Propuesta de Tesis de Licenciatura

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

Título tentativo: [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 Kaspárov— 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 [5], Monte Carlo Tree Search [2] (MCTS) o alguna de sus variantes [6, 13], con funciones de evaluación muy complejas y artesanales que se basan en conocimiento humano sobre el juego.

Hasta los 2010, el desarrollo de engines avanzaba a un paso lento pero consistente hasta que en 2017, Google DeepMind publicó AlphaGo Zero [11] y su sucesor AlphaZero [10, 9] (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 construidas con conocimiento humano, alteró el rumbo del desarrollo de todos los engines modernos (no sólo de Go y ajedrez). En 2018, Yu Nasu introdujo las redes neuronales "Efficiently Updatable Neural-Networks" [8] (ИИUЭ) 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 12[1] (2020). A partir de Stockfish 16.1[2] (2024) la evaluación se realiza exclusivamente mediante redes NNUE, eliminando todo el aspecto humano.

[1] Introducing NNUE evaluation (Stockfish 12)
[2] Removal 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.

Adicionalmente, se entrenarán las redes con dos técnicas distintas: la que se utiliza en el estado del arte para entrenar engines modernos y una técnica propuesta en [1], descritas en la sección de metodología.

Finalmente, se evaluará la performance de los modelos entrenados en: partidas entre ellos y contra otros engines, en la arena pública para bots de Lichess y en la resolución de puzzles con distintos niveles de dificultad y temáticas.

Actividades y metodología

Engine

Se implementará un engine de ajedrez con heurísticas y mejoras clásicas, usando una evaluación con NNUEs (cuantizadas y utilizando SIMD). 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.

Entrenamiento

Para entrenar los modelos, se probarán 2 técnicas:

"Estándar": Se toman posiciones aleatorias del dataset de partidas y se utiliza Stockfish (oráculo) a profundidad fija 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 (es el estado del arte).

"PQR": Esta técnica, inspirada en [1], no utiliza 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, $P \to Q$). Finalmente, se toma una posición R aleatoria tal que $P \to R$ 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 el otro, 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.

Dataset

Se utilizará el dataset abierto de Lichess [7] (CC0) que cuenta con 5.5 miles de millones de partidas públicas jugadas en el sitio, equivalentes a 1.71TB de PGNs[1] comprimidos. En el dataset hay más de 200 miles de millones 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 que también ofrece Lichess para evaluar el desempeño.

Factibilidad

Las técnicas y heurísticas para el desarrollo de engines clásicos cuentan con una extensa bibliografía [4]. La arquitectura de las redes NNUE, su cuantización y su entrenamiento estándar para Stockfish están descritas en [12]. Con estos recursos, la disponibilidad de los datos [7] y contando con el hardware necesario, es factible desarrollar esta tesis en el plazo estipulado.

[1] Portable Game Notation: un formato para grabar los movimientos de una partida y metadatos.

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] *Chess Programming Wiki*. URL: https://www.chessprogramming.org.

[5] Claude G. Diderich y Marc Gengler. «A Survey on Minimax Trees And Associated Algorithms». En: *Minimax and Applications*. Springer US, 1995. DOI: 10.1007/978-1-4613-3557-3_2.

[6] Ahmed Elnaggar et al. «A Comparative Study of Game Tree Searching Methods». En: *International Journal of Advanced Computer Science and Applications* (2014). DOI: 10.14569/IJACSA.2014.050510.

[7] *Lichess Database*. URL: https://database.lichess.org.

[8] 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.

[9] 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.

[10] David Silver et al. «Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm». En: (2017). DOI: 10.48550/arXiv.1712.01815.

[11] David Silver et al. «Mastering the game of Go without human knowledge». En: *Nature* (2017). DOI: 10.1038/nature24270.

[12] Official Stockfish. *nnue.md*. URL: https://github.com/official-stockfish/nnue-pytorch/blob/master/docs/nnue.md.

[13] Maciej Świechowski et al. «Monte Carlo Tree Search: a review of recent modifications and applications». En: *Artificial Intelligence Review* (2022). DOI: 10.1007/s10462-022-10228-y.