Feature set analysis for chess ЭUИИ networks

Tesis de Licenciatura

Martín Emiliano Lombardo

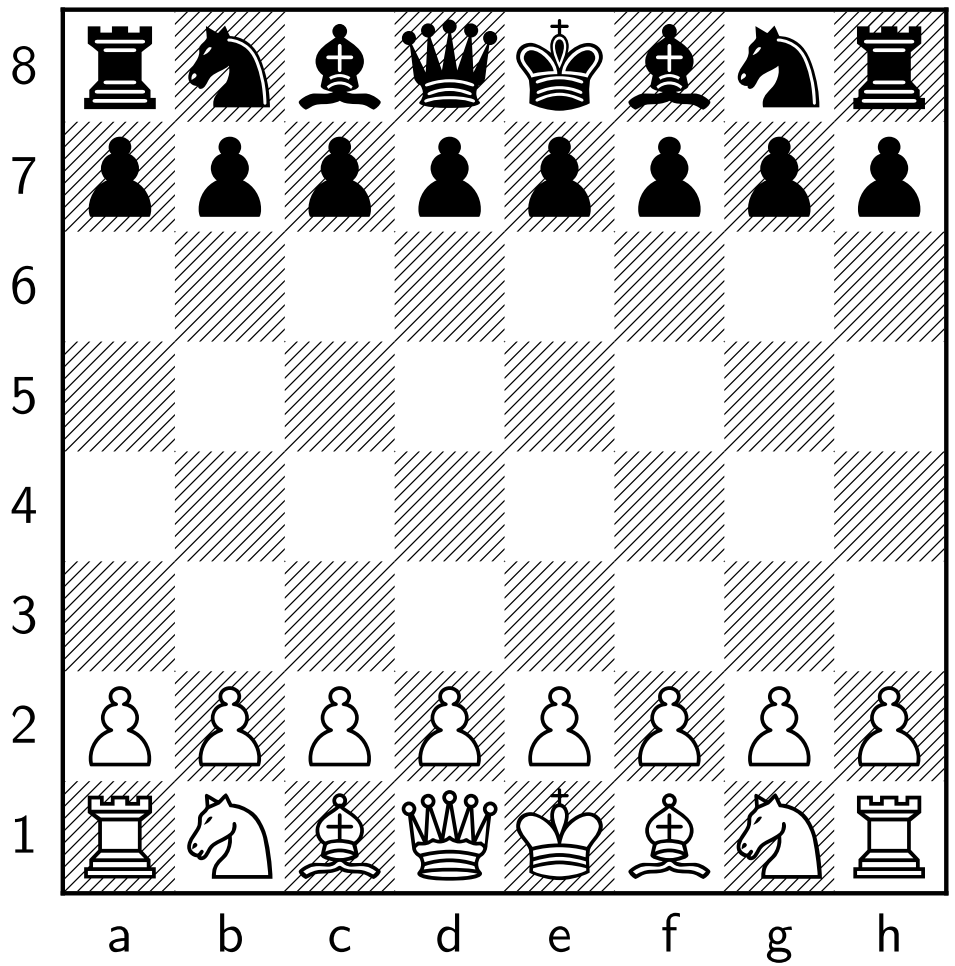
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

2024

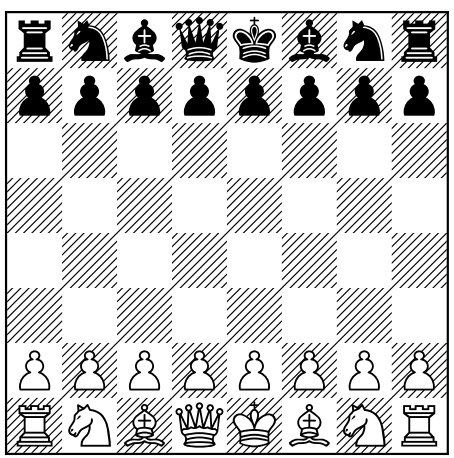
Introducción

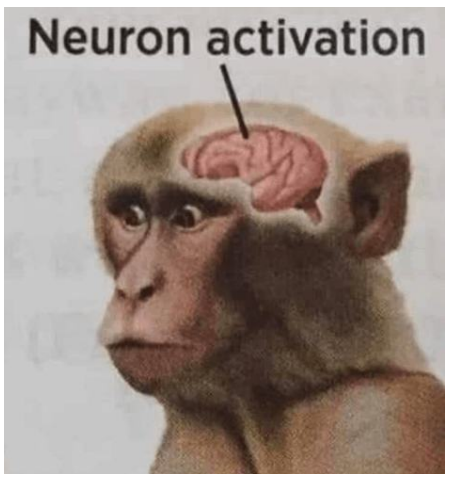
Ajedrez

- Dos jugadores
- Suma cero



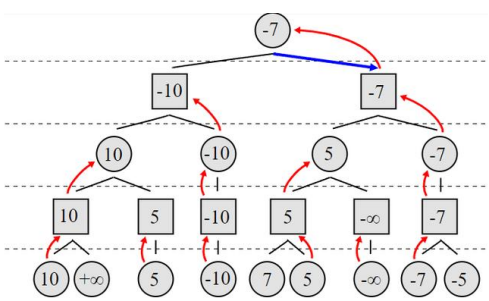
Humano vs. Computadora





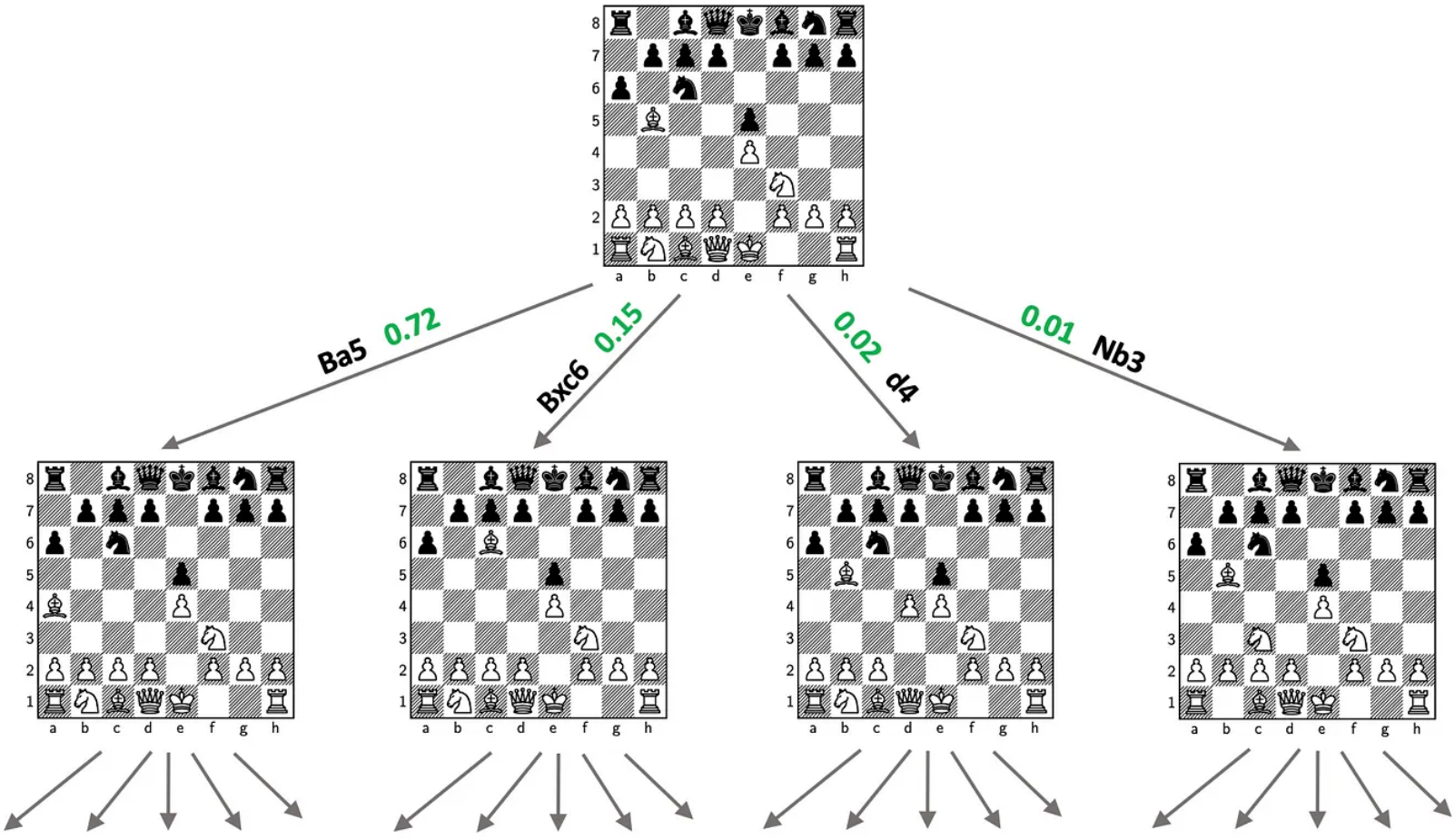
e2e4

Chess Engine



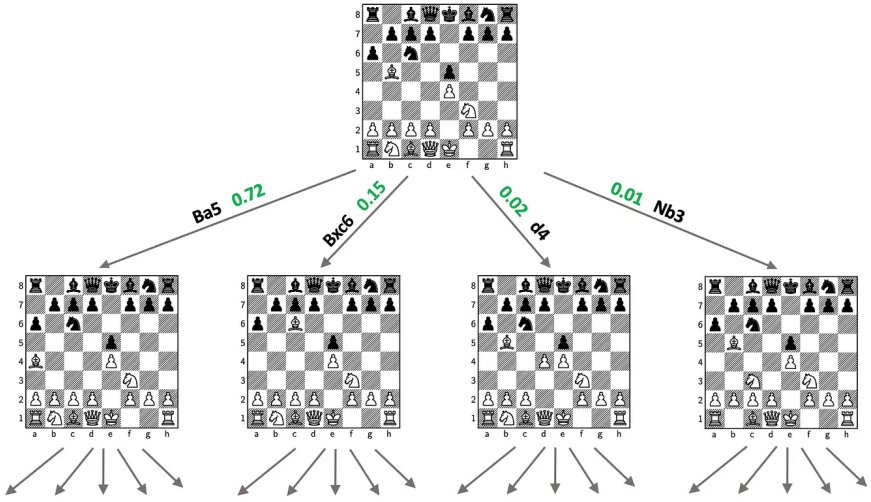
e2e4

Ajedrez como árbol

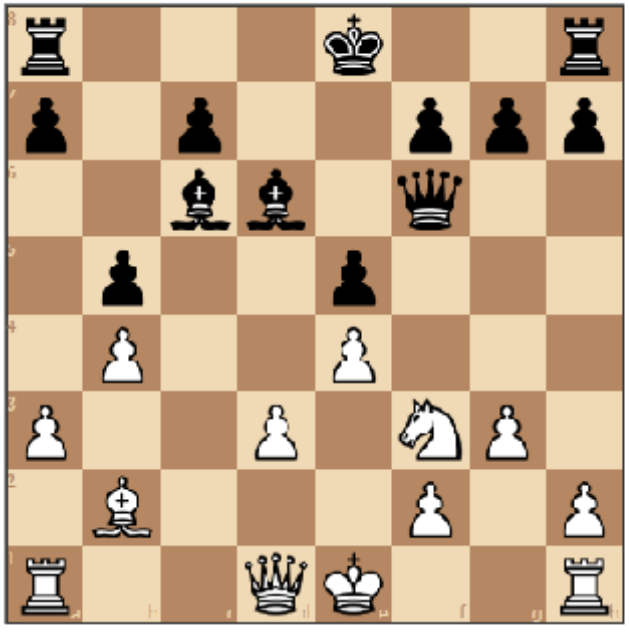


Motores de ajedrez (Chess Engines)

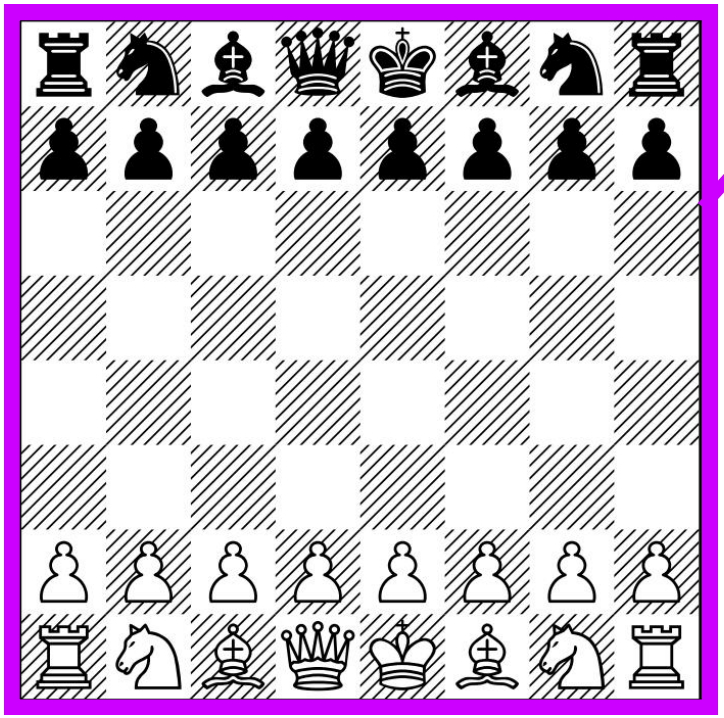
- Exploran el árbol de juego (Minimax, MCTS, etc.)
- Utilizan funciones de evaluación en las hojas
- La evaluación se propaga hacia arriba, según el algoritmo



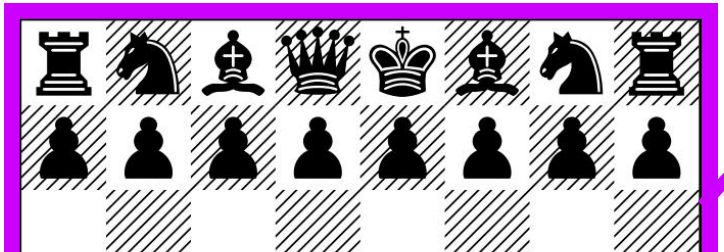
Función de evaluación

$$f\left(\right.$$  $$\left.\right) = 5$$

(adelanto) Feature set: ¿Cómo transformar la posición a un vector?



$$f(?) = ?$$



feature set!

Motores de ajedrez (breve historia)

Plan

asdasd

- Text visible on slide 1
- Text visible on slide 2
- Text visible on slide 3
- Text visible on slide 4

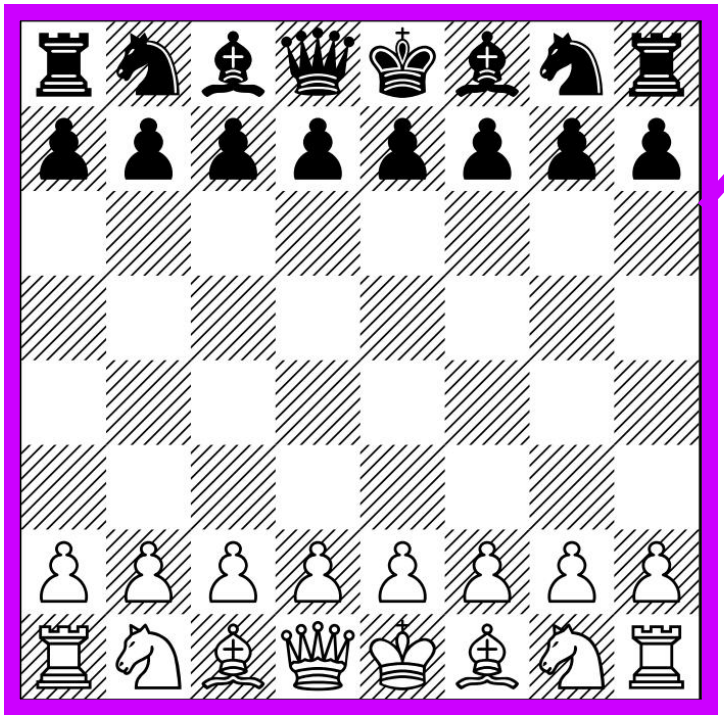
asdasd

Contenido

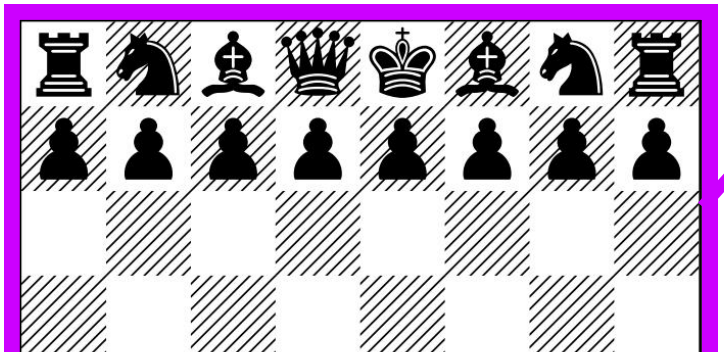
Engine

Feature set

¿Cómo transformar la posición a un vector?



$$f(?) = ?$$



feature set!

$$f(?) = ?$$

Definición

Un **feature set** S_P se define con un conjunto S y un predicado asociado $P(e)$, donde:

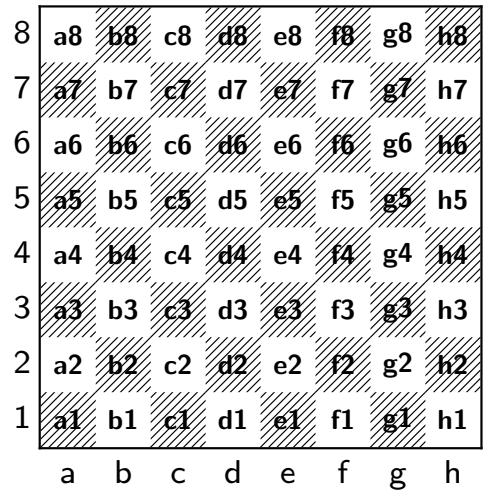
- S es un conjunto de conceptos (rol, color, celda, número, etc.).
- $P(e)$ es un predicado que determina si e está presente (o *activo*) en la posición (implícita).

- Cada elemento en S_P es un *feature*.
- Cada *feature* es un valor en el vector de entrada, valiendo 1 si está *activo* y 0 si no.

Ejemplos de S

Información posicional:

$$\textsc{Files} = \{a, b, ..., h\}$$
$$\textsc{Ranks} = \{1, 2, ..., 8\}$$
$$\textsc{Squares} = \{a1, a2, ..., h8\}$$

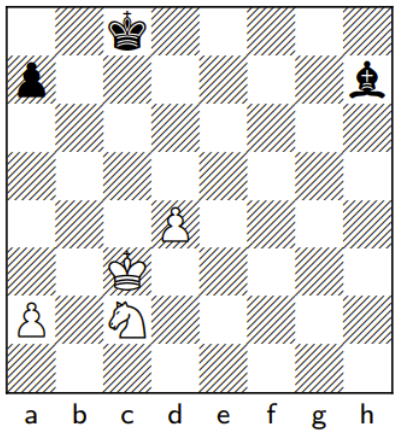


Información sobre las piezas:

$\textsc{Roles} = \{$ ♙ Pawn, ♘ Knight, ♗ Bishop, ♖ Rook, ♕ Queen, ♔ King

$\textsc{Colors} = \{$○ White, ● Black$\}$

Ejemplo completo



	Feature set	
	$(\textsc{Files} \times \textsc{Colors})_P$	$(\textsc{Files} \times \textsc{Roles})_Q$
Active features	$\langle$a, ○$\rangle$, $\langle$a, ●$\rangle$, $\langle$c, ●$\rangle$, $\langle$c, ○$\rangle$, $\langle$d, ○$\rangle$, $\langle$h, ●$\rangle$	$\langle$a, ♙$\rangle$, $\langle$c, ♔$\rangle$, $\langle$c, ♘$\rangle$, $\langle$d, ♙$\rangle$, $\langle$h, ♗$\rangle$

$P(\langle f, c\rangle)$: there is a piece in file f with color c.
$Q(\langle f, r\rangle)$: there is a piece in file f with role r.

Operación: Suma $\oplus$ (concatenación)

Hay veces que es útil combinar información de dos *feature sets*

$$S_P, T_Q : \text{feature sets}$$

$$S_P \oplus T_Q = (S \cup T)_R$$

$$\text{where } R(e) = \begin{cases} P(e) & \text{if } e \in S \\ Q(e) & \text{if } e \in T \end{cases}$$

Operación: Producto $\times$ (and)

$$S_P \times T_Q = (S \times T)_R$$

$$\text{where } R(\langle e_0, e_1 \rangle) = P(e_0) \ \wedge \ Q(e_1)$$

Feature set: All

La codificación más natural de una posición de ajedrez

$$\textsc{All} : (\textsc{Squares} \times \textsc{Roles} \times \textsc{Colors})_P$$

$P(\langle s, r, c \rangle)$: there is a piece in square s with role r and color c

- Es pequeño: $64 \times 6 \times 2 = 768$ *features*
- Es completo: contiene toda la información de la posición
- Es muy rápido computar cuáles *features* están activas

Feature set: King-All ó "KA"

Los engines modernos usan variaciones del siguiente feature set. Permite entender la posición en relación a la posición del rey:

$$\textsc{King-All} = \textsc{Square}_K \times \textsc{All}$$

$K(s)$: s is the square of the king of the side to move

- Es grande: $64 \times 768 = 49152$ *features*
- Es muy rápido como All
- Entrenarlo require un dataset más grande y lleva más tiempo (no me meto acá)

Feature sets: resumen

- $\boldsymbol{S}$: set of concepts (roles, colors, squares, files, ranks, etc.).
- $\boldsymbol{P(e)}$: predicate that defines when the feature e is present in the (implicit) position.
- $\boldsymbol{S_P}$: a feature set. Every element in S_P is a feature. Features that satisfy P are *active*.
- $S_P \times T_Q = (S \times T)_R$ where $R(\langle e_0, e_1 \rangle) = P(e_0) \ \wedge \ Q(e_1)$
- $S_P \oplus T_Q = (S \cup T)_R$ where $R(e) = \begin{cases} P(e) & \text{if } e \in S \\ Q(e) & \text{if } e \in T \end{cases}$

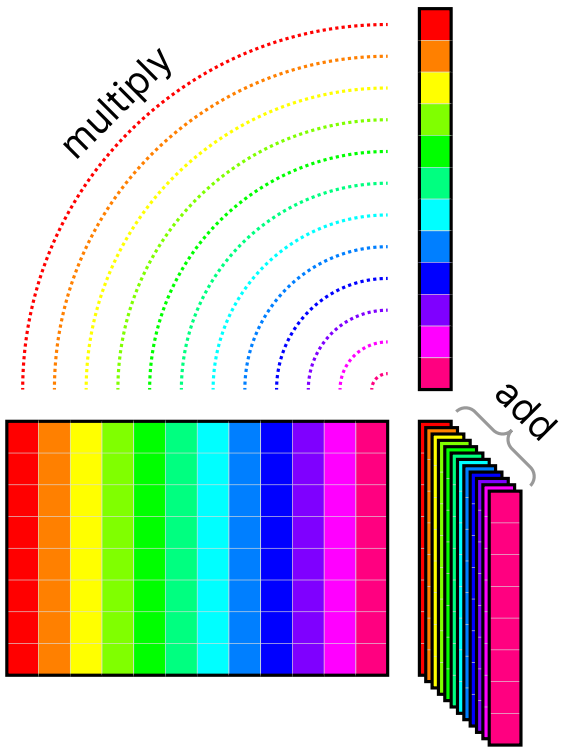
ƎUИИ (NNUE)

ƎUИИ: **E**fficiently **U**pdatable **N**eural **N**etworks

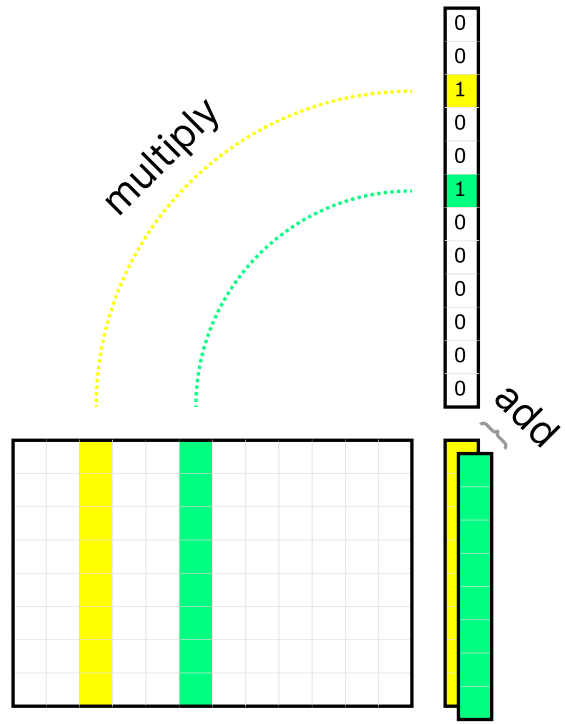
ƎUИИ: **N**eural **N**etworks

- El input es un vector one-hot generado por el *feature set*.
  - Debe tener pocos *features* activos (rala): introduce una cota superior.
- La red es una *feedforward* clásica con dos capas ocultas.

Linear layer



(c) Linear layer



(d) Linear layer with sparse inputs

Figure: Linear layer operation comparison. Figures from [18].

ƎUИИ: **E**fficient **U**pdates

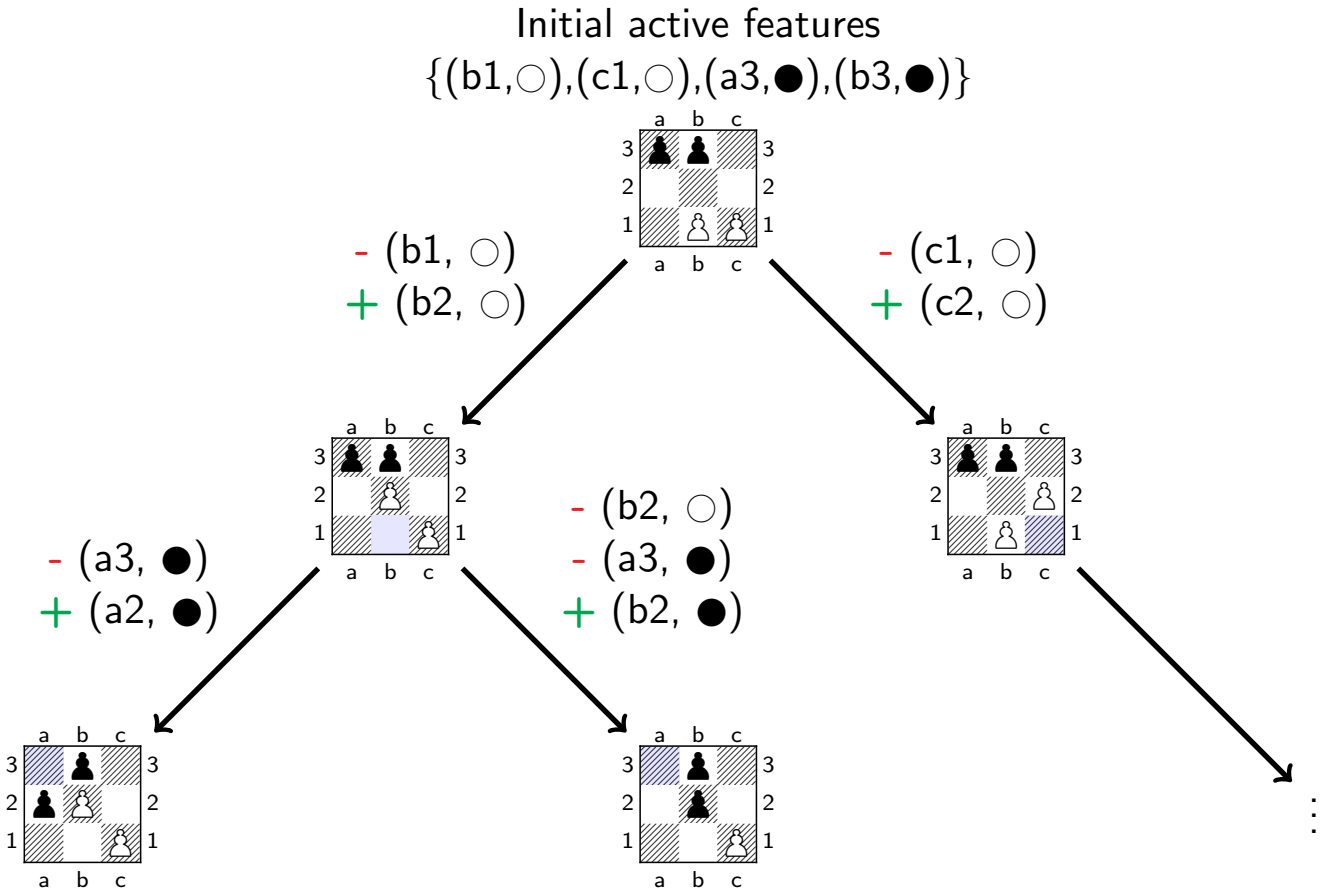


Figure: Partial tree of feature updates (removals and additions) for (Squares × Colors) (white's point of view) in a simplified 3x3

ƎUИИ: Tradeoff

motivacion comparacion de burns

Training

Experimentos

Setup de training

¿Qué hay que definir para entrenar una red?

- **Feature set**: determina la codificación y los patrones que se pueden aprender
- **Dataset**: datos de entrenamiento, visto anteriormente
- **Arquitectura de la red**: el tamaño de cada capa; L_1 y L_2
- **Método de entrenamiento**: PQR/target scores; determina el formato de las muestras y la loss function
- **Hiperparámetros**: learning rate, batch size, epochs, etc.

Setup de evaluación

¿Cómo evalúo el performance de una red entrenada?

- **Loss** (train y val.): indica la calidad de las predicciones.
  - Permite detectar overfitting y otros problemas
- **Puzzle accuracy**: porcentaje de movimientos acertados en puzzles de Lichess.
  - Sólo hay un movimiento correcto
  - Proxy (muy malo) de la fuerza de la red
- **Elo relativo**: la medida más común para comparar engines.
  - Se realizan torneos de 100ms por movimiento
  - El elo es calculado a partir de Ordo

Baseline: motivación

Busco fijar el setup de entrenamiento con valores razonables

- El feature set va a cambiar cada experimento
- El dataset está fijo
- El método de entrenamiento principal es *target scores*

Entonces queda por determinar...

- La arquitectura de la red (L_1 y L_2)
- Los hiperparámetros

Baseline: hiperparámetros

Los hiperparámetros fueron seleccionados en base al trainer oficial de Stockfish:

- **Learning rate**: 0.0005
- **Exponential decay**: 0.99
- **Batch size**: 16384
- **Epoch size**: 100 million
  - cada epoch realiza 6104 batches
- **Epochs**: 256
  - cada run observa *25.6 billion* samples

Baseline: experimento

Sólo queda buscar parámetros L_1 y L_2 razonables. Realizo una búsqueda en grilla con:

- L1 $\in \{256, 512, 1024, 2048\}$
- L2 $\in \{32, 64, 128, 256\}$

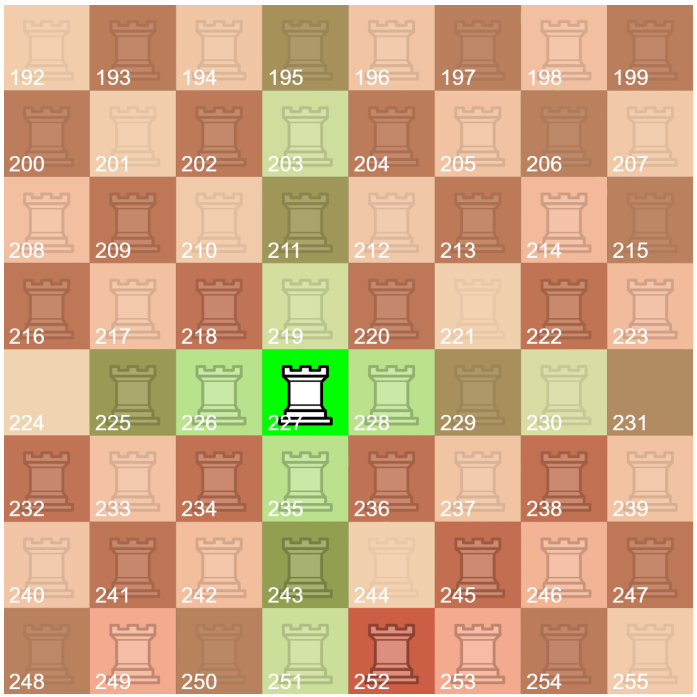
El feature set a utilizar es $\textsc{All}[768]$.

Baseline: resultados

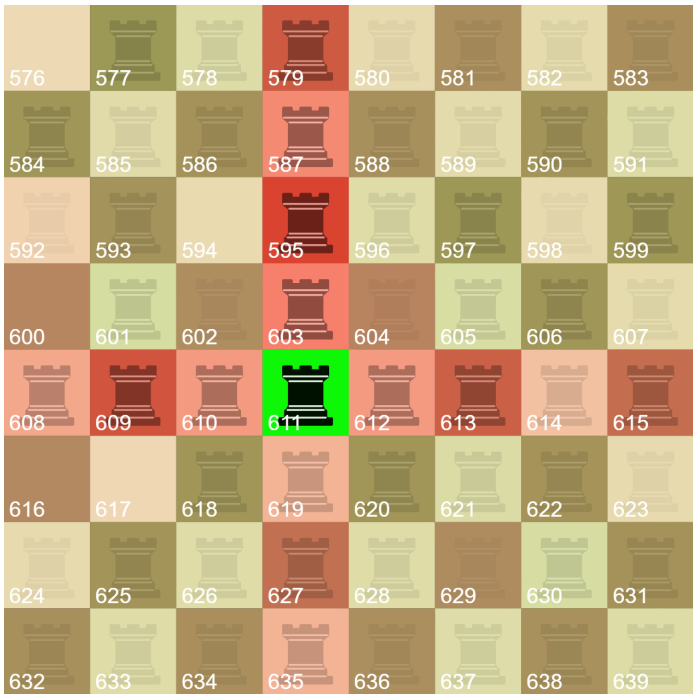
Baseline: conclusión

- **L2=32**. El performance cae dramáticamente si L2 aumenta, utilizo el más bajo.
  - Sería buena idea probar valores más chicos de L2.
- **L1=512**. Es el mejor valor para L2=64 y L2=128, y en margen de error para L2=32.
  - Además es el más rápido de entrenar.

Axis encoding: motivación



(a) ○ White



(b) ● Black

Figure: Weights of **a neuron** in the L1 layer, which are connected to features in ALL where the role is ♖ Rook. The intensity represents the weight value, and the color represents the sign (although not relevant).

Conclusión

Ajedrez

- asdasd