Desenvolupament i Avaluació d'Algorismes d'Escacs: Xarxes Neuronals i MCTS vs. Stockfish i Leela Chess

Oriol Camps Isus

Resum—En aquest treball de fi de grau s'investiguen i comparen diverses metodologies per a desenvolupar algorismes d'escacs eficients. S'implementa una xarxa neuronal capaç d'avaluar posicions d'escacs i s'utilitza, en combinació amb un algorisme de cerca Montecarlo Tree Search (MCTS), per a determinar el millor moviment en cada situació. Aquests models són comparats amb dos dels motors d'escacs més avançats: Stockfish i Leela Chess 0, amb l'objectiu d'avaluar la seva efectivitat i precisió. La comparació es basa en la capacitat de cada algorisme per a prendre decisions òptimes en partides d'escacs, proporcionant així una anàlisi detallada del rendiment de les diferents aproximacions algorítmiques.

Paraules clau—Algorismes d'escacs, Xarxes neuronals, Montecarlo Tree Search (MCTS), Stockfish, Leela Chess 0, Intel·ligència artificial, Models predictius, Optimització de moviments, Aprenentatge automàtic, Avaluació d'algorismes.

Abstract— In this end-of-degree work, various methodologies to develop efficient chess algorithms are investigated and compared. A neural network capable of assessing chess positions is implemented and used, in combination with a Montecarlo Tree Search algorithm (MCTS), to determine the best movement in each situation. These models are compared to two of the most advanced chess engines: Stockfish and Leela Chess 0, with the aim of evaluating its effectiveness and precision. The comparison is based on the ability of each algorithm to make optimal decisions in chess games, thus providing a detailed analysis of the performance of the different algorithmic approa ches.

Index Terms—Chess algorithms, neural networks, Monte Carlo Tree Search (MCTS), Stockfish, Leela Chess 0, Artificial intelligence, predictive models, motion optimization, machine learning, algorithm evaluation.

1 Introducció

En el món dels escacs, la presència de la intel·ligència artificial (IA) ha estat constant i influent al llarg de les últimes dècades [1]. Els escacs representen un repte significatiu per als sistemes d'IA a causa de la seva vasta quantitat de possibles combinacions i la necessitat de prendre decisions precises en un entorn de temps limitat. Cada moviment en una partida d'escacs no només ha de ser tàcticament sòlid, sinó que també ha de contribuir a una estratègia guanyadora global, fet que requereix una comprensió profunda tant de la posició actual com de les possibles posicions futures.

Des dels primers intents d'automatitzar el pensament estratègic fins a les modernes xarxes neuronals profundes, s'ha desenvolupat una gran varietat d'algoritmes per abordar la complexitat inherent dels escacs [2]. Aquests algoritmes varien des de simples heurístiques fins a models avançats d'aprenentatge automàtic que s'entrenen jugant milions de partides contra ells mateixos.

En aquest treball, ens proposem investigar quins dels diferents algoritmes d'IA són més eficaços en la identificació i execució dels millors moviments en una partida d'escacs. Així mateix, explorarem les diferents tècniques que aquests algoritmes utilitzen per analitzar les posicions del tauler i predir les millors jugades, comparant les seves estratègies d'exploració, la seva capacitat de generalització i la eficàcia en diferents situacions tant conegudes com inexplorades [3]. Aquesta anàlisi ens permetrà identificar els avantatges i limitacions de cada enfocament, proporcionant una comprensió més profunda de com la IA pot ser utilitzada per optimitzar el joc d'escacs a nivells que abans eren impensables per als éssers humans.

1.1 Motors d'escacs

Els motors d'escacs moderns són programes informàtics que utilitzen algoritmes avançats per jugar als escacs a un nivell superhumà [4]. Es basen en diferents estratègies per prendre decisions i generar tàctiques durant la partida. En el món dels escacs hi ha moltes possibles propostes de motors de joc que permeten trobar la jugada adequada, en aquest treball ens centrarem en dos, motors basats en cerca i motors basats en xarxes neuronals.

1.1.1 Motors basats en cerca

Aquests motors són programes o sistemes que utilitzen algorismes específics per a explorar de manera eficient l'arbre de possibles moviments i estats del joc. Aquests motors

- *E-mail de contacte: 1558778@uab.cat*
- *Menció realitzada: Computació*
- *Treball tutoritzat per: Xim Cerdà Company (Ciències de la Computació)*
- *Curs 2023/24.*

estan dissenyats per a prendre decisions informades sobre quin és el millor moviment a realitzar en una determinada posició. Actualment, els més utilitzats son MiniMax amb Alpha-Beta Pruning [2].

1.1.2 Motors basats en xarxes neuronals

Aquests són sistemes computacionals que s'inspiren en la forma en què funciona el cervell humà per processar informació i aprendre. Aquestes xarxes estan dissenyades per reconèixer patrons, aprendre de dades i fer prediccions o classificacions a partir de la informació que reben. El motor que utilitza xarxes neuronals més reconegut és Leela Chess 0 (Lc0) [2].

2 Objectius del projecte

Un cop investigats projectes similars i la informació necessària, el treball consisteix en la cerca de l'algoritme més eficaç per a jugar al joc dels escacs de manera autònoma, és a dir, trobar quin dels algoritmes és el millor per a jugar als escacs. Es compararan una sèrie d'algoritmes i es decidirà quin és el millor d'acord amb una sèrie de partides que jugaran entre ells. Alguns dels algoritmes a comparar seran creats per mi i altres seran de codi obert.

En aquest projecte es desenvoluparan una xarxa neuronal que permeti establir un valor a una certa posició d'escacs i l'algoritme MCTS [5] el qual pugui escollir el millor moviment per cada jugada. Aquests dos algoritmes s'avaluaran per separat, però també es combinaran per tal de millorar les seves característiques. A més seran comparats amb algoritmes ja creats i de codi obert per tal de comprovar la seva eficàcia, en aquest treball es compararan amb els motors Stockfish, el motor més popular i complet actualment, i amb Lc0, la qual utilitza xarxes neuronals i aprenentatge per reforç.

Espero que la combinació entre xarxes neuronals i MCTS sigui una de les opcions més prometedores, ja que les xarxes neuronals són molt precises avaluant posicions i generant candidats a jugades, mentre que MCTS és molt eficient recorrent l'arbre de joc i trobant la millor solució. De fet, actualment és una de les combinacions més utilitzades en els motors d'escacs.

De manera alternativa, és possible que els algoritmes de forma individual siguin més eficaços a l'hora d'avaluar i anticipar les possibles jugades.

3 Metodologia

Per al desenvolupament del projecte hi haurà diverses fases, la primera és la fase de disseny en la que es definiran l'arquitectura i se seleccionaran les eines que s'utilitzaran, a més de buscar les dades d'entrenament, la segona fase és de desenvolupament i entrenament dels diferents algoritmes i la seva posterior unió, la tercera fase consisteix a testejar el funcionament dels models desenvolupats i la posterior avaluació de rendiment, també s'optimitzarà al màxim el rendiment de cada model, i finalment la quarta fase consisteix en la comparació i avaluació dels motors creats.

- Fase 1: Disseny i arquitectura
  - Definir l'arquitectura del motor.
  - Seleccionar les eines de desenvolupament.
  - Recopilar dades d'entrenament.
- Fase 2: Entrenament i desenvolupament
  - Entrenar la xarxa neuronal.
  - Implementar l'algorisme MCTS.
  - Integrar la xarxa neuronal i l'algorisme MCTS.
- Fase 3: Avaluació
  - Avaluar el rendiment del motor d'escacs.
  - Identificar i corregir errors.
  - Optimitzar el rendiment.
- Fase 4: Comparació
  - Avaluar el funcionament de tots els motors d'escacs.
  - Competició de motors d'escacs.

3.1 Disseny i arquitectura

La primera part de qualsevol projecte és la part de disseny, en aquest cas el que es dissenya és una xarxa neuronal i la utilització de diferents algoritmes de machine learning per tal d'aconseguir l'objectiu desitjat.

Per tal de dissenyar el model de xarxa neuronal que utilitzaré el primer que es necessita és una gran quantitat de dades que et permetin entrenar al model, en aquest cas, després de cercar diferents bases de dades que et permetin avaluar una situació donada en el taulell vaig trobar una que tenia una gran quantitat de dades i s'ajustava a les necessitats del projecte. Aquest dataset consistia en dues columnes, una columna amb la posició del taulell escrita en format FEN, el qual escriu la posició de les peces que hi ha en el taulell en ordre, de dreta a esquerra i de dalt a baix, i la seva avaluació en centipawn, aquesta mesura és la que es fa servir de manera estàndard per a avaluar les jugades. A aquest conjunt de dades les he separat en un conjunt de test i un altre de train, per tal d'avaluar el model en cada iteració.

Un cop tenim les dades necessàries per a entrenar el model s'ha de desenvolupar un disseny de xarxa neuronal que sigui funcional per a aquestes dades i que pugui assolir l'objectiu previst. El primer pass és saber els inputs i outputs que tindrà la xarxa neuronal, en el meu cas, el model tindrà dues entrades, un array de 64 posicions que serà el taulell complet de la jugada i un array de 6 elements que seran dades del moment de la partida, com per exemple, si hi ha jaque o si es pot enrocar, l'output serà un enter que representa la puntuació donada a la jugada. En aquest punt hi ha una gran quantitat de possibles combinacions pel que fa a les capes internes, ja que una xarxa neuronal està composta de capes de neurones, i cada capa pot tenir un nombre qualsevol de neurones, la forma d'aquestes capes i com interactuen entre si pot fer que el model sigui més o menys eficaç. Donat aquest marc de possibles solucions el que he fet és utilitzar la prova i error, he creat un model simple i l'he entrenat amb el conjunt de dades de train, per a posteriorment provar-lo amb el conjunt de test.

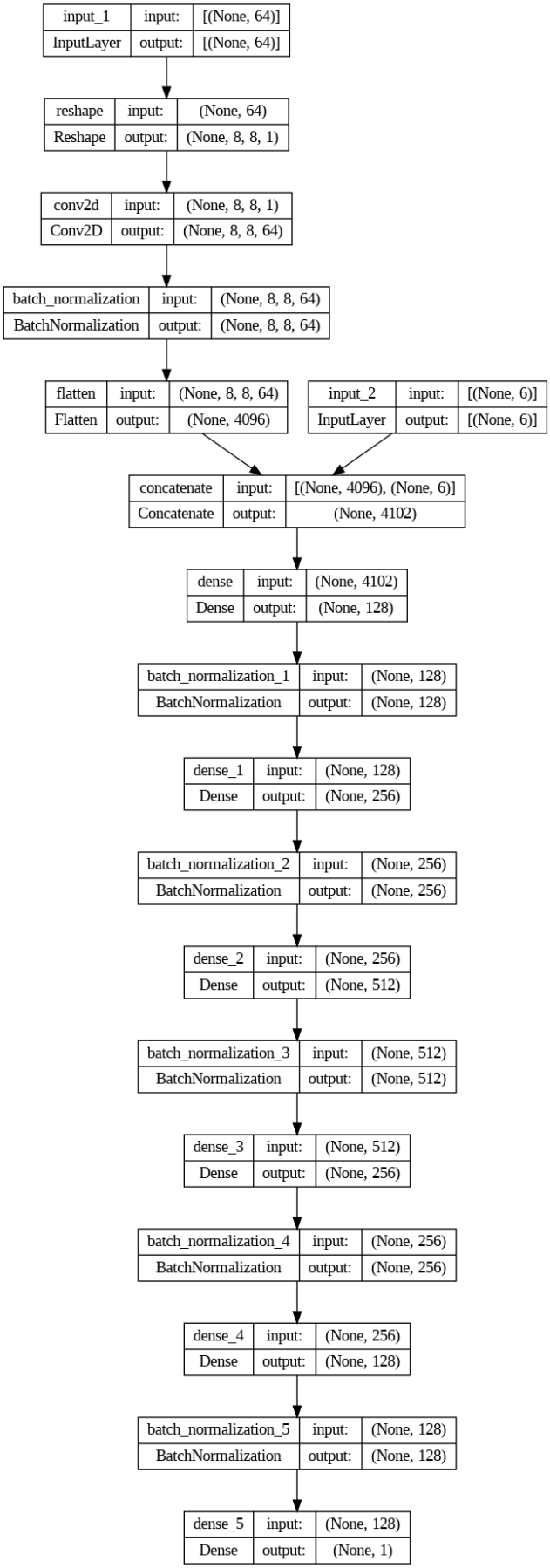

Figura 1: *Representació gràfica de la xarxa neuronal.*

El model que finalment he utilitzat (Figura 1) es tracta d'un conjunt de capes serialitzades, és a dir, la sortida d'una capa és l'entrada de la següent.

Primerament, s'introdueixen les dades del taulell que passen per un conjunt de capes que apliquen un filtre convolucional, aquestes capes permeten a la xarxa neuronal detectar patrons en la formació de les peces del taulell, està composta per una capa que transforma les dades d'entrada d'un array a una matriu (capa *'Reshape'*) per tal que, just a continuació, s'apliqui una capa convolucional (capa *'Conv2D'*) i finalment es normalitzin els resultats (capa *'BatchNormalization'*) i tornin a la forma original (capa *'Flatten'*). Un cop aplicat aquest filtre, s'ajunten amb la resta d'entrades de la xarxa neuronal (capa *'Concatenate'*).

Finalment, utilitzo una estructura de capes que consisteix en una capa densa (capa *'Dense'*), aquesta genera una connexió entre totes les neurones prèvies i les neurones de la capa i un altre que normalitza la sortida de les neurones (capa *'BatchNormalization'*). Aquesta formació de capes permet a la xarxa entendre el conjunt de dades i relacionar certs aspectes de la posició del taulell amb l'avaluació d'aquesta.

Tot seguit he dissenyat l'algoritme de MCTS, en aquest cas hi ha diferents variacions d'aquest algoritme, he considerat utilitzar alguns com UCT-MCTS, que utilitza una sèrie de ponderacions a cada node per a d'afavorir el desenvolupament de nodes sense explorar. Finalment, he decidit fer servir una combinació de l'algoritme MCTS i Min-Max amb poda alpha-beta, ja que aquesta combinació és la més eficient i de les millors opcions en jocs amb informació completa.

Aquesta combinació consisteix a usar l'algoritme MCTS en l'exploració inicial i un cop seleccionats els nodes més prometedors, aplicar Min-Max per aprofundir en una avaluació més precisa, és a dir, l'algoritme MCTS es centra en l'exploració de noves jugades i moviments més prometedors mentre que Min-Max selecciona les millors jugades possibles entre totes les anteriors.

Un cop fets els dos dissenys he de provar com combinar-los per tal de millorar el seu rendiment, ja que, la xarxa neuronal serveix per a interpretar de forma eficient el valor d'una posició, i l'algoritme et permet decidir quina de totes és la millor. El que he plantejat és utilitzar la xarxa neuronal com a part de l'algoritme MCTS, així es pot aprofitar la velocitat i precisió de la xarxa neuronal per a esbrinar la puntuació de cada estat de l'arbre.

3.2 Entrenament i desenvolupament

Tenint clar el disseny dels diferents algoritmes el següent pas és el desenvolupament d'aquests, en aquest cas el primer que vaig dur a terme va ser l'entrenament de la xarxa neuronal, ja que, tenint en compte que els temps d'entrenament són molt elevats podia desenvolupar l'algoritme MCTS en aquests períodes d'entrenament.

3.2.1 Entrenament de la xarxa neuronal

En el moment d'entrenar una xarxa neuronal és important definir les mètriques d'entrenament, en aquest cas ens centrarem en 3 diferents, una mètrica general, la funció optimitzadora i la funció de pèrdua.

La mètrica general s'utilitza per a avaluar el rendiment de la xarxa neuronal durant el procés d'entrenament, en el meu cas utilitzo la funció d'error absolut mitjà (MAE) [3] aquesta calcula la diferència entre les prediccions i el valor real. Per altra banda, la funció d'optimització que utilitzo és la funció Adam [6], la qual permet ajustar els paràmetres de la xarxa neuronal seguint l'error del gradient [7], el qual mesura quant ha de canviar el gradient per tal de millorar la predicció, aquest canvi depèn d'un paràmetre que he ajustat a 0.01, ja que és el que millor funciona per l'estructura de la xarxa neuronal. Finalment, la funció de pèrdua, que s'utilitza per a mesurar la diferència entre el valor real i la predicció del model, utilitza aquest resultat per a fer la retropropagació i ajustar els paràmetres de les neurones, en el meu cas he utilitzat la funció d'error quadrat mitjà (MSE) [3] la qual és molt similar al MAE però utilitzant el quadrat de la diferència.

Utilitzant les mètriques anteriors he entrenat la xarxa neuronal, per a fer-ho es necessita una gran quantitat de recursos computacionals, ja que és un procés costós per als processadors, és per això que he utilitzat Google Colab el qual em permet utilitzar els recursos de Google per a executar el codi que entreni a la xarxa neuronal.

Pel que fa a l'entrenament he utilitzat un subset de dades de validació després de cada època per tal de verificar si hi havia overfitting en el model. Després de cada sessió he tret dues gràfiques, una que mostra l'evolució de la funció Loss i un altre del MAE en cada època.

3.2.2 Desenvolupament de l'algoritme MCTS

Per altra banda, he desenvolupat l'algoritme MCTS, com he especificat anteriorment, aquest és una combinació de dos algoritmes, el primer és MCTS i el segon és Min-Max, és per això que primer s'han de desenvolupar per separat per tal de, posteriorment, fusionar-los. També utilitzaré la xarxa neuronal que he desenvolupat per a avaluar cada node de l'arbre.

El MCTS consta de dues classes, la primera que gestiona totes les fases del MCTS, selecció, expansió, simulació, retropropagació i selecció de moviment, a més a més, és on es guarda el model de la xarxa neuronal i un comptador d'iteracions, aquestes permeten definir quantes vegades s'executaran totes les fases abans de definir un moviment, com més iteracions més precisió però més temps d'execució. Una segona classe que emmagatzema tota la informació de cada node i el funcionament de cada un per tal de crear un arbre de joc complet. Cada node conte l'estat del joc que representa, el pare d'aquest node, un diccionari amb tots els fills, on cada fill està representat amb un moviment que canvia entre estats, també conté el nombre de vegades que ha estat visitat, el valor actual d'aquest node, la quantitat de vegades que algun dels fills guanya la partida i finalment una llista amb tots els fills que no han estat visitats.

Posteriorment, he creat la funció que executa Min-Max, permet avaluar quin és el resultat a partir d'un node ja explorat pel MCTS, posteriorment vaig implementar aquesta funció dins la classe MCTS per tal d'utilitzar-ho en l'execució de l'algoritme.

Per tal d'integrar la xarxa neuronal en l'algoritme el que he fet és avaluar l'estat de la jugada de la fulla i utilitzar aquest valor per a fer la retropropagació fins al node arrel i així actualitzar tots els valors possibles de l'arbre, això em permet avaluar de forma ràpida i precisa cada node de l'arbre, ja que si no el que generalment es fa és intentar crear una heurística segons diferents factors de l'estat actual, aquest procés generalment és molt més costós que si s'utilitza la xarxa neuronal.

3.3 Avaluació de model

Una vegada desenvolupats els diferents motors s'han d'identificar els possibles errors i corregir-los com més aviat millor per tal de poder avaluar el seu rendiment i optimitzar el funcionament i els resultats.

3.3.1 Identificació i correcció d'errors

El primer que vaig fer va ser una prova d'entrenament amb el model inicial (Figura 2), vaig entrenar-lo utilitzant un subset de docents mil dades i un total de 50 èpoques, cada època consta de quatre mil iteracions.

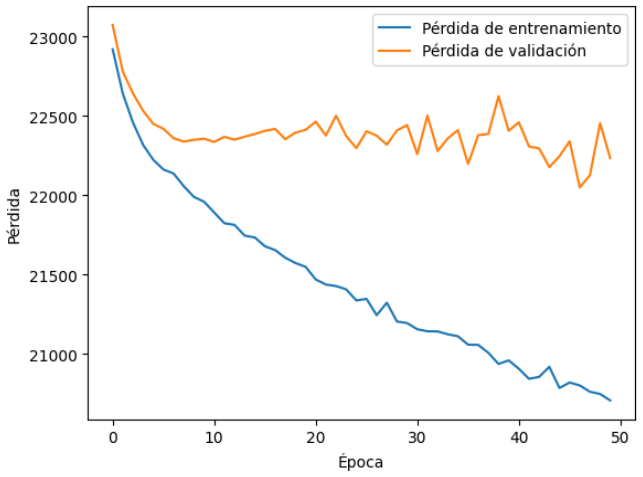

***Figura 2**: Gràfica d'entrenament, en blau les dades d'entrenament i en taronja les dades de validació amb 200 mil dades, 50 èpoques i 0.3 dropout*

Com es pot veure en la gràfica (Figura 2), els resultats de la validació no milloren igual que els de test, el qual indica que hi ha un sobreentrenament (overfitting en anglès) i que l'entrenament no està sent efectiu. Una possible solució per aquest problema és augmentar el drop-out per tal que les neurones no s'ajustin tant a les dades d'entrenament i puguin globalitzar una mica millor els restats.

En aquest cas he augmentat el drop-out a 0.5, per tant, la meitat de les neurones es desactivaran després de cada època i així evitar l'overfitting, amb aquest canvi he tornat a entrenar la xarxa neuronal amb les mateixes condició que en el cas anterior.

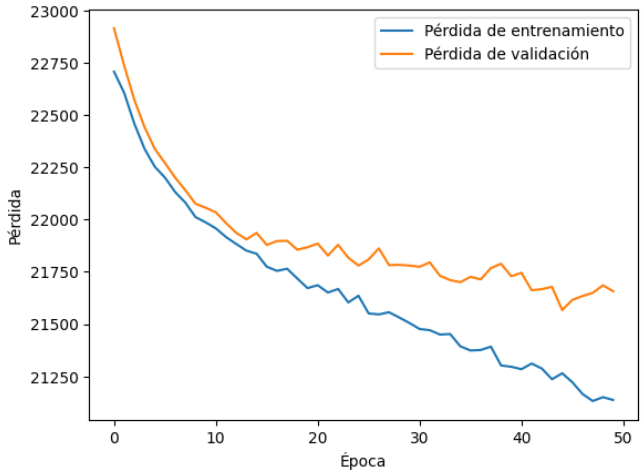

***Figura 3**: Gràfica d'entrenament, en blau les dades d'entrenament i en taronja les dades de validació amb 200 mil dades, 50 èpoques i 0.5 dropout*

La gràfica (Figura 3) ens mostra que el problema de l'overfitting s'ha mitigat, tot i que es pot veure com al final de l'entrenament es torna a separar la gràfica, és per això que he augmentat la mida del subset d'entrenament i així augmentar la diversitat de les dades i que el model pugui generalitzar de forma més precisa, l'he augmentat fins a 400 mil i també he augmentat el nombre d'èpoques de l'entrenament, això em permetrà veure com evoluciona el model i si pot millorar el seu rendiment.

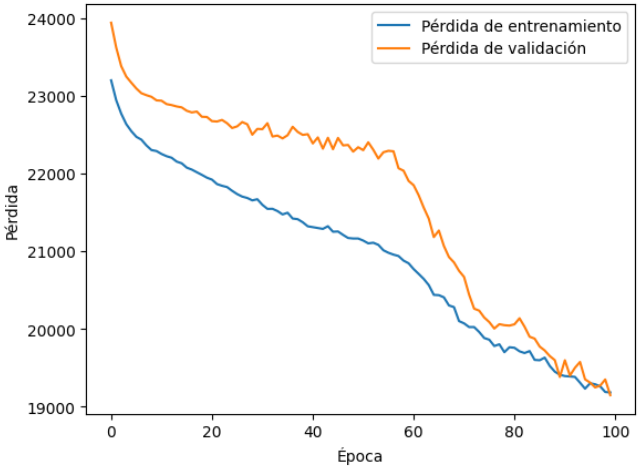

***Figura 4:** Gràfica d'entrenament, en blau les dades d'entrenament i en taronja les dades de validació amb 400 mil dades, 100 èpoques i 0.5 dropout*

Aquests resultats (Figura 4) mostren que l'overfitting millora amb cada època, veiem com al final de l'entrenament els resultats de la validació son molt similars als de l'entrenament, per tant, està sent efectiu, tot i que el descens de la corba d'entrenament és molt pronunciat a les èpoques finals, aquest fet podria indicar que el model té marge de millora quant a resultats globals.

Per tal de comprovar aquesta possibilitat de millora vaig augmentar les èpoques a 200 per a assegurar-me que l'entrenament hagués arribat al límit.

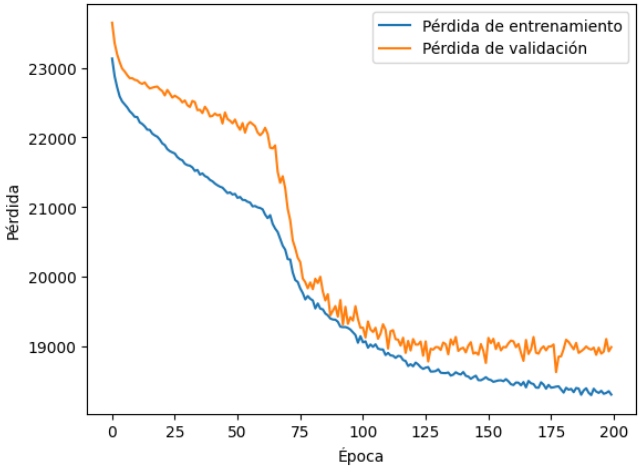

***Figura 5:** Gràfica d'entrenament, en blau les dades d'entrenament i en taronja les dades de validació amb 400 mil dades, 200 èpoques i 0.5 dropout*

El resultat de la gràfica (Figura 5) ha sigut l'esperat, en aquest cas l'entrenament és efectiu per unes 25 èpoques més, a partir d'aquest punt els resultats de validació ja no milloren, tot i que els de l'entrenament si, igual que en casos anteriors, podem veure overfitting al final.

Un cop la xarxa neuronal ha sigut entrenada, vaig comprovar el funcionament de l'algoritme MCTS juntament amb Min-Max. El primer que vaig comprovar va ser el temps d'execució de cada fase del MCTS, selecció, expansió, simulació, retropropagació i selecció de moviment, a més a més de la funció de l'algoritme Min-Max. En la primera execució vaig veure que Min-Max tenia un temps d'execució de més d'un minut, això és degut a la gran quantitat de moviments que es poden fer en els escacs, per la qual cosa ha d'iterar moltes vegades per a comprovar tots els moviments.

Per tal d'arreglar aquest problema vaig fer una cerca de possibles millores per Min-Max, la primera que vaig implementar va ser utilitzar una poda alfa beta més agressiva, és a dir, assegurar-me que una branca es poda l'abans possible. Un altre millora que he implementat ha sigut la Transposition Table la qual permet a l'algoritme avaluar posicions prèviament conegudes i així millorar el seu rendiment. L'última millora que vaig incorporar va ser l'ordenació de moviments segons si el moviment captura una peça o no, ja que aquests moviments generen grans canvis en l'avaluació de la posició per el que la poda Alfa-Beta és més efectiva. Amb aquestes millores he aconseguit un temps mitjà de mig minut, és a dir, he reduït el temps a la meitat.

3.3.2 Avaluació de models

Un cop corregits els problemes d'eficiència toca avaluar els models per separat i així veure que tan bé compleixen el seu objectiu.

Pel que fa a la xarxa neuronal l'he avaluat utilitzant les dades de test, he calculat l'Error Mitjà Absolut (MAE, sigles en angles) [3] i l'Error Mitjà Quadràtic (MSE, sigles en angles) [3]. Aquestes mesures mostren que tan lluny està el resultat del resultat esperat, és a dir, com més gran sigui l'error més lluny està el valor predit del valor esperat. Si els

calculem comparant valors predits pel model i els valors reals el resultat és: MAE, 58.2994 i MSE, 18985.7971. Tots dos valors són elevats, tot i que, tenint en compte que el que es calcula està en centipawn l'error no és tan significatiu, és a dir, el MAE és de mig peó pel que fa a l'avaluació de la posició, per tant, l'error no és tan gran com sembla.

Pel que fa al MCTS he avaluat el seu rendiment en un àmbit diferent, ja que aquest no retorna valors numèrics sinó que escull la millor jugada segons el seu criteri. És per això que he provat el seu rendiment competint en una partida real. He utilitzat al model Stockfish, perquè aquest em permet limitar el nivell al qual juga, és a dir, el puc fer més o menys precís segons uns paràmetres. També he disputat partides contra humans de diferents nivells, des de persones que no saben com es juga i persones que tenen un cert nivell d'escacs.

Quant a les partides contra Stockfish, he utilitzat tres variants, la primera on el motor està limitat al màxim, un punt entremig i un altre on el motor no està limitat. Contra cada variant ha jugat cinc partides, on el color de cada motor és aleatori.

Taula 1. Guanyadors de les partides. La taula mostra el guanyador de la partida en cada cas

Limitació Stockfish	Partida 1	Partida 2	Partida 3	Partida 4	Partida 5
100%	Stockfish	MCTS	MCTS	MCTS	MCTS
50%	Stockfish	Stockfish	Stockfish	Stockfish	Stockfish
0%	Stockfish	Stockfish	Stockfish	Stockfish	Stockfish

Com es pot veure en els resultats (Taula 1), el model MCTS és capaç de guanyar al model Stockfish quan aquest està altament limitat, això mostra que MCTS és capaç de crear estratègies guanyadores, tot i que aquestes estratègies no són molt robustes, ja que no és capaç de guanyar si es redueix la limitació.

Taula 2. Resultats de les partides. La taula mostra el resultat de les partides entre humans i l'algoritme

Nivell del jugador	Baix	Intermedi	Avançat
MCTS	2-1	0-3	0-3

Quant a les partides contra humans (Taula 2), el motor ha guanyat als jugadors completament inexperts, perquè, igual que contra Stockfish, és capaç de crear estratègies mínimes i capturar les peces importants. Quan s'enfronta a jugadors que sí que juguen als escacs o tenen una certa experiència, el motor es queda enrere pel que fa a rendiment, perquè les estratègies formades pel motor no són altament robustes i fàcilment es poden evitar per part dels jugadors amb més experiència.

3.4 Comparació entre models

En aquest apartat es comparà l'eficàcia dels models mitjançant un procés de competició on cada model compatirà contra tots els altres models per tal de veure quin d'ells és el millor.

3.4.1 MCTS vs Xarxa neuronal

El primer a provar és l'eficàcia entre els dos models creats per mi, ja que cada un utilitza diferents estratègies per a trobar la millor jugada. Per provar quin dels dos models és millor han jugat un total de 51 partides de les quals MCTS ha guanyat totes les partides a la Xarxa neuronal. Aquesta clara avantatge del model MCTS és deguda a la capacitat d'exploració de camins de joc que permet al model trobar i crear millors estratègies a futur, ja que, gràcies al Min-Max, és capaç de preveure una alta quantitat de jugades i avaluar de forma precisa quin és el millor moviment en cada cas i com contestar als moviments de l'oponent.

3.4.2 Procés de competició

Per tal d'avaluar les diferències entre models cada parella jugarà un total de nou partides diferents, on, en cada partida, els colors s'assignen de manera aleatòria. Finalment, es considerarà millor al model que guanyi més partides de les nou.

Taula 3. Resultats de les competicions. La taula mostra el resultat de la competició en cada cas, el valor de l'esquerra correspon al model de la fila i el valor de la dreta al de la columna

	Xarxa Neuronal	**Stockfish**	**Leela Chess 0**
MCTS	9-0	0-9	0-9
	Xarxa Neuronal	0-9	0-9
		Stockfish	0-9

Els resultats (Taula 3) demostren que els models Stockfish i Leela Chess 0 ofereixen un rendiment superior gràcies a la seva optimització avançada i a l'extens ús de recursos computacionals per a la seva millora contínua. En canvi, els models que he desenvolupat, MCTS i Xarxa Neuronal, presenten un rendiment inferior. Aquesta diferència es deu principalment a la limitació de recursos disponibles i al temps restringit per a la seva optimització i entrenament, factors que han impedit aconseguir el nivell de sofisticació dels altres models. És important remarcar que l'algoritme MCTS és superior a la Xarxa Neuronal, aquesta diferència es deu a la complexitat de l'algoritme MCTS, ja que, aquest és capaç de simular jugades futures i preveure quins moviments es jugaran en els següents torns.

4. Conclusió

En aquest treball, s'han desenvolupat dos algorismes per jugar als escacs: un basat en una xarxa neuronal per avaluar posicions i un altre que combina l'algoritme Monte Carlo Tree Search (MCTS) amb Min-Max per seleccionar la millor jugada. En les 51 partides disputades entre aquests dos algorismes, MCTS ha demostrat una clara superioritat, guanyant gairebé totes les partides. Aquest resultat es pot atribuir a la capacitat de MCTS per explorar múltiples camins de joc de manera més profunda i exhaustiva que la xarxa neuronal, la qual es veu limitada per la seva capacitat d'aprenentatge i la qualitat de les dades d'entrenament

disponibles.

Comparant aquests algorismes amb motors d'escacs d'alt nivell com Stockfish i Leela Chess 0, es va evidenciar una gran diferència en rendiment. Aquests motors professionals, desenvolupats amb recursos significatius i tècniques avançades, superen àmpliament els algorismes propis. Això s'explica per la seva capacitat d'explorar el joc amb una precisió molt més gran gràcies a una infraestructura computacional robusta i accés a grans volums de dades de qualitat. Els resultats reflecteixen la necessitat d'optimització contínua i l'ús de dades extenses per competir a nivells més alts, proporcionant una base sòlida per a futures investigacions i millores en la intel·ligència artificial aplicada als escacs.

BIBLIOGRAFIA

[1] T. Koenig, «Tracking the evolution of artificial intelligence through the lens of its ability to play chess,» 19 04 2024. [En línia]. Available: https://repository.yu.edu/server/api/core/bitstreams/ba13a066-3e45-4728-80f4-9793fcc127e0/content. [Últim accés: 21 05 2024].

[2] G. Haworth i N. Hernandez, «The 20th Top Chess Engine Championship, TCEC20,» J. Int. Comput. Games Assoc., Maryland, USA, 2021.

[3] P. Martí Sanahuja, «Métricas de evaluación de rendimiento para predicciones de series temporales,» 05 01 2021. [En línia]. Available: https://polmartisanahuja.com/metricas-de-evaluacion-de-rendimiento-para-predicciones-de-series-temporales/. [Últim accés: 15 03 2024].

[4] L. F. Siles, «5 inteligencias artificiales y motores de ajedrez que arrasan,» 02 10 2020. [En línia]. Available: https://www.chess.com/es/article/view/inteligencia-artificial-ajedrez. [Últim accés: 15 03 2024].

[5] B. Nasarre Embid, «Método de Monte-Carlo Tree Search (MCTS) para resolver problemas de alta complejidad.,» Universidad de Zaragoza, 06 2012. [En línia]. Available: https://zaguan.unizar.es/record/8010/files/TAZ-PFC-2012-393.pdf. [Últim accés: 15 03 2024].

[6] J. Schäfer, «Estimación adaptativa del momento: entender a Adam y utilizarlo correctamente,» 20 12 2023. [En línia]. Available: https://konfuzio.com/es/estimacion-adaptativa-de-momentos/. [Últim accés: 16 03 2024].

[7] N. B. A. Moreno Cabañas, «Universidad de Chile - Facultad de Ciencias Físicas y Matemáticas,» 2023. [En línia]. Available: https://repositorio.uchile.cl/handle/2250/197408. [Últim accés: 13 05 2024].

[8] E. Mayefsky, . A. Francine i M. Sirota, «Strategies and tactics for intelligent search,» 2003. [En línia]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/credits.html. [Últim accés: 13 03 2024].

[9] L. Kocsis i C. Szepesvári, Bandit Based Monte-Carlo Planning, Budapest, Hungary: Computer and Automation Research Institute of the Hungarian Academy of Sciences, 2006.