"Red neuronal que juegue ajedrez cuántico a partir de simulaciones con Montecarlo"

Autor/Autores:

1. Piero Marcelo Pastor Pacheco 20210836
2. Fabrizio Gabriel Gómez Buccallo 20212602
3. Jean Paul Tomasto Cordova 20202574
4. Jesus Mauricio Huayhua Flores 20196201
5. Pablo Eduardo Huayanay Quisocala 20193484

Este proyecto se realizó con la finalidad de implementar una red neuronal capaz de usar los conceptos de la física cuántica aplicada a los tableros de ajedrez.

1 Introduccion

1.1 ¿Qué es el ajedrez cuántico?

El ajedrez cuantico es una variante del ajedrez que incorpora conceptos de la mecánica cuántica como la superposición y el entrelazamiento. Las piezas no ocupan posiciones concretas en el tablero, estas pueden existir en diferentes lugares, creando una dinámica compleja para los jugadores involucrados.

1.2 Objetivo del trabajo academico

Se busca desarrollar una Intleigencia capaz de jugar este juego tan particular, que además maneje los conceptos de la mecanica cuantica involucrados para jugar de manera competitiva esta variante tan extraña de ajedrez.

1.3 Aplicaciones

Debido a su manejo de decisiones a nivel cuántico., puede ser útil en areas donde se necesitat tomar decisiones en sistemas cuánticos, además de permitir una comprenisión menos compleja de los algoritmos cúanticos y modelos de Inteligencia Artificial en contextos de alta complejidad.

2 Trabajo Relacionado

2.1 DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess

Este trabajo describe el desarrollo de DeepChess, un modelo de red neuronal profunda que aprende a evaluar posiciones de ajedrez sin conocimiento previo de las reglas del juego. El entrenamiento se realiza utilizando una gran base de datos de partidas de ajedrez, combinando preentrenamiento no supervisado con entrenamiento supervisado para comparar posiciones y elegir la más favorable. DeepChess es relevante para el desarrollo de un modelo de red neuronal para ajedrez cuántico porque demuestra la viabilidad de entrenar un modelo desde cero, sin reglas predefinidas, lo cual es aplicable a la naturaleza compleja y diferente de las reglas del ajedrez cuántico. [David et al., 2016]

2.2 Neural Networks for Chess: The Magic of Deep and Reinforcement Learning Revealed

Este documento proporciona una visión detallada de cómo las redes neuronales, especialmente aquellas basadas en aprendizaje profundo y reforzamiento, han revolucionado los motores de ajedrez, mencionando casos como AlphaZero y Leela Chess Zero que se destacan por su enfoque de autoaprendizaje y búsqueda mediante Monte Carlo. La información sobre AlphaZero y su enfoque de autoaprendizaje puede ser crucial para desarrollar un modelo de ajedrez cuántico, ya que permite entender cómo un sistema puede aprender estrategias complejas por sí mismo, similar a lo que se necesitaría para un juego con reglas cuánticas. [Klein, 2021]

2.3 Monte Carlo Tree Search, Neural Networks & Chess

El documento analiza la combinación de Monte Carlo Tree Search (MCTS) y redes neuronales convolucionales (CNN) para crear un motor de ajedrez que pueda competir contra otros motores como Stockfish. Inspirado en AlphaZero, se utiliza MCTS para explorar posiciones y CNN para evaluar la calidad de las jugadas. Este enfoque es relevante para un modelo de ajedrez cuántico porque MCTS podría ser

una herramienta útil para manejar la complejidad de las posibles configuraciones cuánticas, mientras que las CNN pueden ayudar a evaluar posiciones no tradicionales en el tablero. [Steinberg and Jarnagin, 2021]

2.4 Predicting Chess Moves with Multilayer Perceptron and Limited Lookahead

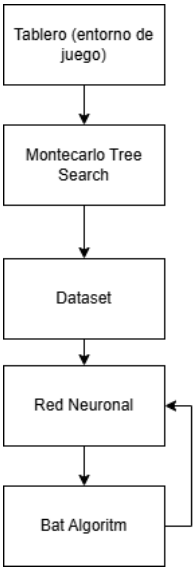
Este estudio explora el uso de una red neuronal perceptrón multicapa para predecir movimientos de ajedrez sin una búsqueda profunda. El enfoque se centra en evaluar el tablero de manera eficiente para reducir la complejidad de las búsquedas. Es útil para un modelo de ajedrez cuántico ya que proporciona una forma de evaluar las posiciones del tablero de manera rápida y eficiente, lo cual es crucial dado el aumento de posibles estados en un sistema cuántico. [Mehta et al., 2020]

2.5 Training a Convolutional Neural Network to Evaluate Chess Positions

Este trabajo investiga la capacidad de una red neuronal convolucional para evaluar posiciones de ajedrez al predecir las evaluaciones del motor Stockfish. Utiliza datos de partidas de ajedrez para entrenar el modelo y demuestra que las CNN pueden captar características complejas de las posiciones en el tablero. La aplicación de CNN para evaluar posiciones es relevante para el ajedrez cuántico, ya que permite el análisis de configuraciones complejas, lo cual es fundamental para entender las dinámicas del juego cuántico. [Vikström, 2019]

3 Metodología

Con el fin de poder jugar de manera correcta, coherente y desafiante al ajedrez cuántico; se requiere el uso de una red neuronal que pueda ser capaz de calcular movimientos a partir de un estado inicial. Dicho estado es un tablero común y corriente con las piezas en él. Para poder llegar a este objetivo se ha planteado el siguiente gráfico que representa el pipeline por el cual se conseguirá la data, para posteriormente usarse en el entrenamiento de la red neuronal, y también tener en cuenta el ajuste de hiper parámetros mediante un algoritmo metaheurístico.



A continuación una breve descripción de cada componente del pipeline.

3.1 Quantum chess

Sobre él se juegan las partidas, tiene implementadas tanto las reglas del juego tradicional, como las reglas de la mecánica cuántica para volver de este un juego más interesante. Además ya contiene el tablero y las piezas necesarias para poder desarrollar todas las simulaciones posibles, y además que pueda ser jugado por una persona normal. Los movimientos especiales con mecánica cuántica que se emplearon son los siguientes:

3.1.1 Movimiento

Para poder trabajar el movimiento convencional que implica cambiar la posición de una pieza entera, o una superposición de esta misma, a otra; se utilizó la compuerta iSWAP, que se debe aplicar entre los dos qubits que intercambiarán valores. Esta compuerta lo que hace es intercambiar el valor de un qubit con otro a la hora de colapsar el circuito completo. Y esto sucede por la matriz que representa a la compuerta matemáticamente.

$$U_{iSwap} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Se utiliza esta porque es más sensible a los patrones de memoria y puede aprender de manera más segura los patrones del circuito y así se evitan swaps inefectivos que dejarían el juego injugable por la aleatoriedad.

3.1.2 Split

Para poder realizar una superposición entre dos posiciones, se debe de mantener el uso de la compuerta previamente analizada, pero esto sería únicamente para uno de los dos destinos que implican el split. El otro destino debe verse evaluado mediante sqrt(iSWAP), entre este y el mismo origen. Esta raíz sobre la compuerta, efectúa una superposición que implica que más adelante cuando se realice el colapso, existirán un 0.5 de probabilidades que aparezca en una posición u otra. Cuantos más splits haya, se irán reduciendo las probabilidades de manera lineal al respecto de cada split o slide realizado.

$$U_{\sqrt{i\text{SWAP}}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} & 0 \\ 0 & \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.1.3 Slide

El slide implica que se realice un movimiento atravesando a una pieza y pasando sobre ella como si no estuviera. Para poder hacer uso de esta regla, la pieza que será obviada, debe de encontrarse en alguna superposición. Y la compuerta que maneja este proceso, es el Controlled-iSWAP, y se tendrá que aplicar como controlador a la casilla que será atravesada, para sí en caso sea cero se tome en cuenta ese movimiento como válido, mientras que se la pieza controladora se mantuvo ahí al colapsar; el movimiento se invalide por completo.

$$U_{\text{slide}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

3.1.4 Split-Slide

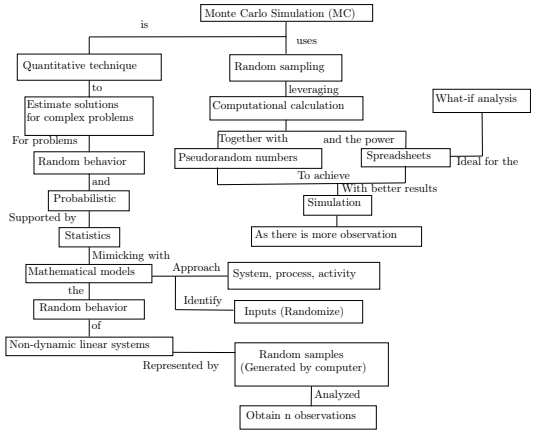
Para poder realizar un split-slide se optó por utilizar una combinación de compuertas. Es decir, se realizará un Controlled-iSWAP entre la posición y su destino con slide. Para luego en caso ya no hay otro slide, se realice un sqrt(iSWAP) hacia la dirección sin mayor problemas. En caso, sea un slide en la misma dirección, se realice un sqrt(iSWAP), pero tomando como inicio el qubit donde se realizó inicialmente el Controlled-iSWAP. Y por último caso, en caso se vaya a don direcciones distintas y ambos requieren slide, a la segunda se le realizará un sqrt(Controlled-iSWAP) desde el mismo qubit de origen.

3.1.5 Merge

Solo es un movimiento que a nivel de lógica permite que se encuentre en esa posición, mas no a nivel matemático, porque luego cuando se realice el colapso, el circuito generado simulará todo y como ese resultado comenzó a moverse junto, no importa de donde llegue, lo que importa es que por un lado, u otro, se mantendrá el camino. Pero la compuerta utilizada será un Controlled-iSWAP para que si un resultado tiene 0, el otro se lo envíe, y si este tiene 0, su contraparte se la envíe.

3.2 Monte Carlo Tree Search

Cumple la función de jugar de manera aceptable el juego. No obstante, se tendrá que limitar tanto su ramificación como su longitud de simulación, debido a la cantidad exorbitante de movimientos posibles a partir de un estado.



El algoritmo de monte carlo utilizará como función objetivo la menor cantidad de movimientos para poder ganar la partida. No obstante, también se hará uso de una heurística, esto porque aparte de guiar el camino para la solución; como muy probablemente el árbol no pueda llegar a simular hasta tener el coste total, entonces la heurística al ser optimista llevará por caminos según su propiedad.
$h1(n) =$ Puntaje del jugador en base al valor de cada pieza.

$c =$ Coste (Cantidad de jugadas hasta el jaque mate)

1. **Admisibilidad**: La heurística resulta admisible porque relaja el problema. Ya no busca realizar jaque mate que puede resultar complejo, comenzará a buscar la mayor puntuación que implica solo comer las más valiosas posibles

2. **Consistencia**: Es consistente, porque conforme se tengan más sucesores, estos siempre resultan más sencillos debido a la cantidad de puntaje reducido.

En base a lo que pueda jugar el algoritmo, se le pondrá como contrincante otro igual para que así simulen partidas; y como el algoritmo buscará el mejor movimiento

posible, entonces de esta manera se nos proveerá de data útil para la red neuronal, cosa que realizando partidas con personas resultaría tanto más costoso, como no necesariamente valioso, porque estas personas pueden cometer errores. Se simularán aproximadamente cien partidas, para a partir de aquí guardar en un dataset tanto entradas como salidas que resultarán de aprendizaje para la red neuronal.

3.3 Dataset

El dataset que se genere a partir de las partidas de Monte Carlo, consistirá en dos vectores. El primero será el tablero de ajedrez que tendrá identificada cada pieza con un número y un signo.

Pieza	ID	Blanco	Negro	Valor
Torre 1	1	1	-1	$1 \times$ Color
Caballo 1	2	1	-1	$2 \times$ Color
Alfil 1	3	1	-1	$3 \times$ Color
Reina	4	1	-1	$4 \times$ Color
Rey	5	1	-1	$5 \times$ Color
Alfil 2	6	1	-1	$6 \times$ Color
Caballo 2	7	1	-1	$7 \times$ Color
Torre 2	8	1	-1	$8 \times$ Color
Peon	[9...16]	1	-1	$i \times$ Color
Vacio	0	0	0	0

Y con estos valores, el tablero que se tenga en base a strings se convertirá en un arreglo numérico basado en el diccionario previamente establecido. Con respecto al output para predecir, se utilizará de igual manera un vector. Este indicará:
v = (Movimiento, Origen 1, Origen 2, Destino 1, Destino 2, Coronación)
Y los valores establecidos para cada uno de los parámetros del vector son:

3.3.1 Movimiento

Movimiento	Valor
Basico	1
Split	2
Swap	3

Para el movimiento no se toma en cuenta el slide, ya que, está implícito dentro de cualquier movimiento, menos el merge.

3.3.2 Orígenes y destinos

Valor	Necesita movimiento	No necesita
Origen 1	[0,63]	-
Origen 2	[0,63]	-1
Destino 1	[0,63]	-
Destino 2	[0,63]	-1

3.3.3 Coronación

Pieza	Valor
Reina	0
Torre	1
Caballo	2
Alfil	3
No hay	-1

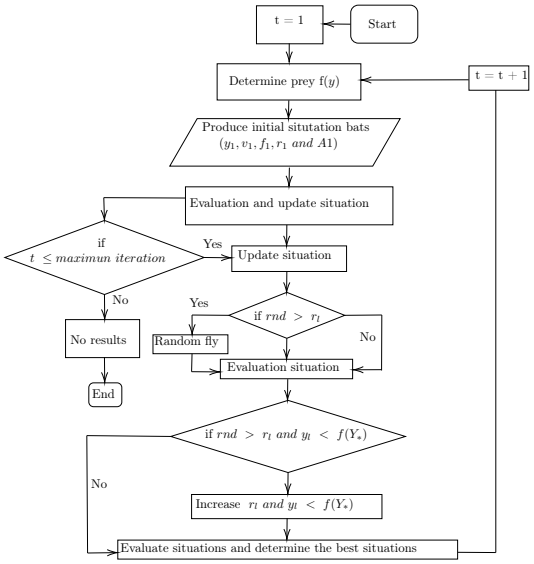
La data no llevará un tratamiento especial, ya que, como es generada por el computador, esta ya está completa y es fiable por el algoritmo que la generó

3.4 Multi Layer Neural Network

Encargado de tomar las desiciones en base de los analisis realizados a nivel del Montecarlo. Es el responsable de ordenar que es lo que se hace mientras avanza la partida de ajedrez que busca ganar el algoritmo.
Tras conseguir de las simulaciones con Monte Carlo data de entrenamiento. Esta se procederá a utilizar como input a la red neuronal.
Para el proceso de entrenamiento se tomará a la data con un 25 % que será para el test, mientras que un 75 % para el train. De esta forma se podrá ver si las jugadas que realizará el modelo tienen lógica con lo que aprendió y si realmente está trabajando bajo la misma lógica de Monte Carlo para llegar a las mismas conclusiones sobre qué movimientos realizar.
La red neuronal multicapa se encargará de mejorar el proceso de aprendizaje, ya que, pasará por capas de neuronas, para así ajustar a cada parámetro de salida necesario de manera más óptima. De igual manera habrá una capa final que ajustará los resultados a la salida que realmente sirve, ya que, lo que se plantea predecir es discreto dentro de rangos fijos según cada componente.
La cantidad de capas o la cantidad de neuronas por capas, y todos los hiper parámetros, se inicializará según la bibliografía estudiada, pero se irán ajustando con el uso del Bat Algorithm.
Finalmente, cuando se compruebe que la red puede predecir movimientos del algoritmo Monte Carlo, entonces se pondrá a prueba en juegos para ver su funcionamiento.

3.5 Bat Algorithm

Si bien debido al dataset generado de manera confiable gracias al Monte Carlo Tree Search, de todas formas, se deben de optimizar los hiper parámetros de las redes neuronales. Para esta optimización se utilizará el bat algorithm.



Se basa en la ecolocalización de los murciélagos para así poder llegar a la solución más óptima, en este caso sería la cantidad de capas, las neuronas por capa, la cantidad de épocas, entre otras.

3.6 Despliegue

El uso de la red neuronal será únicamente sus pesos para poder predecir a partir de un tablero dado, que movimiento realizar. Para esto, se realizará un despliegue dentro de python para que se pueda jugar con un tablero en ASCII; la metodología de juego será por turnos donde empezará el jugador con las blancas, y la red neuronal será las negras.

4 Implicaciones Éticas

1. Seguridad de Datos: Respecto a este punto, no se trabaja con datos personales de ninguna persona. Por el contrario, la data que debe manejarse para el Trabajo Académico es una data que no implica riesgos para la seguridad de nadie.

2. Sesgo: Se priorizan movimientos que puedan acabar en el menor tiempo posible las partidas llevadas a cabo por la IA desarrollada en el documento.

5 Repositorio

Link al repositorio de github link.

6 Declaración y contribución

1. Piero Marcelo Pastor Pacheco: Creación de reglas de juego cuantica y metodología.

2. Fabrizio Gabriel Gómez Buccallo: Implementación del tablero de ajedrez

3. Jean Paul Tomasto Cordova: Busqueda de informacion

4. Jesus Mauricio Huayhua Flores: Redacción de documento

5. Pablo Eduardo Huayanay Quisocala: Implementación del tablero de ajedrez

7 Referencias

Referencias

[David et al., 2016] David, O. E., Netanyahu, N. S., and Wolf, L. (2016). Deepchess: End-to-end deep neural network for automatic learning in chess. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9887 LNCS:88 – 96. Cited by: 49; All Open Access, Green Open Access.

[Klein, 2021] Klein, D. (2021). *Neural Networks For Chess: The Magic of Deep and Reinforcementlearning Revealed*. Amazon Digital Services LLC - Kdp.

[Mehta et al., 2020] Mehta, F., Raipure, H., Shirsat, S., Bhatnagar, S., and Bhovi, B. (2020). Predicting chess moves with multilayer perceptron and limited lookahead. *J Eng Res Appl*, 10(4):05–08.

[Steinberg and Jarnagin, 2021] Steinberg, M. D. and Jarnagin, Z. R. (2021). Monte carlo tree search, neural networks & chess. Master's thesis, Northeastern University, Khoury College of Computer Sciences.

[Vikström, 2019] Vikström, J. (2019). Training a convolutional neural network to evaluate chess positions.