Feature set analysis for chess ƎUИИ networks

Tesis de Licenciatura

Martín Emiliano Lombardo

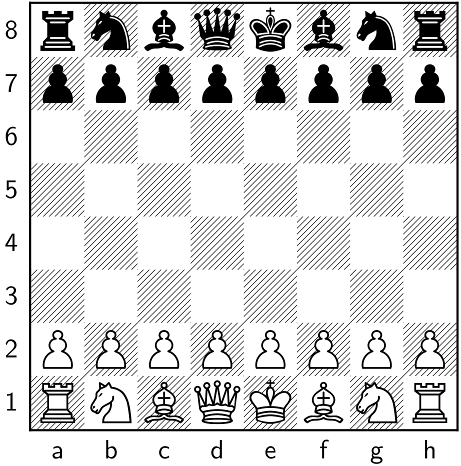
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

2024

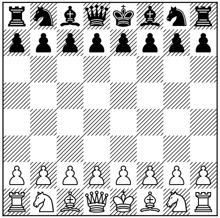
Introducción

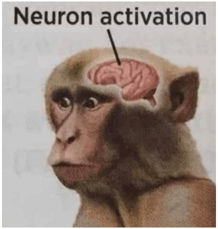
Ajedrez

- Dos jugadores
- Suma cero



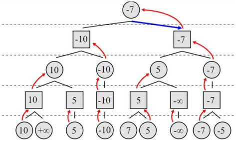
Humano vs. Computadora





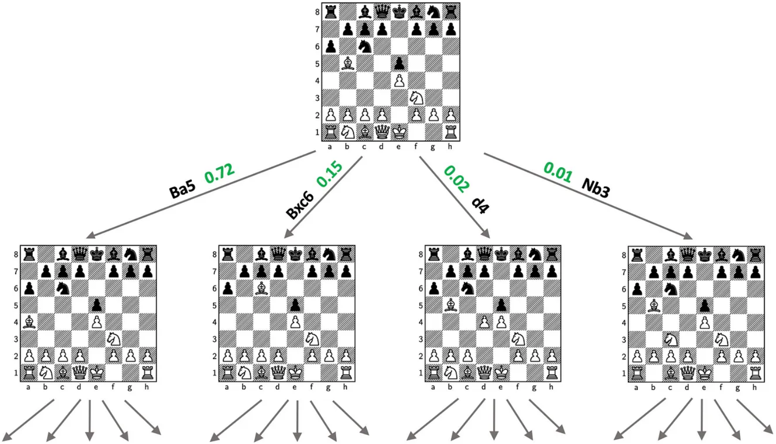
e2e4

Chess Engine



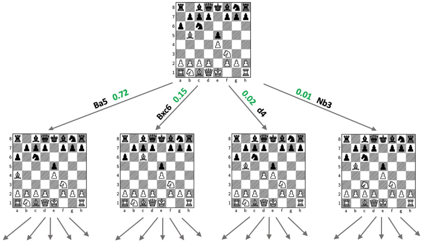
e2e4

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

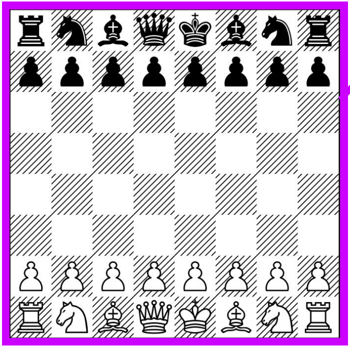


Función de evaluación o "eval"

$f\Bigg($  $\Bigg) = 5$

Intentan resumir todo el subárbol en un solo número.
En general son creadas *artesanalmente*

(adelanto) Feature sets: ¿Cómo transformar la posición a un vector para usar NNs?



feature set!

$$f(?) = ?$$

Motores de ajedrez (breve historia)

- **1950s**: Se desarrollan los primeros *algoritmos* de ajedrez
- **1960s+**: Aparecen los primeros *motores de ajedrez*, lentos y débiles
- **1997** (hito): IBM DeepMind vence a Garry Kasparov en un torneo
- **2017 y 2018**: Google DeepMind publica AlphaGo Zero y su sucesor AlphaZero
  - se reemplaza la función de evaluación por una red neuronal
- **2018**: Yu Nasu introduce las redes ƎUИИ para Shogi
- **2020**: Stockfish 12 introduce redes ƎUИИ en su evaluación
  - se utilizan a la par de evaluaciones artesanales
- **2024**: Stockfish 16.1 elimina todo aspecto humano de su evaluación, todo es mediante redes neuronales

Plan de la tesis

El objetivo principal es **proponer y evaluar novedosos feature sets**. Además, **probar una técnica de entrenamiento** no convencional.

El plan de la presentación es el siguiente:

- Implementación de un motor de ajedrez clásico
- Definición y ejemplos de feature sets
- NNUEs
- Entrenamiento de las redes
- Experimentos

Motor

Motor de ajedrez

Para evaluar las redes NNUEs es necesario un motor de ajedrez.

Buscamos construir un **motor de ajedrez clásico**, con **optimizaciones clásicas** pero **que use NNUEs** para evaluar posiciones.

Minimax

Primera idea: evalúo todas las posiciones a las que me puedo mover y elijo la mejor.

Pero si extendemos la idea recursivamente... es el algoritmo **minimax**.

- △ **Maximizing nodes**: nuestro jugador. Elige el movimiento que maximice la evaluación.
- ▽ **Minimizing nodes**: el oponente. Elige el movimiento que minimiza la evaluación.

Minimax

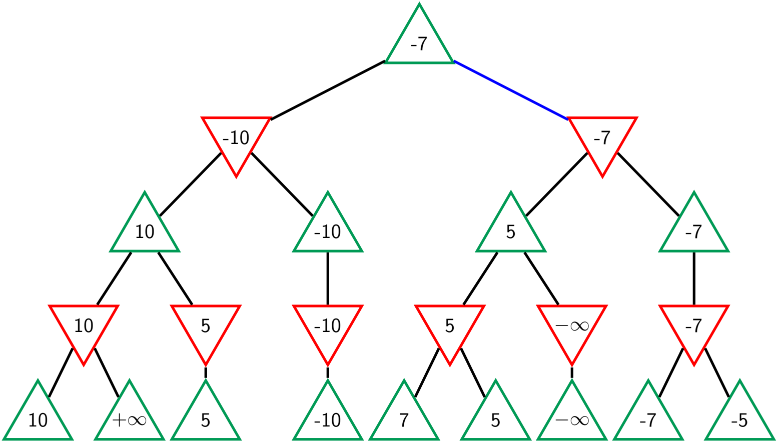


Figure: Un árbol minimax de 4 de profundidad. El "mejor" movimiento para el jugador maximizador es el que lleva a la evaluación más alta, macada en azul.

Iterative deepening

No queremos hacer minimax a una profundidad fija, si no a un tiempo fijo (100 milisegundos).

Iterative deepening es una técnica que consiste en hacer minimax a profundidades crecientes, hasta que se acabe el tiempo.

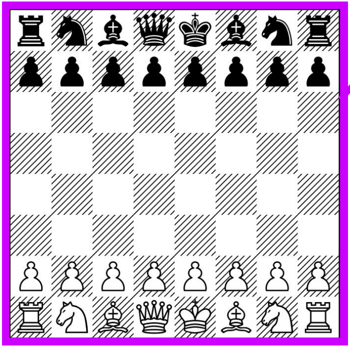
Che pero no pierdo todo el cómputo que hice en la iteración anterior? **Si, pero...**

Optimizaciones

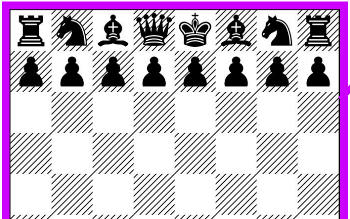
- Poda Alpha-beta (anim)
- Reordenamiento de movimientos (peor caso Minimax)
  - MVV/LVA (Most Valuable Victim/Least Valuable Attacker)
  - $\downarrow$
- Tablas de transposición: un caché

Feature set

¿Cómo transformar la posición a un vector?



$$f(?) = ?$$



feature set!

$$f(?) = ?$$

Definición

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

- $\boldsymbol{S}$ es un conjunto de conceptos (rol, color, celda, número, etc.).
- $\boldsymbol{P(e)}$ 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:

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

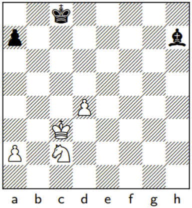
8	a8	b8	c8	d8	e8	f8	g8	h8
7	a7	b7	c7	d7	e7	f7	g7	h7
6	a6	b6	c6	d6	e6	f6	g6	h6
5	a5	b5	c5	d5	e5	f5	g5	h5
4	a4	b4	c4	d4	e4	f4	g4	h4
3	a3	b3	c3	d3	e3	f3	g3	h3
2	a2	b2	c2	d2	e2	f2	g2	h2
1	a1	b1	c1	d1	e1	f1	g1	h1
	a	b	c	d	e	f	g	h

Información sobre las piezas:

ROLES = { ♙ Pawn, ♘ Knight, ♗ Bishop, ♖ Rook, ♕ Queen, ♔ King

COLORS = {○ White, ● Black}

Ejemplo completo



	Feature set	
	(FILES × COLORS)$_P$	(FILES × ROLES)$_Q$
Active features	⟨a, ○⟩, ⟨a, ●⟩, ⟨c, ●⟩, ⟨c, ○⟩, ⟨d, ○⟩, ⟨h, ●⟩	⟨a, ♙⟩, ⟨c, ♔⟩, ⟨c, ♘⟩, ⟨d, ♙⟩, ⟨h, ♝⟩

$P(\langle f, c\rangle)$: 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*

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

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

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

Operación: Producto × (and)

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

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

Feature set: $\textsc{All}$

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: $\textsc{King-All}$ ó "KA"

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

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

$$K(s)\text{: } s \text{ is the square of the king of the side to move}$$

- Es grande: $64 \times 768 = 49152$ *features*
- Es muy rápido como $\textsc{All}$
- 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 de 3 fully connected.
  - con activaciones ClippedReLU

ƎUИИ: La red

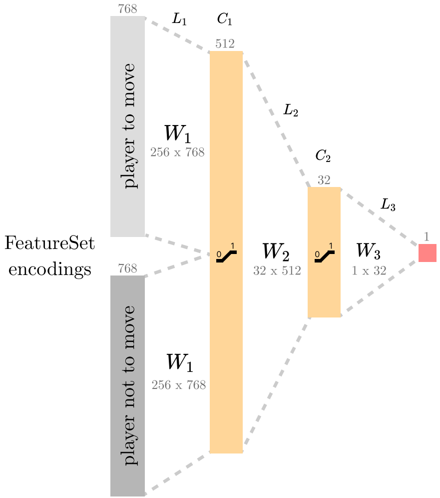
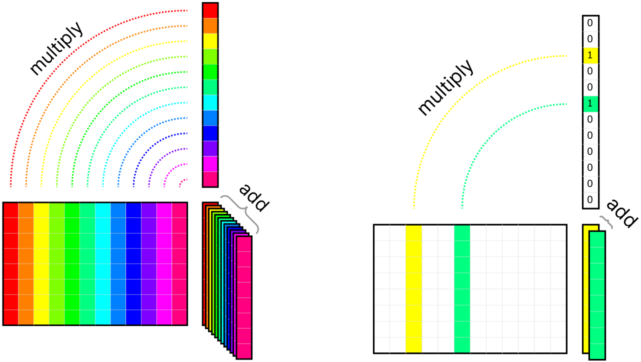


Figure: Neural network architecture with $\boldsymbol{N} = 768$, $\boldsymbol{M} = 256$, $\boldsymbol{O} = 32$. Not to scale.

Linear layer



(a) Linear layer

(b) Linear layer with sparse inputs

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

Concatenación de la primera capa

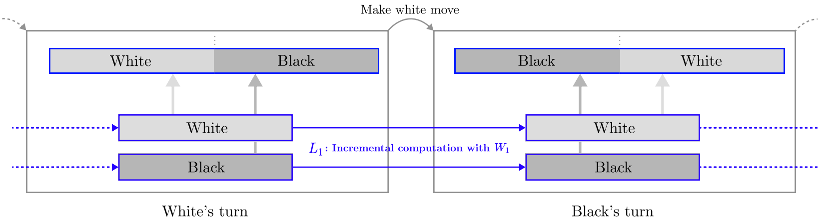


Figure: Concatenation of the first layer's output after a move is made. Inspired by a CPW figure.

ƎUИИ: **E**fficient **U**pdates

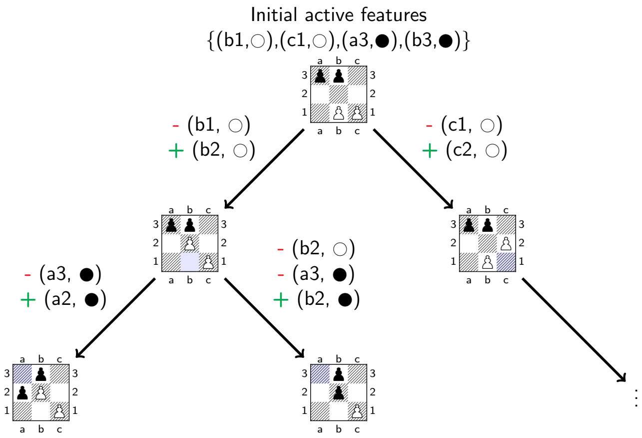


Figure: Árbol parcial de feature updates (agregados y borrados) para (Squares × Colors) (POV blanco) en un tablero simplificado 3x3 de peones.

ƎUИИ: Tradeoff

Tiempo de inferencia vs. **nodos visitados**.

Si la red es rápida (y por lo tanto débil), se pueden visitar más nodos y llegar más profundo.
Si se tienen predicciones de mayor calidad (y por lo tanto más lento), se visitan menos nodos pero con evaluaciones más precisas.

No es directo determinar qué resulta más fuerte.

Entrenamiento

Dataset

- Para entrenar estas redes se necesitan **decenas de miles de millones** de samples. Generarlo a mano es inviable.
- Uso el mismo dataset usado para entrenar Stockfish 16.1 (135GB, 48.4 billion)
- Cada sample se ve así:

FEN	,	**Score**	,	**Best move**

- 130 GB $\rightarrow$ 2 TB $\rightarrow$ 522 GB

Métodos de entrenamiento

1. **Target scores** o **Score target**: Utiliza las evaluaciones del dataset como target.
2. **PQR**: Utiliza dos principios *razonables* para armar una función de pérdida.

Score-space a WDL-space

- **Score-space**: los scores en el dataset están entre [-10000, 10000] (*centipawn* o proporcional)
- **WDL-space**: otra escala donde 0 es perder, 0.5 es empate y 1 es ganar

Queremos que la red genere valores en **score-space**, pero para las funciones de pérdida es mejor usar **WDL-space**.

Score-space a WDL-space

El modelo WDL dice que el winrate se puede modelar como una función de la evaluación.
Los datos muestran que la función sigmoide da una buena aproximación:

$$\mathcal{W}(f(P)) = \sigma\left(\frac{f(P)-a}{b}\right) = \frac{1}{1+e^{-\frac{f(P)-a}{b}}}$$

Score-space a WDL-space

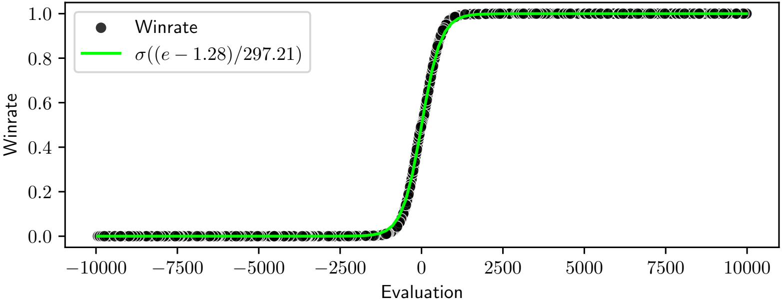


Figure: Modelo WDL ajustado a 100 million de evaluaciones en el dataset.

Score-space a WDL-space

¿Para qué WDL?

- Las evaluaciones están mas “cerca” en WDL-space:
  - 7500 vs 8000: 1% winrate
  - 50 vs 550 : 30% winrate
- Se puede interpolar con los resultados (no lo hago)
  - $\lambda \cdot \mathcal{W}(f(P)) + (1 - \lambda) \cdot r$
- Gradientes más chicos

Método 1: Target scores

Usamos los valores del dataset como target.
La función de pérdida es **Mean Square Error (MSE)** con potencia 2.6.

$$\mathcal{L}(y, f(x, \boldsymbol{W})) = \frac{1}{N}\sum_{i}^{N} \left|\mathcal{W}(y_i) - \mathcal{W}(f(x_i, \boldsymbol{W}))\right|^{2.6}$$

donde. . .

1. N es la cantidad de muestras.
2. y son las evaluaciones objetivo.
3. f es el modelo.
4. x son los inputs (vector del feature sets).
5. $\boldsymbol{W}$ son los parámetros del modelo.
6. $\mathcal{W}$ es la función de winrate que mapea de score-space a WDL-space.

Método 2: PQR

Técnica vista en un blogpost de 2014 por Erik Bernhardsson, que se basa en dos principios:

1. Para dos posiciones en suseción $P \to Q$ observadas en el dataset, tenemos que $f(P) = -f(Q)$. Esto es porque el juego es de suma cero.
2. Ir desde P, no a la posición observada Q, sino a una posición *random* $P \to R$, se debe cumplir $f(R) > f(Q)$ porque un movimiento random es mejor para el siguiente jugador y peor para el que hizo el movimiento.

Se puede construir una función de pérdida que refleje la igualdad en (1) y la desigualdad en (2).

Método 2: PQR

La función de pérdida es la suma de la log-verosimilitud negativa de las inecuaciones:

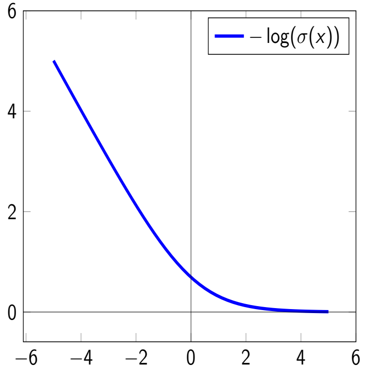
- $f(R) > f(Q)$
- $f(P) > -f(Q)$
- $f(P) < -f(Q)$

Método 2: PQR

$$\mathcal{L}(x^P, x^Q, x^R, \boldsymbol{W}) = \frac{1}{N}\sum_{i}^{N} -\log\left(\sigma(r_i - q_i)\right) \\ -\log\left(\sigma(p_i + q_i)\right) \\ -\log\left(\sigma(-(p_i + q_i))\right)$$

1. x^i son los inputs (vector del feature sets) para las posiciones $i \in \{P, Q, R\}$.
2. $\overline{\mathcal{W}}(x) = 2\mathcal{W}(x) - 1$ es una función que mapea de WDL-space [0, 1] a $[-1, 1]$, así $\overline{\mathcal{W}}(x) = -\overline{\mathcal{W}}(-x)$.
3. $p_i = \overline{\mathcal{W}}(f(x_i^P, \boldsymbol{W}))$, $q_i = \overline{\mathcal{W}}(f(x_i^Q, \boldsymbol{W})$, $r_i = \overline{\mathcal{W}}(f(x_i^R, \boldsymbol{W})$.

Método 2: PQR



La función se acerca a 0 cuando x crece y se acerca a ∞ cuando x tiende a $-\infty$.

Método 2: PQR

Veamos cada uno de los términos:

1. $-\log(\sigma(r_i - q_i))$: Este término es chico cuando $r_i > q_i$, y grande cuando $r_i < q_i$.
2. $-\log(\sigma(p_i + q_i))$: Este término es chico cuando $p_i > -q_i$, y grande cuando $p_i < -q_i$.
3. $-\log(\sigma(-(p_i + q_i))$: Este término es chico cuando $p_i < -q_i$, y grande cuando $p_i > -q_i$.

El término (1) sostiene la inecuación $f(R) > f(Q)$, y los términos (2) y (3) la igualdad $f(P) = -f(Q)$.

Experimentos

Setup de training

Recapitulando... ¿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

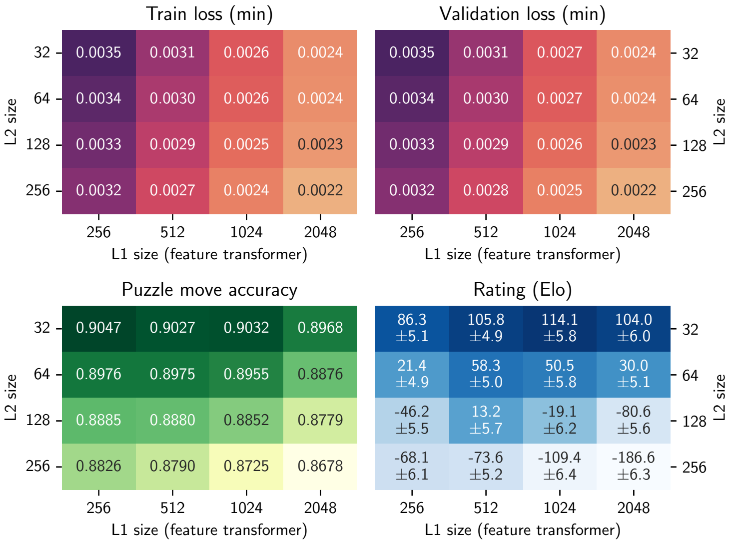
Baseline: experimento

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].

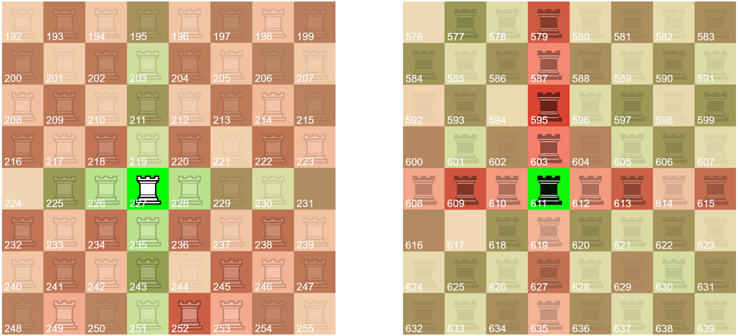
Baseline: resultados



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).

Axis encoding: motivación

La red detecta patrones parecidos a los movimientos de las piezas. Para hacerle la vida más fácil a la red, propongo agregar features como:

"*there is a ○ White ♖ Rook in the 4th rank*"

Axis encoding: experimento

⟷	↕	⤢	⤡
Horizontal (across files)	Vertical (across ranks)	Diagonal 1	Diagonal 2

Depiction	**Block name**	**Definition**	**Number of features**
⟷	H	$(\textsc{Files} \times \textsc{Roles} \times \textsc{Colors})_P$	96
↕	V	$(\textsc{Ranks} \times \textsc{Roles} \times \textsc{Colors})_P$	96
⤢	D1	$(\textsc{Diags1} \times \textsc{Roles} \times \textsc{Colors})_P$	180
⤡	D2	$(\textsc{Diags2} \times \textsc{Roles} \times \textsc{Colors})_P$	180

$\mathsf{P}(\langle x, r, c\rangle)$: there is a piece in x with role r and color c

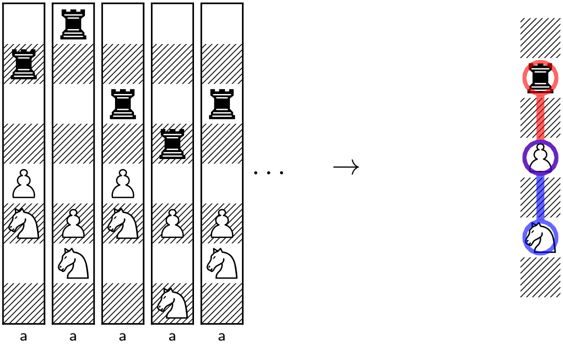
Axis encoding: experimento

Depiction	Feature set	Number of features
$\leftrightarrow \oplus \updownarrow$	$\mathrm{H} \oplus \mathrm{V}$	192
$\nearrow\!\!\!\!\swarrow \oplus \nwarrow\!\!\!\!\searrow$	$\mathrm{D1} \oplus \mathrm{D2}$	360
$\leftrightarrow \oplus \updownarrow \oplus \nearrow\!\!\!\!\swarrow \oplus \nwarrow\!\!\!\!\searrow$	$\mathrm{H} \oplus \mathrm{V} \oplus \mathrm{D1} \oplus \mathrm{D2}$	552
$\textsc{All} \oplus \leftrightarrow \oplus \updownarrow$	$\textsc{All} \oplus \mathrm{H} \oplus \mathrm{V}$	960
$\textsc{All} \oplus \nearrow\!\!\!\!\swarrow \oplus \nwarrow\!\!\!\!\searrow$	$\textsc{All} \oplus \mathrm{D1} \oplus \mathrm{D2}$	1128
$\textsc{All} \oplus \leftrightarrow \oplus \updownarrow \oplus \nearrow\!\!\!\!\swarrow \oplus \nwarrow\!\!\!\!\searrow$	$\textsc{All} \oplus \mathrm{H} \oplus \mathrm{V} \oplus \mathrm{D1} \oplus \mathrm{D2}$	1320

Axis encoding: resultados

Feature set	Number of features	Val. loss *min*	Rating *elo (rel. to* ALL*)*	Puzzles *move acc.*
↔ ⊕ ↕	192	0.005810	-384.3 ± 5.1	0.8618
⤢ ⊕ ⤡	360	0.006707	-444.1 ± 5.1	0.8517
↔ ⊕ ↕ ⊕ ⤢ ⊕ ⤡	552	0.003907	-183.5 ± 4.1	0.8748
ALL (reference)	768	0.003134	**0.0**	0.8865
ALL ⊕ ↔ ⊕ ↕	960	0.003082	-27.1 ± 4.1	0.8851
ALL ⊕ ⤢ ⊕ ⤡	1128	0.003087	-26.1 ± 3.8	0.8814
ALL ⊕ ↔ ⊕ ↕ ⊕ ⤢ ⊕ ⤡	1320	**0.003067**	-58.7 ± 3.7	0.8766

Pairwise axes: motivación



Configuraciones distintas,
situaciones similares

Las mismas dos features
(par rojo y par azul)

Pairwise axes: motivación

Comparando con el experimento anterior, es más específico en vez de más general:

"*there is a ○ White ♖ Rook in the 4th rank*"

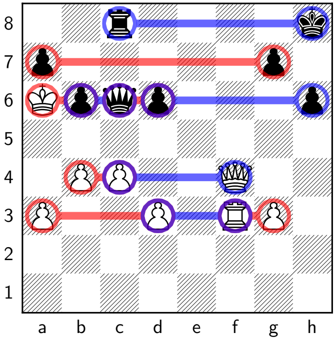
vs.

"*there is a ● Black ♖ Rook next to a ○ White ♙ Pawn in the 'a' file*"

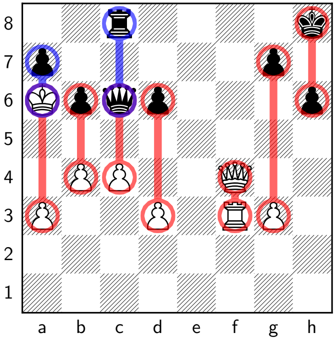
Pairwise axes: experimento

D.	Block name	Definition	Num. of features
	PH	$(\text{Ranks} \times (\text{Roles} \times \text{Colors}) \times (\text{Roles} \times \text{Colors}))_P$ $P(\langle r, r_1, c_1, r_2, c_2 \rangle)$: there is a piece in rank r with role r_1 and color c_1 to the left of a piece with role r_2 and color c_2	1152
	PV	$(\text{Files} \times (\text{Roles} \times \text{Colors}) \times (\text{Roles} \times \text{Colors}))_Q$ $Q(\langle f, r_1, c_1, r_2, c_2 \rangle)$: there is a piece in file f with role r_1 and color c_1 below a piece with role r_2 and color c_2	1152

Pairwise axes: experimento



Pairwise horizontal (PH)



Pairwise vertical (PV)

Pairwise axes: experimento

Los feature sets a entrenar son:

- $\textsc{All} \oplus \mathrm{PH}$ (1920 features)
- $\textsc{All} \oplus \mathrm{PV}$ (1920 features)
- $\textsc{All} \oplus \mathrm{PH} \oplus \mathrm{PV}$ (3072 features)

Pairwise axes: resultados

Feature set	Number of features	Val. loss *min*	Rating *elo (rel. to* ALL*)*
ALL (reference)	768	0.003134	**0.0**
ALL ⊕	1920	0.003033	-38.2 ± 4.8
ALL ⊕	1920	0.002946	-8.4 ± 5.0
ALL ⊕ ⊕	3072	**0.002868**	-37.6 ± 4.9

- Reducir el número de pairs puede llevar a una mejora por sobre ALL (ej. ♙)

Mobility: motivación

- La *mobilidad* en ajedrez es una medida de la cantidad de movimientos que puede hacer un jugador en una posición.
- Un paper de Eliot Slater (1950) mostró que hay una correlación entre la mobilidad de un jugador y la cantidad de partidas ganadas.
- Se usa en funciones de evaluación hechas a mano.
- Propongo agregar mobilidad como features en la red.

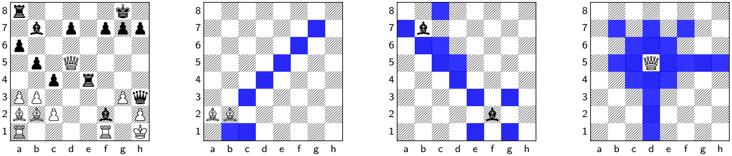
Mobility: experimento

Hay dos maneras de codificar la mobilidad:

- Bitsets (por rol/color)
- Cantidades (por rol/color)

Mobility: experimento (bitsets)

Los features proveen **las celdas** a las que una pieza de determinado rol/color puede moverse.



Board

○White ♗ Bishop

●Black ♗ Bishop

○White ♕ Queen

...

La cantidad de features es $64 \times 6 \times 2 = 768$, la misma que ALL.

Mobility: experimento (counts)

Los features proveen **la cantidad de celdas** a las que una pieza de determinado rol/color puede moverse. Esto reduce la cantidad de features significativamente.

Piece role	**Min**	**Max**
♙ Pawn	0	8+
♘ Knight	0	15+
♗ Bishop	0	16+
♖ Rook	0	25+
♕ Queen	0	25+
♔ King	0	8

Mobility: experimento (counts)

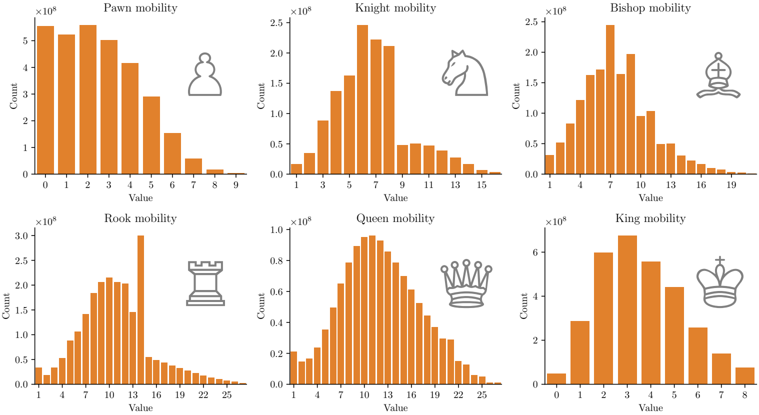


Figure: Total mobility values for each piece on the board. Computed using 2 billion boards. The value 0 for the ♘ Knight, ♗ Bishop, ♖ Rook, and ♕ Queen has been excluded from the plot, as it is very common.

Mobility: experimento

Block name	Definition	Number of features
MB	$(\textsc{Squares} \times \textsc{Roles} \times \textsc{Colors})_P$ $\mathrm{P}(\langle s, r, c \rangle)$: there is a piece of role r and color c that **can move to** square s	768
MC	$(\{0, 1, \ldots\} \times \textsc{Roles} \times \textsc{Colors})_P$ $\mathrm{P}(\langle m, r, c \rangle)$: the value of mobility for a piece of role r and color c is m	206

Los feature sets a entrenar son: $\textsc{All} \oplus \mathrm{MB}$ (1536 features) y $\textsc{All} \oplus \mathrm{MC}$ (974 features).

Mobility: resultados

Table: Mobility encodings results

Feature set	**Number of features**	**Val. loss** *min*	**Rating** *elo (rel. to* ALL*)*
ALL (reference)	768	0.003134	**0.0**
ALL $\oplus$ MB	1536	0.002824	-260.9 $\pm$ 5.4
ALL $\oplus$ MC	974	0.003032	-280.9 $\pm$ 5.6

- Las predicciones mejoran muy poco (el loss no se reduce tanto).
- Por ende, el costo de las actualizar los features es más alto al beneficio que aportan.
- MB tiene más updates que MC, pero menor loss que compensa.

Feature set statistics

Depiction	Feature block	Number of features	Average features...		
			active per position	added per move	removed per move
	ALL	768	14.68	0.98	0.60
	H	96	14.68	0.60	0.43
	V	96	14.68	0.61	0.43
	D1	180	14.68	0.77	0.52
	D2	180	14.68	0.77	0.52
	PH	1152	8.23	0.92	0.57
	PV	1152	8.30	0.83	0.53
MB	MB	768	48.93	5.68	4.35
MC	MC	206	12.00	2.34	1.48

PQR: motivación

Recordando...

- **P**: Una posición en el dataset
- **Q**: La posición obtenida a partir de aplicar el "mejor" movimiento a P, según el dataset
- **R**: Una posición aleatoria obtenida a partir de P, tal que $R \neq Q$

Y los principios:

1. Si $P \to Q$, entonces $f(P) = -f(Q)$ (suma cero)
2. Si $P \to R$ tal que $R \neq Q$, entonces $f(R) > f(Q)$

PQR: motivación

¿Los principios funcionan en la práctica? Veamos...

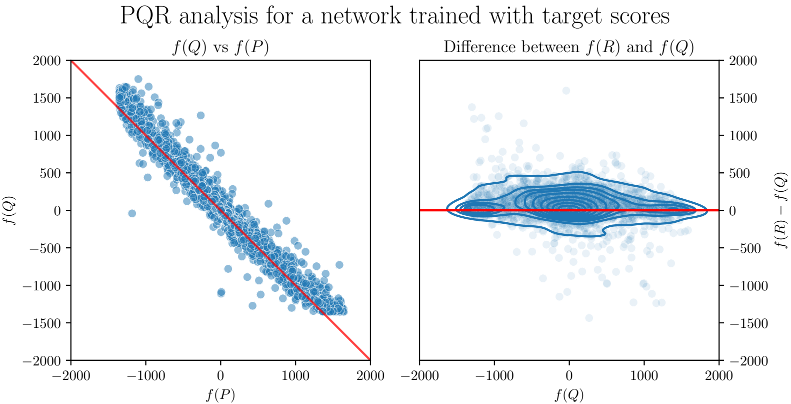


Figure: Analysis of $N = 4000$ PQR samples using a model trained with target scores and the feature set ALL.

PQR: experimento

A. Entrenar de cero, directamente con PQR
   - no espero que sea mejor que target scores

B. Continuar de un checkpoint entrenado con el otro método
   - no tiene que aprender tanto de entrada
   - mejor caso: mejora lentamente
   - peor caso: se "olvida" todo lo anterior (resulta peor)
   - se entrena con distintos learning rates

PQR: experimento

Eligiendo R.

Conclusión

adasdas

- asdasd