemente superior (28 victorias y 73 empates contra el mejor engine del momento). Introdujeron un nuevo enfoque para el desarrollo de engines de juegos de tablero: entrenar una red neuronal convolucional con un algoritmo de aprendizaje por refuerzo para que aprenda a jugar por sí misma.

Este cambio de paradigma en donde la evaluación de las posiciones se realiza mediante redes neuronales en vez de funciones construídas con conocimiento humano, alteró el rumbo del desarrollo de todos los engines modernos. En 2018, Yu Nasu introdujo las redes neuronales “Efficiently Updatable Neural-Networks” [4] (NNUE) para el juego Shogi. Las redes NNUE permiten evaluar posiciones similares con menos cómputo que si se lo hiciera de forma completa, lo que las hace ideales para ser utilizadas en engines con búsqueda de árbol. A partir de entonces, todos los engines modernos han incorporado redes NNUE o alguna especie de red neuronal a su evaluación.

El motor de ajedrez Stockfish, uno de los más fuertes del mundo, ha incorporado redes NNUE mezclado con evaluación clásica en la versión 12¹ (2020). A partir de Stockfish 16.1² (2024) la evaluación se realiza exclusivamente mediante redes NNUE, eliminando todo el aspecto humano.

¹Introducing NNUE evaluation (Stockfish 12)

²Removal of handcrafted evaluation (Stockfish 16.1)

Objetivo

El objetivo de la tesis es experimentar con diferentes **feature sets** en **redes neuronales NNUE** para un engine con optimizaciones clásicas de ajedrez.

NNUEs

Las NNUEs son redes neuronales utilizadas para evaluar las posiciones en los nodos hoja de las búsquedas de los engines. Estas redes tienen la particularidad de que su arquitectura permite evaluar posiciones similares con menos cómputo que si se lo hiciera de forma completa. Al explorar el árbol de búsqueda, el estado de la primera capa de la red se puede actualizar de forma eficiente, amortiguando el cómputo de la primera capa casi en su totalidad (que aprovechamos que sea la más densa y cara).

Además, estas redes se cuantizan a 8 bits y se implementan mediante operaciones SIMD, haciendo el cómputo mucho más eficiente.

Feature sets

Un feature set es un conjunto de características que podemos extraer de una posición, como la ubicación, el color y rol de las piezas. El objetivo es experimentar con diversos sets, teniendo de referencia los existentes y proponer otros nuevos.

Por ejemplo, podemos definir el feature set natural HALF-PIECE (obviar el HALF, tiene que ver con la arquitectura de la red) como $\langle piece_square, piece_role, piece_color \rangle$, donde *piece_square* es la ubicación de la pieza en el tablero, *piece_role* es el tipo de pieza (peon, torre, etc) y *piece_color* es el color de la pieza. Cada tupla tiene asociada un índice en el vector de entrada de la red, que se setea a 1 si el feature está activo y 0 si no. Como tenemos 64 casillas, 6 tipos de piezas y 2 colores, hay $64 * 6 * 2 = 768$ features en este feature set. A modo de referencia, el feature set actual de Stockfish, HALFKAV2_HM tiene 22.528 features.

dos técnicas para entrenar las redes: la que se utiliza en el estado del arte y una técnica descrita en un blog, descritas en la sección de metodología.

Actividades y metodología

Engine

Se implementa un engine de ajedrez con heurísticas y mejoras clásicas, usando una evaluación con NNUEs (cuantizadas y en SIMD). Se utilizará para medir la performance de los modelos: jugar partidas entre ellos (elo) y resolver puzzles. La implementación será negamax, con poda alfa-beta incluyendo las heurísticas: ordenamiento de movimientos (MVVA, killer/history), búsqueda quiescente, null-move y tabla de transposiciones.

Dataset

Se utilizará el dataset de Lichess, que cuenta con 5.5B de partidas públicas jugadas

en Lichess, equivalentes a 1.71TB de PGNs comprimidos. En el dataset hay más de 200B de posiciones, pero dado la cantidad de partidas, voy a considerar solo una posición por partida, para mejorar la diversidad. Además se utilizará el dataset de puzzles de Lichess para evaluar la performance.

Entrenamiento

Para entrenar los modelos, se prueban 2 alternativas:

“Eval”: Se toman posiciones aleatorias del dataset de partidas y se utiliza Stockfish a profundidad fija como oráculo para obtener una evaluación, generando un nuevo dataset. Luego se entrena el modelo usando estos puntajes como target. Esto es lo mismo que hace Stockfish y debería ser lo mejor.

[1]

“ PQR ”: No se usa ningún oráculo, se genera un nuevo dataset de triplas (P, Q, R) . Se toma una posición P aleatoria en una partida. Luego se toma la posición observada como Q (es decir la posición siguiente en la partida, la que se jugó, $P \rightarrow Q$). Finalmente se toma una posición R aleatoria tal que $P \rightarrow R$ y $R \neq Q$. Suponiendo que f es el modelo, la premisa de esta técnica es que los jugadores eligen movimientos que son buenos para ellos, pero malos para los otros, entonces $f(P) = -f(Q)$. Por la misma razón, ir de P a R (es decir no Q) una posición aleatoria, se espera que $f(R) > f(Q)$, porque el movimiento aleatorio es mejor para el jugador siguiente y peor para el que hizo el movimiento. Se utiliza una función de pérdida con esas inecuaciones.

Factibilidad

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Referencias

- [1] Erik Bernhardsson. *Deep learning for... chess*. 2014. URL: <https://erikbern.com/2014/11/29/deep-learning-for-chess.html>.
- [2] Cameron B. Browne et al. «A Survey of Monte Carlo Tree Search Methods». En: *IEEE Transactions on Computational Intelligence and AI in Games* (2012). DOI: 10.1109/TCIAIG.2012.2186810.
- [3] Murray Campbell, A. Joseph Hoane y Feng-hsiung Hsu. «Deep Blue». En: *Artificial Intelligence* (2002). DOI: 10.1016/S0004-3702(01)00129-1.
- [4] Yu Nasu. «NNUE: Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi». En: *Ziosoft Computer Shogi Club* (2018). URL: https://www.apply.computer-shogi.org/wcsc28/appeal/the_end_of_genesis_T.N.K.evolution_turbo_type_D/nnue.pdf.
- [5] David Silver et al. «A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play». En: *Science* (2018). DOI: 10.1126/science.aar6404.
- [6] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. DOI: 10.48550/arXiv.1712.01815.
- [7] David Silver et al. «Mastering the game of Go without human knowledge». En: *Nature* (2017). DOI: 10.1038/nature24270.