# 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

adasdasd asociadada sda sdaasd asociadada ads

# Antecedentes

adasdasdasd asociadaasd asociadadaasda sdasdas dado

# Actividades y metodología

# Factibilidad

El objetivo de la tesis es experimentar con diferentes feature sets en redes neuronales "NNUE" para engines de ajedrez.

**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. La idea es experimentar con diversos sets, teniendo de referencia los existentes y nuevos.

Por ejemplo podemos definir el feature set natural HALF-PIECE (ignorar 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 1.0 si el feature está activo (0 si no). Como tenemos 64 casillas, 6 tipos de piezas y 2 colores, tenemos $64 * 6 * 2 = 768$ features en este feature set.

A modo de referencia, el feature set actual de Stockfish, HALFKAV2_HM tiene 22528 features.

**NNUEs**

Las "Efficiently updatable neural networks" (NNUEs) son redes neuronales utilizadas como evaluación 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.

**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.

"$PQR$": 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, $P \to Q$). 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 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.

Como vemos en [1] si.

## References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.