Sergi Alberich Velasco

ESTUDI DE L'EXPLICABILITAT DELS MOVIMENTS DEL JOC D'ESCACS A TRAVÉS DE XARXES NEURONALS

TREBALL DE FI DE GRAU

Codirigit pel Dr. Agustí Solanas Gómez
i el Dr. Josep Maria Banús Alsina

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona
2023

Resum.

El següent treball tracta sobre la creació d'un algorisme basat en xarxes neuronals per a l'explicació dels moviments en el joc dels escacs. Més en concret, es busca poder crear una nova eina que sigui utilitzada per a que els aprenents del joc pugui entendre com un moviment afecta la posició i perquè es considera que aquest ha estat bo o dolent.

Per a dur a terme aquesta tasca, es busquen partides d'escacs per a poder entrenar la xarxa, s'extreuen les dades necessàries d'aquesta i es crea una base de dades. També, es dissenya una codificació per a traduir les partides i poder-les usar com a entrada d'una xarxa i es dissenya i entrena una xarxa neuronal exitosament per a poder predir, a partir de la posició inicial i final d'un moviment, la valoració i algunes característiques d'aquest. Posteriorment, es dissenyen, entrenen i verifiquen més xarxes per a poder interpretar nova informació de les posicions.

Finalment, es crea un algorisme capaç de generar correctament una explicació del moviment i del perquè de la valoració. Tot això es mostra amb la creació d'una GUI per a poder jugar partides i analitzar moviments.

Tot aquest treball ha estat fet en Python, utilitzant les llibreries que aquest llenguatge ofereix per a poder dur a terme tot el descrit anteriorment.

Resumen.

El siguiente trabajo trata sobre la creación de un algoritmo basado en redes neuronales para la explicación de los movimientos en el juego del ajedrez. Más en concreto, se busca poder crear una nueva herramienta que sea utilizada para que los aprendices del juego puedan entender como un movimiento afecta la posición y porque se considera que este ha sido bueno o malo.

Para llevar a cabo esta tarea, se buscan partidas de ajedrez para poder entrenar la red, se extraen los datos necesarios de esta y se crea una base de datos. También, se diseña una codificación para traducir las partidas y poderlas usar como entrada de una red y se diseña y entrena una red neuronal exitosamente para poder predecir, a partir de la posición inicial y final de un movimiento, la valoración y algunas características de este. Posteriormente, se diseñan, entrenan y verifican más redes para poder interpretar nueva información de las posiciones.

Finalmente, se crea un algoritmo capaz de generar correctamente una explicación del movimiento y del porqué de la valoración. Todo esto se muestra con la creación de una GUI para poder jugar partidas y analizar movimientos.

Todo este trabajo ha sido realizado en Python, utilizando las librerías que este lenguaje ofrece para poder llevar a cabo todo lo descrito anteriormente.

Abstract.

The following project deals with the creation of an algorithm based on neural networks for the explanation of moves in the game of chess. More specifically, it seeks to be a new tool used so that the learners of the game can understand how a movement affects the position and why it is considered to be good or bad.

To carry out this task, chess games are searched in order to train the network, the necessary data is extracted from them and a database is created. Also, a coding is designed to translate the matches and be able to use them as input of the network and a neural network is designed and trained successfully to be able to predict, from the initial and final position of a movement, the evaluation and some characteristics of it. Subsequently, more networks are designed, trained and verified to be able to interpret new position information.

Finally, an algorithm capable of correctly generating an explanation of the movement and the reason for the assessment is created. All this is shown with the creation of a GUI to be able to play games and analyse moves.

All this work has been done in Python, using the libraries that this language offers to carry out everything described a

Índex

Índex de taules

Índex d'il·lustracions

1. Introducció

1.1 Context i motivació

Aquest treball, com es pot veure pel títol, tracta sobre els escacs. Tot i que no sigui algú que competeix en aquest esport, sempre m'ha agradat i he gaudit molt jugant. Des de ben petit quan la meva mare em va introduir i fèiem partides com a passatemps ja em va sembla molt interessant, però mai vaig arribar més enllà del que creia que era, un joc.

No va ser fins que ja vaig tenir una edat, que vaig tornar a reenganxar-me al joc i apassionar-me per aprendre més i ser millor jugador, portant-me això a descobrir el món dels escacs i tot el que aquest taulell amb 64 requadres i 32 peces pot oferir. Des de les obertures, on es marca l'inici de la partida i es defineix com es vol jugar aquesta, passant pel mig-joc, on es comença a executar un pla i veure cap a on es vol portar el joc i, finalment, els finals de partida, on s'acaba d'esprémer aquesta avantatja que s'ha anat acumulant al llarg de la partida per a poder acabar vencent al rival.

Va ser en aquest camí d'aprendre on vaig descobrir la importància dels ordinadors i la computació en els escacs moderns i com aquests ajuden des dels jugadors més principiants a veure com han de fer millors moviments, fins als més experts a trobar el moviment òptim en cada posició. Però tothom que hagi provat qualsevol d'aquests motors d'escacs s'adonarà d'una cosa molt important, es pot veure ràpidament quin és el moviment més bo, però això no servirà de res si no se sap el perquè.

És en aquest perquè de cada moviment on tot jugador d'escacs passa la seva carrera intentant aprendre. Perquè certs moviments són millors que altres o perquè hi ha moviments que tot i semblar antinaturals són els millors. Això és el que, sent estudiant d'informàtica em semblava més curiós, com pot ser que un ordinador sigui capaç de saber el millor moviment veient moltes jugades en el futur, però no pugui dir per què aquesta jugada és la bona. I és això, aquesta voluntat de voler saber més del joc i fer que sigui més accessible per a tothom que m'ha portat a la realització d'aquest treball.

1.2 Objectius

Els objectius d'aquest treball és aconseguir fer un primer pas en la possibilitat d'explicar perquè un moviment d'escacs és bo o dolent. Per a fer això, cal comprendre els fonaments teòrics de les xarxes neuronals artificials i modificar-los per a aplicar-los en l'anàlisi de moviments d'escacs. També he d'adquirir nous coneixements en el llenguatge de programació Python, específicament en l'ús de biblioteques com TensorFlow o PyTorch per a implementar xarxes neuronals. Cal investigar i seleccionar les tècniques i algorismes més adequats per a l'anàlisi i avaluació de moviments d'escacs i recopilar i preparar un conjunt de dades adequat per a entrenar i validar la xarxa, incloent-hi exemples de moviments de tot tipus. L'objectiu final és doncs implementar una xarxa neuronal en Python que sigui capaç d'analitzar i classificar els moviments d'escacs com a bons o dolents en funció dels criteris establerts amb un bon rendiment, entenent aquest rendiment com el percentatge d'encert. A més, desenvolupar una interfície gràfica d'usuari (GUI) intuïtiva i atractiva que permeti als usuaris interactuar amb la xarxa neuronal, ingressar moviments i obtenir una explicació del per què es considera bo o dolent.

També caldrà documentar adequadament tot el procés de desenvolupament, incloent-hi el disseny de la xarxa neuronal, la metodologia d'entrenament, els resultats obtinguts i les

conclusions aconseguides. En resum, millorar les habilitats de recerca, resolució de problemes, programació i presentació d'informes tècnics, aplicant els coneixements adquirits a un projecte pràctic i desafiador en el camp de la intel·ligència artificial aplicada als escacs.

2. Fonaments teòrics

2.1 Escacs: regles i conceptes bàsics

Els escacs són un joc de taula en el qual s'enfronten dos jugadors, cadascun dels quals té 16 peces de valors diversos que pot moure, segons unes certes regles, sobre un tauler dividit en 64 quadres alternativament blancs i negres; sempre comença el jugador que juga amb les peces blanques i guanya el que aconsegueix donar mat al rei del seu contrincant.

La disposició inicial de les peces és fixa. La segona fila és ocupada per 8 peons. Les torres se situen a les cantonades, mentre que els dos cavalls es posicionen a cada costat d'aquestes, seguits pels alfils. Al costat d'aquests últims, la dama o reina sempre ocupa la casella del seu respectiu color (la dama blanca en casella blanca, la negra en casella negra). Finalment, el rei es col·loca al costat de la reina, en l'última casella disponible.

Il·lustració 1. Disposició inicial de les peces d'escacs

Un cop tenim les peces col·locades caldrà saber com es mou cadascuna d'aquestes. Com ja he esmentat, 6 tipus de peces, cadascuna amb el seu propi patró de moviment. Una peça no pot travessar a una altra (encara que el cavall pot saltar sobre les altres), ni pot desplaçar-se a una casella ocupada per una peça del mateix color. No obstant això, les peces poden moure's a les caselles ocupades per les peces enemigues per a capturar-les. Les peces es mouen pel tauler amb tres objectius: capturar una peça enemiga (reemplaçant-la en la seva casella original), defensar a les peces aliades per a evitar que siguin capturades, o controlar les caselles clau del tauler.

- **El rei**

  Quant al rei, és la peça més important però també la més feble. Només pot avançar una casella en qualsevol adreça: cap amunt, a baix, als costats o en diagonal. En cap cas el rei pot moure's a una casella en la qual quedaria en escac, és a dir, en la qual podria ser capturat. És important recordar que el rei es troba en escac quan és directament amenaçat per una altra peça.

- **La reina**

La dama o reina és la peça més forta en el joc. Té la capacitat de moure's en qualsevol direcció (cap endavant, enrere, als costats i en diagonal) i per tantes caselles com es desitgi, sempre que no passi sobre una peça del mateix color. Si la dama captura una peça enemiga, el seu moviment acaba, com ocorre amb totes les altres peces.

- **La torre**

La torre té l'habilitat de desplaçar-se en línia recta tantes caselles com sigui necessari, però només en les direccions cap endavant, cap endarrere o cap als costats.

- **L'alfil**

L'alfil pot moure's sense restriccions en diagonal, en qualsevol direcció i per qualsevol quantitat de caselles. Cada alfil inicia la partida en una casella del mateix color (sigui clar o fosc) i ha de romandre en aquesta coloració al llarg del joc. Els dos alfils es complementen entre si en cobrir-se mútuament les febleses.

- **El cavall**

Els cavalls tenen un moviment únic en comparació amb altres peces: avancen dues caselles en una direcció i després una addicional en un angle de 90 graus, la qual cosa produeix la forma d'una lletra "L" en la seva trajectòria sobre el tauler. Addicionalment, els cavalls són les úniques peces que tenen la capacitat de saltar sobre altres peces en el seu camí.

- **El peó**

Els peons tenen un moviment i una capacitat de captura diferents: avancen en línia recta cap endavant, però capturen en diagonal. En una jugada, només poden avançar una casella, excepte en el seu primer moviment, en el qual tenen l'opció d'avançar dues caselles. La captura dels peons només és possible en les caselles situades en diagonal i davant d'ells. Els peons no poden retrocedir, ni tan sols per a capturar una peça rival. A més, si una peça obstrueix el camí d'un peó, aquest no pot avançar ni capturar a la peça que obstrueix.

Explicada cada peça i el seu comportament, cal explicar algunes regles especials incorporades per a fer el joc més dinàmic i entretingut:

- **Coronar un peó**

Els peons tenen la capacitat de coronar i convertir-se en qualsevol peça, excepte el rei, quan arriben a l'altre extrem del tauler. La majoria de les vegades, els peons que es coronen es converteixen en una dama, però poden triar qualsevol altra peça disponible. Només els peons tenen l'habilitat de promocionar.

- **Capturar al pas ("en passant")**

La regla de la captura "en passant" es refereix a una situació en la qual un peó avança dues caselles en el seu primer moviment i, en fer-ho, es col·loca al costat d'un peó enemic. En aquesta situació, el peó enemic té l'opció de capturar al primer peó movent-se a la casella que aquest hagués ocupat si només hagués avançat una casella. Aquesta jugada només es pot fer immediatament després del moviment del primer peó; en cas contrari, l'oportunitat de capturar "en passant" ja no estarà disponible.

- **Enroc**

La jugada de l'enroc permet al jugador realitzar dos moviments importants en una sola jugada: col·locar al rei en una posició més segura i treure a la torre de la seva cantonada per a posar-la en joc. En un sol torn, el rei es mou dues caselles cap a un costat i la torre se situa al costat oposat d'ell. No obstant això, s'han de complir les següents condicions per a poder realitzar l'enroc:

1. Ser la primera jugada del rei i de la torre involucrats.
2. No pot haver-hi cap peça entre el rei i la torre.
3. El rei no pot estar en escac ni passar per una casella amenaçada.

Si la torre més pròxima al rei es mou, es diu enroc curt o enroc del rei; si la torre del costat oposat es mou, es diu enroc llarg o enroc de la dama. Independentment del tipus d'enroc, el rei només es mou dues caselles en aquesta jugada.

Per acabar, cal esmentar quan la partida acaba, ja que no només es pot guanyar per mat, també es pot empatar o com es diu als escacs fer taules.

- **Guanyar per mat**

Un mat ocorre quan el rei està en escac i no té manera d'escapar. Per a escapar d'un escac, el rei pot fer tres coses:

1. Moure's a una casella segura (sense enrocar-se)
2. Bloquejar l'escac interposant una altra peça del seu propi bàndol
3. Capturar la peça que amenaça al rei.

Si el rei no pot escapar de l'escac, la partida acaba. En general, el rei no és capturat o retirat del tauler, simplement es declara l'escac i mat i la partida es dona per finalitzada.

- **Empatar una partida**

Hi ha 5 raons per les quals una partida pot acabar en taules:

- S'han efectuat 50 moviments i cap dels jugadors ha mogut un peó ni capturat una peça
- Un jugador pot declarar Taules si la mateixa posició es repeteix tres vegades (encara que no necessàriament tres vegades seguides).
- No hi ha suficients peces en el tauler per a forçar un escac i mat (exemple: un Rei i un Cavall contra un altre Rei)
- Els jugadors poden acordar Taules en qualsevol moment i deixar de jugar.
- La posició arriba a un punt mort en el qual és el torn d'un jugador, però el seu rei no està en escac i, tanmateix, no pot fer cap moviment reglamentari. Això es coneix comunament com "ofegat".

2.2 Computació en els escacs

Wilhelm Steinitz, el primer Campió del món d'escacs a finals del segle XIX, va animar els seus oponents a atacar-lo per tal de desenvolupar estratègies d'escacs posicionals.

Mikhaïl Botvínnik, el sisè Campió del Món a mitjans del segle XX, va dedicar quantitats significatives de temps a la preparació física i mental abans de torneigs i partits, establint un nou estàndard per a la professionalitat en escacs.

A la dècada de 1970, la influència de l'onzè Campió del món Robert James Fischer va provocar canvis significatius en la teoria d'obertures, amb nous sistemes com l'eriçó i noves idees descobertes en obertures antigues, mostrant les infinites possibilitats d'aquest joc.

No obstant això, la revolució més innovadora en els escacs ha estat l'aparició de motors d'escacs per ordinador. Fins i tot els principiants ara estan familiaritzats amb els motors populars com "Stockfish" i "Rybka", i les aplicacions del mòbils són ara més fortes que els Grans Mestres. Com a resultat, els jugadors han començat a utilitzar aquestes eines per aprendre i millorar el seu propi joc.

La història de l'aparició de la primera màquina que pogués jugar a escacs ve de la mà del mateix home a qui se li pot atribuir la creació dels ordinadors, Alan Turing, qui el 1950, va escriure el primer programa d'escacs per ordinador i va ser pioner en el camp dels ordinadors digitals. En aquell moment, Turing va haver de conformar-se amb una simulació de l'execució del seu programa amb llapis i paper. El programa de Turing era un jugador terrible, però va servir bé el seu propòsit principal: va demostrar que els ordinadors poden jugar als escacs. El mateix any, Claude Shannon va traçar un pla d'acció perquè els ordinadors fossin programats per jugar bé als escacs. Donada la velocitat extremadament baixa dels ordinadors en el 1950, no estava clar si els ordinadors mai podrien vèncer els humans fins i tot en jocs tan senzills com les dames.

El 1958, un programa d'escacs va vèncer un jugador humà per primera vegada. El jugador humà, un secretari de l'equip de programadors que mai havia jugat als escacs abans, va rebre classes de jugar només una hora abans de la partida, i va ser derrotat pel programa d'escacs. Per poc remarcable que aquesta gesta pogués semblar avui, servia per demostrar que el coneixement podria ser incrustat en un programa d'escacs.

Després d'aquesta victòria, alguns dels primers programadors d'ordinadors que jugaven als escacs van predir que abans dels anys seixanta hi hauria un programa d'escacs que seria el campió del món. Però, a finals dels anys seixanta, Spasski era el campió del món d'escacs i els programes d'escacs jugaven al nivell d'un jugador d'escacs d'institut. Això va animar a alguns a afirmar que els ordinadors mai podran acomplir cap tasca intel·ligent, per no esmentar l'obtenció del campionat mundial.

No obstant això, la predicció es va fer de nou en els anys setanta, en aquesta ocasió implicant una aposta entre David Levy, un mestre d'escacs britànic (els jugadors d'escacs són classificats per la Federació Internacional d'Escacs segons la seva actuació. Els títols es concedeixen apropiadament, entre ells tenim Mestre Internacional (MI), Gran Mestre (GM) i Campió del Món. Levy és un MI.), i John McCarthy, un distingit investigador en intel·ligència artificial. L'aposta, d'intencions més modestes, afirmava que el 1978, un ordinador podria vèncer el senyor Levy en un partit d'escacs.

El 1978, el millor programa d'escacs de l'època (CHESS 4.7) va ser derrotat per Levy en un partit al millor de cinc jugat a Toronto amb una puntuació de 3 victòries per als humans, unes taules i una victòria per al CHESS 4.7.

Si el propòsit de l'aposta era fer que els investigadors pensessin dues vegades abans de predir una victòria, llavors aquesta aposta amb prou feines va servir: Malgrat les prediccions fallides de 1958-1968 i 1968-1978, els experts en escacs per ordinador van predir de nou que un ordinador obtindria el campionat mundial d'escacs durant els deu anys vinents. La seva lògica era que, després de tot, hi ha un llarg camí des del nivell mitjà (on estaven inicialment) fins a l'exigència d'una partida a un Mestre Internacional... la partida perduda pel senyor Levy.

Però una vegada més, el 1988 va arribar i, tot i això, el campió d'escacs era humà. L'any següent, l'ordinador d'escacs Deep Thought (DT) va vèncer fàcilment l'MI Levy. DT era l'ordinador d'escacs més fort mai construït. Els desenvolupadors van afirmar que l'ordinador Deep Thought estava a només tres anys del títol mundial.

El 1989 es va jugar un partit d'exhibició entre el campió del món Garry Kasparov i DT, que va acabar en una derrota de Kaspàrov. A partir d'aquest moment es va considerar que els ordinadors havien superat als humans.

Un cop coneguda la història, cal saber, com juga un ordinador als escacs? Un ordinador pot jugar als escacs generant moviments a l'atzar, verificant la seva validesa, i escollint-ne un per jugar. Si fos aquest el cas, aquest ordinador seria un jugador feble i sovint seria derrotat en només uns pocs moviments. La viabilitat de dissenyar un programa que pugui jugar bé els escacs depèn de si és possible considerar totes les conseqüències possibles de cada moviment fins al final de la partida, i després seleccionar l'estratègia òptima. L'estratègia òptima garantiria una seqüència guanyadora de moviments independentment dels moviments de l'oponent. Per exemple, al tres en ratlla, l'estratègia òptima sempre és un empat o victòria. No obstant això, considerar cada moviment possible i les seves respostes potencials no és factible a causa del nombre astronòmic de possibilitats, que s'estima que són aproximadament 10^{40}. Com a resultat, l'espai de cerca es limita a aproximar el moviment òptim entre el gran nombre de possibilitats.

Per a poder facilitar aquest càlcul de jugades, Claude Shannon va proposar dues estratègies el 1950 per limitar el nombre de possibilitats considerades per un ordinador que jugava als escacs. Aquestes estratègies, que encara s'utilitzen avui en dia tant per programes d'escacs competents com incompetents, s'anomenen l'estratègia del tipus "Xannon A". La primera estratègia implica considerar tots els moviments possibles fins a un cert nombre de profunditats, moment en què l'ordinador selecciona el millor moviment basat en alguns criteris, com la posició de les peces al final de la cerca. Aquest enfocament reconeix que és impossible considerar tots els moviments possibles i contramoviments fins al final del joc, de manera que l'ordinador limita la profunditat de la seva cerca per produir una resposta oportuna.

Curiosament, fins i tot els jugadors d'escacs humans només miren endavant un nombre limitat de moviments fins que aconsegueixen una posició satisfactòria. Un cop l'ordinador avalua una posició, necessita un mètode per seleccionar la millor seqüència entre tots els moviments provats i avaluats. Una funció d'avaluació assigna un nombre a cada posició basant-se en característiques conegudes que donen un avantatge al jugador, com ara avantatge material o posicional i desenvolupament. Per exemple, si l'ordinador té més peces en el tauler i un fort atac després d'una seqüència de moviments, la posició se li assigna un

gran enter positiu. Per contra, si la situació és la contrària, s'assigna la negació d'aquest nombre positiu.

La Teoria dels Jocs de Von Neumann i Morgenstern proporciona una manera de seleccionar la millor seqüència entre tots els moviments provats i avaluats. Es van adonar que una partida es pot representar com un arbre de posicions i possibles moviments, arrelats en la posició actual amb cada branca representant un possible moviment legal. Cada branca condueix a nous nodes a l'arbre, que representen les posicions resultants després de cada moviment. S'afegeixen noves branques a l'arbre, i el procés continua.

En aquest arbre de joc, el jugador tria el moviment que maximitza el seu guany, que també es coneix com a "estratègia mínima". La funció d'avaluació determina si un moviment maximitza el guany d'un jugador. El jugador que va moure per última vegada tria les branques que són la millor resposta per a cada posició en el penúltim nivell, que està associat amb una puntuació. L'altre jugador llavors tria el seu moviment segons les puntuacions obtingudes per a cada posició. L'ordinador selecciona el moviment que condueix a la posició amb la millor puntuació d'avaluació. No obstant això, els ordinadors no poden anar més enllà d'una profunditat de deu moviments de mitjana a causa de l'augment exponencial de jugades.

Els programes que s'executen en ordinadors d'alta velocitat tendeixen a utilitzar l'estratègia de tipus A, mentre que els programes que s'executen en arquitectures de microprocessadors utilitzen l'estratègia de tipus B. S'han desenvolupat algunes tècniques per millorar ambdues estratègies, que han permès als ordinadors jugar significativament millor amb el temps. Aquestes tècniques inclouen:

- Pronació alfa-beta: Aquesta tècnica es basa en la suposició que si l'oponent ja ha trobat un moviment fort en una situació donada, no val la pena explorar altres possibilitats que condueixin a la mateixa situació. En comptes d'això, l'ordinador pot centrar-se en moviments que eviten aquestes posicions, cosa que pot reduir molt l'espai de cerca.

- Llibres d'obertures: Amb els anys, els jugadors d'escacs han descobert certes seqüències d'obertura (primers moviments d'una partida) que es consideren òptimes per a ambdós bàndols. Aquestes seqüències s'han compilat en bases de dades, que poden ser utilitzades pels programes d'escacs per fer els seus moviments inicials.

- Bases de dades de finals de partida: De la mateixa manera que els llibres d'obertura, hi ha bases de dades de posicions finals que han estat analitzades extensament tant per humans com per ordinadors. Aquestes bases de dades poden ajudar els programes a fer moviments òptims en situacions finals.

- Maquinari especialitzat: A mesura que els principals colls d'ampolla en els escacs per ordinador es van fer evidents, els enginyers van començar a construir maquinari personalitzat per accelerar les tasques que requereixen més temps, com ara avaluar posicions i generar moviments.

- Moviments assassins: Si un moviment s'ha identificat com a molt bo o molt dolent en una posició similar abans, és probable que torni a ser bo o dolent. Els programes d'escacs poden aprofitar-se d'això provant primer aquests "moviments assassins", que poden podar grans parts de l'arbre de cerca.

- Taules de transposició: En alguns casos, es pot arribar a la mateixa posició mitjançant diferents seqüències de moviments. Per evitar càlculs redundants, els programes d'escacs poden emmagatzemar els resultats de les cerques anteriors en memòria, i reutilitzar-los quan troben posicions equivalents.

- Profunditat variable: Els programes d'escacs poden utilitzar diverses heurístiques per decidir quan deixar de buscar més a l'arbre. Per exemple, poden veure l'estabilitat o l'agressivitat d'una posició, o quan ha canviat recentment la funció d'avaluació.

2.3 Aprenentatge automàtic: algorismes i tècniques

Els algorismes d'aprenentatge automàtic són algorismes que utilitzen dades per aprendre i millorar la seva capacitat per dur a terme tasques específiques sense ser programats explícitament per a aquestes tasques. Això significa que l'algorisme d'aprenentatge automàtic pot detectar patrons i relacions en les dades que li són proporcionades i utilitzar aquest coneixement per executar tasques noves i diferents.

Els algorismes d'aprenentatge automàtic es poden dividir en tres categories principals: supervisat, no supervisat i per reforçament. En l'aprenentatge supervisat, l'algorisme rep un conjunt de dades etiquetades, és a dir, un conjunt de dades amb una resposta coneguda per a cada entrada, i utilitza aquestes dades per aprendre a predir noves entrades. En l'aprenentatge no supervisat, l'algorisme rep un conjunt de dades no etiquetades i ha de trobar patrons i agrupaments en les dades per comprendre millor la seva estructura. En l'aprenentatge per reforçament, l'algorisme aprèn a partir de la interacció amb l'entorn, rebent recompenses o penalitzacions per a les seves accions i ajustant la seva estratègia per maximitzar la recompensa.

Els algorismes d'aprenentatge automàtic són utilitzats en una gran varietat d'àrees, com ara la classificació d'imatges, la recomanació de productes, la detecció de fraus, la conducció autònoma i la detecció de malalties, entre altres.

Els principals algorismes i tècniques per a dur a terme aquest aprenentatge són els següents:

- Arbres de decisió
  L'algorisme de l'arbre de decisió implica utilitzar una estructura similar a un arbre per modelar les decisions i els seus resultats, incloent-hi probabilitats, costos i beneficis. Aquest enfocament permet que un procés sistemàtic i lògic de resolució de problemes arribi a la conclusió correcta. Una versió avançada d'aquest algorisme és l'algorisme d'Arbre Aleatori, que utilitza múltiples arbres de decisió per evitar el sobreajustament (quan un algorisme s'ajusta només a les dades d'entrenament però no a totes), un problema comú amb els models d'arbres de decisió.

- Algorismes bayesians
  Els algorismes bayesians utilitzen els principis del teorema bayesià per analitzar la relació entre diverses variables i estimar la probabilitat d'un esdeveniment o resultat basat en aquesta relació. S'utilitzen habitualment en problemes de regressió i classificació, on l'objectiu és predir el valor d'una variable dependent basada en els valors de variables independents. L'enfocament bayesià és particularment útil quan es tracta de dades incertes o incompletes, ja que permet la incorporació de

coneixement previ i actualització de probabilitats a mesura que noves dades estiguin disponibles.

- Algorismes de regressió
  Els algorismes de regressió són útils en l'aprenentatge automàtic estadístic, ja que intenten crear un model que representa la relació entre variables. En examinar aquestes relacions, es construeix una funció que aproxima la relació observada. A mesura que s'observen més variables, el model esdevé més precís i proporciona un rang de valors possibles amb un cert grau de confiança i error.

- Vector de suport
  És una tècnica utilitzada per classificar punts de dades en un espai multidimensional. Es tracta de la creació d'un hiperplà que separa les dades en diferents grups amb el marge màxim possible entre ells. Aquest enfocament s'utilitza comunament en aplicacions com la publicitat i l'empalmament d'ARN humà per classificar amb precisió els punts de dades i fer prediccions basades en les seves característiques.

- Mètodes d'acoblament
  Són tècniques d'aprenentatge automàtic que impliquen combinar diversos models per millorar el rendiment. Aquests models són sovint diferents i més febles en l'aïllament, però quan es combinen poden produir millors resultats. La idea darrere dels mètodes de conjunt és manejar les limitacions dels models individuals palanquejant els punts forts dels altres. Combinant models, es pot reduir el biaix i la variància, així com reduir el sobreajustament mitjançant la mitjana de les prediccions dels models.

- Algorismes d'agrupació
  Aquests algorismes estan dissenyats per agrupar punts de dades en grups basats en les seves similituds. Això ajuda a identificar patrons i tendències dins de les dades. Hi ha diversos mètodes d'agrupament, com ara jeràrquic, centroide, distribució i densitat, cadascun amb la seva manera única d'agrupar les dades. Aquests mètodes es poden utilitzar per a diversos propòsits com la segmentació del client, la segmentació de la imatge i la detecció d'anomalies.

- Algorismes d'aprenentatge per regla d'associació
  Aquest concepte es refereix a l'establiment de regles entre diferents elements i transaccions dins d'un conjunt de dades. Concretament, implica identificar la probabilitat que un element o conjunt d'elements es produeixi donada l'ocurrència d'un altre element o conjunt d'elements. Aquestes regles es determinen analitzant els diversos conjunts d'elements i elements individuals presents a la base de dades.

- Xarxes neuronals artificials
  Consisteixen en neurones artificials interconnectades que es comuniquen entre elles, i la força de les connexions entre elles s'ajusta segons experiències d'aprenentatge anteriors. A mesura que s'introdueixen més dades a la xarxa, continua aprenent i millorant. Hi ha diversos algorismes associats amb Xarxes Neurals Artificials, i un dels més significatius és l'aprenentatge profund. L'aprenentatge profund implica construir xarxes neuronals més grans i complexes que puguin gestionar tasques més complexes.

- Algorismes d'anàlisi de dimensionalitat
  Dimensionalitat es refereix al nombre de variables i les seves dimensions corresponents en un conjunt de dades donat. Les tècniques de reducció de la dimensionalitat estan dissenyades per reduir el nombre de variables en les dades, tot conservant informació important. Aquest procés ajuda a eliminar variables redundants o irrellevants, al mateix temps que conserva les importants que contribueixen al resultat general.

2.4 Xarxes neuronals: definició i tipus

Les xarxes neuronals són un tipus d'aprenentatge automàtic dissenyat per assemblar-se a l'estructura i funció del cervell humà. També es coneixen com a xarxes neuronals artificials o xarxes neuronals de simulació, i poden ajudar els ordinadors a aprendre dels seus errors per tal de millorar contínuament. Les xarxes neuronals s'utilitzen per resoldre problemes difícils com ara resumir documents i reconèixer cares. Mitjançant l'ús de conjunts de dades etiquetats, les xarxes neuronals poden classificar i agrupar dades segons entrades similars.

El seu funcionament es basa a aprendre a generar la sortida correcta sense la necessitat de regles de programació precises. Per fer això, van analitzant exemples etiquetats durant l'entrenament per determinar les característiques d'entrada necessàries per produir la sortida correcta. Amb prou exemples, les xarxes neuronals poden processar noves entrades desconegudes i produir resultats precisos. Com més experiència i aportacions observen, més precisos són els resultats. Hi ha quatre procediments essencials perquè les xarxes neuronals funcionin correctament: recordar patrons, categoritzar informació, agrupar dades i fer prediccions. Les xarxes neuronals poden emprar diferents tipus d'aprenentatge, com l'aprenentatge supervisat, no supervisat i el reforç.

Les xarxes neuronals artificials estan dissenyades per imitar la funció del cervell humà i resoldre problemes complexos. Hi ha diversos tipus de xarxes neuronals artificials que difereixen en complexitat, estructura i casos d'ús. Aquestes xarxes estan dissenyades per reflectir neurones i sinapsis, amb diferents tipus de neurones artificials i connexions entre nodes. El flux de dades a través de la xarxa i la densitat dels nodes també són factors importants que diferencien cada tipus de xarxa neuronal.

Cada tipus de xarxa neuronal té les seves pròpies característiques úniques i és adequat per resoldre problemes específics. Alguns exemples de diferents tipus de xarxes neuronals artificials són les següents:

- Xarxes neuronals de retroalimentació

  És un tipus de model d'aprenentatge automàtic dissenyat per a processar dades seqüencials o dades amb una estructura temporal. A diferència de les xarxes neuronals tradicionals, que processen cada entrada de manera independent, aquestes xarxes tenen connexions de retroalimentació que els permeten mantenir un estat intern o memòria de les entrades anteriors.

- Les xarxes neuronals de perceptró i perceptró multicapa

  El Perceptró és un tipus bàsic de xarxa neuronal artificial que es va crear en els inicis. Serveix com a classificador binari, el que significa que classifica les dades en dos grups

diferents. És un model lineal i és una de les formes més simples d'una xarxa neuronal artificial.

Les xarxes multicapa són més complexes i denses que els perceptrons. Tenen la capacitat de contenir nombroses capes ocultes entre les capes d'entrada i sortida. Cada node d'una capa específica està connectat a cada node de la capa següent, creant xarxes completament connectades que són útils per a l'aprenentatge profund.

Aquestes xarxes es poden utilitzar per a tasques més complicades com el reconeixement de veu i la classificació intricada. No obstant això, a causa de la seva profunditat i complexitat, aquests models poden requerir una potència de processament significativa i temps de manteniment.

- Les xarxes neuronals de funció radial

Les xarxes neuronals de funció radial consisteixen en una capa d'entrada, una capa amb nodes que utilitzen funcions radials amb paràmetres variables, i una capa de sortida. Aquests models es poden utilitzar per a diverses tasques com la classificació, la regressió per a sèries temporals i el control del sistema. Les funcions de base radial s'utilitzen per calcular el valor absolut entre un punt central i un punt donat. En les tasques de classificació, les funcions de base radial s'utilitzen per calcular la distància entre una entrada i una classificació apresa. Si l'entrada és la més propera a una etiqueta específica, es classifica com a tal.

Les xarxes de funció radial s'utilitzen comunament en el control del sistema, com ara controlar la restauració d'energia després d'una apagada d'energia. La xarxa neuronal pot prioritzar la restauració de l'energia a zones amb el nombre més gran de persones o serveis bàsics.

- Les xarxes neuronals recurrents

Les xarxes neuronals recurrents són particularment efectives quan el model necessita gestionar dades seqüencials. El model utilitza bucles de retroalimentació per referir-se als passos anteriors de la xarxa neuronal, permetent-li complir millor les tasques i millorar les seves prediccions. Les capes entre les capes d'entrada i sortida són "recurrents", el que significa que la informació important es reté i torna als bucles. Això permet al model entendre millor el context d'una entrada i refinar la seva sortida. Per exemple, un sistema de text predictiu podria utilitzar informació de la paraula anterior per predir millor la paraula següent. Les xarxes neuronals recurrents són especialment útils per al processament del llenguatge natural, com ara els xatbots reactius o la traducció del llenguatge. Aquests models sovint utilitzen models de seqüència a seqüència, que consisteixen tant en una xarxa codificadora com descodificadora.

- Les xarxes neuronals modulars

Una xarxa neuronal artificial modular es compon de múltiples xarxes o components més petits que operen independentment, però treballen junts per dur a terme una tasca específica. Desglossant les tasques complexes en components més petits, es pot augmentar la velocitat de processament general. Cada xarxa de components realitza una subtasca diferent que, quan es combina, completa la tasca global i produeix la sortida. Aquest tipus de xarxa neuronal artificial pot millorar significativament l'eficiència de processos complexos i es pot utilitzar en una varietat d'entorns.

3. Metodologia

3.1 Elecció del model de xarxa neuronal

En aquest apartat es parla sobre les decisions preses per a triar el tipus de xarxa neuronal.

Amb el propòsit de la xarxa d'agafar com entrada una posició en una partida d'escacs i treure'n l'avaluació, dintre de les xarxes explicades anteriorment l'opció triada ha estat un perceptró multicapa o MLP (per les seves sigles en anglès Multi-Layer Perceptron). Aquesta decisió ha estat motivada perquè s'ha interpretat el problema com una classificació, on s'entra la posició al tauler d'escacs i la sortida classificarà aquesta posició segons la seva avaluació en un rang donat. Per a aquestes condicions, el MLP s'ajusta al problema perquè presenta molta flexibilitat en l'elecció de l'arquitectura. Això significa que es pot personalitzar el nombre de capes ocultes i la quantitat de neurones en cada capa per a adaptar-se al problema en qüestió. Per tant, es pot ajustar l'arquitectura per a millorar el rendiment del model.

A més, El MLP és una xarxa neuronal que pot manejar dades no lineals, com ho serien les posicions d'escacs, i pot capturar aquesta complexitat a través de la combinació de múltiples capes de neurones. També, presenta un bon rendiment en problemes de classificació, i com s'ha esmentat anteriorment, la funció d'aquesta xarxa serà seleccionar un valor (en un rang) com a avaluació d'una posició. Pel que fa a entrades petites (sense gran nombre de valors) el MLP presenta una bona efectivitat, per tant, si es codifica correctament l'entrada amb un nombre relativament petit de bits el MLP pot ser un model eficient i fàcil d'entrenar per a aquest problema.

En resum, el MLP és una bona opció per al problema d'avaluació de posicions d'escacs degut a la seva flexibilitat en l'elecció de l'arquitectura, capacitat per a manejar dades no lineals, bon rendiment en problemes de classificació i efectivitat en problemes de petita escala.

3.2 Selecció i creació de la base de dades de posicions d'escacs

El següent pas en la construcció de la xarxa neuronal és la selecció de dades, tant per a l'entrenament de la xarxa com per al posterior testatge d'aquesta.

Les notacions de les partides d'escacs són molt variades i han anat canviant molt durant el temps. En els escacs per a ordinadors hi ha dues notacions principals:

- PGN (Portable Game Notation) és un format de fitxer utilitzat per a gravar moviments en notació algebraica anglesa estàndard, juntament amb marques addicionals per a especificar jugadors i circumstàncies de joc.
- Steno-Chess és un altre format de fitxer dissenyat per al processament d'ordinadors. Se sacrifica la capacitat dels humans de jugar a través dels jocs a canvi d'un format més concís que requereix menys dades per emmagatzemar un joc.

- FEN (Forsyth-Edwards Notation) és un format d'una sola línia que especifica les posicions actuals de les peces en un tauler d'escacs, així com altres informacions com drets d'enroc, nombre de moviment i color en moviment. Sovint s'utilitza per generar un tauler en una posició no estàndard.

- Extended Position Description (EPD) és un format que proporciona les posicions actuals d'un tauler d'escacs amb un conjunt de valors d'atribut estructurats, utilitzant caràcters ASCII. Està dissenyat per a l'intercanvi de dades i comandaments entre els programes d'escacs i per a la representació de bases de dades portàtils d'obertures. EPD és una opció millor que FEN per a certes variants d'escacs.

Per a l'obtenció de gran quantitat de posicions, el que caldrà fer serà accedir a una base de dades de partides (on es trobaran partides en format PGN, ja que es representen partides completes) i, un cop tinguem cada partida, representar cada moviment d'aquesta en format FEN, finalment, guardar aquest FEN per a la posterior transformació en el format més adient per a la nostra xarxa neuronal.

L'obtenció de les partides també ha estat deliberada, ja que no han estat agafades de qualsevol jugador sinó que s'ha fet un recull de partides de grans mestres amb diferents estils de joc (defensiu, ofensiu, tàctic...) per a tenir en compte la qualitat dels jocs seleccionats. Aquesta selecció de jocs ha estat extreta de la pàgina https://www.pgnmentor.com/, on es pot trobar un històric de partides (en format PGN) de totes les partides oficials jugades per tots els grans mestres. D'aquesta pàgina s'han seleccionat els següents arxius:

- Adams.pgn – GM Michael Adams, 3422 partides
- Akobian.pgn – GM Varuzhan Akobian, 1442 partides
- Carlsen.pgn – GM Magnus Carlsen, 4668 partides
- Caruana.pgn – GM Fabiano Caruana, 3714 partides
- Jovaba.pgn – GM Baadur Jobava, 4140 partides
- Kasparov.pgn – GM Garry Kasparov, 2128 partides
- MacKenzie.pgn – GM George MacKenzie, 198 partides
- Malakhov.pgn – GM Vladimir Malakhov, 2006 partides
- McDonnell.pgn – GM Alexander McDonnell, 106 partides
- Nakamura.pgn – GM Hikaru Nakamura, 6059 partides
- VachierLagrave.pgn – GM Maxime Vachier-Lagrave, 3927 partides

Aquesta selecció dona un total de 31810 partides. El següent punt en la creació de la base de dades serà transformar cadascun d'aquests PGN a totes les diferents posicions que es veuen al llarg de la partida. Com es pot veure a la il·lustració 2, cadascuna de les partides està formada per les dades de la partida (data, lloc, resultat...) seguides de la sèrie de moviments que es van fent partint de la posició inicial.

```
[Event "Lloyds Bank op"]
[Site "London"]
[Date "1984.??.??"]
[Round "1"]
[White "Adams, Michael"]
[Black "Sedgwick, David"]
[Result "1-0"]
[WhiteElo ""]
[BlackElo ""]
[ECO "C05"]

1.e4 e6 2.d4 d5 3.Nd2 Nf6 4.e5 Nfd7 5.f4 c5 6.c3 Nc6 7.Ndf3 cxd4 8.cxd4 f6
9.Bd3 Bb4+ 10.Bd2 Qb6 11.Ne2 fxe5 12.fxe5 O-O 13.a3 Be7 14.Qc2 Rxf3 15.gxf3 Nxd4
16.Nxd4 Qxd4 17.O-O-O Nxe5 18.Bxh7+ Kh8 19.Kb1 Qh4 20.Bc3 Bf6 21.f4 Nc4 22.Bxf6 Qxf6
23.Bd3 b5 24.Qe2 Bd7 25.Rhg1 Be8 26.Rde1 Bf7 27.Rg3 Rc8 28.Reg1 Nd6 29.Rxg7 Nf5
30.R7g5 Rc7 31.Bxf5 exf5 32.Rh5+  1-0
```

Il·lustració 2. Dades d'una partida en un arxiu PGN

Per a fer les transformacions he creat una funció en Python per a llegir cadascun d'aquests fitxers, i mitjançant l'ús de la llibreria "chess" i del motor d'escacs Stockfish es genera cada moviment a un taulell virtual i a partir d'aquest es genera el FEN corresponent a la posició abans de passar al moviment següent, a més, també s'avalua (se li dona un valor numèric per a saber qui va guanyant) la posició. Aquests dos valors generats es guarden en un arxiu CSV per a poder ser utilitzats posteriorment quan s'hagi d'entrenar la xarxa neuronal. En total, després de transformar totes les partides, s'obté un total de 2660412 posicions amb la seva corresponent avaluació.

3.3 Codificació de les dades d'entrada

Tot i tenir la partida amb una codificació estàndard com és el FEN, aquesta codificació s'ha de transformar per a poder ser entesa per la xarxa neuronal. Aquesta transformació implica convertir cadascuna de les peces i els atributs d'una partida d'escacs en un valor numèric que pugui ser entès per la xarxa i usat per a calcular la sortida. Perquè la mateixa codificació FEN ja presenta els atributs necessaris, m'he basat en aquesta per a la codificació. A continuació, es mostra el FEN que conté la posició inicial i totes les seves dades:

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

En aquesta codificació es tenen en compte 6 factors, separats pels cinc espais entre el text:

1. Dades de col·locació de peces: Cada fila és descrita, començant amb la 8 i acabant amb la 1, amb un "/" entre cada una; dins de cada fila, els continguts de les caselles és descrit en ordre des de la columna "a" fins a la columna "h". Cada peça s'identifica amb una sola lletra presa dels noms en anglès de les peces (peó = "P", cavall = "N", alfil = "B", torre = "R", dama = "Q" i rei = "K"). Les peces blanques es designen utilitzant lletres majúscules ("PNBRQK"), mentre que les negres utilitzen lletres minúscules ("pnbrqk"). Un conjunt d'un o més quadrats buits consecutius dins d'un rang es denota per un dígit de "1" a "8", corresponent al nombre de quadrats.

2. Color actiu: «w» vol dir que les blanques han de moure; «b» vol dir que les negres han de moure.

3. Disponibilitat d'enroc: Si cap dels dos bàndols té la capacitat d'enrocar, aquest camp utilitza el caràcter "-". En cas contrari, aquest camp conté una o més lletres: "K" si les blanques poden enrocar al flanc de rei, "Q" si les blanques poden enrocar al flanc de dama, "k" si les negres poden enrocar al flanc de rei, i "q" si les negres poden enrocar al flanc de dama.

4. Quadre de destinació al pas: És una casella sobre la qual un peó acaba de passar mentre es mouen dues caselles; es dona en notació algebraica (columna-fila, per exemple, “e3”). Si no hi ha cap casella de destinació al pas, aquest camp utilitza el caràcter “-“. Això es registra independentment de si hi ha un peó en posició de capturar al pas.

5. Número de mig moviment: El nombre de semi-moviments (es considera moviment complet quan els dos jugadors mouen, semi-moviment considera cada moviment de jugar individual) des de l'última captura o avanç de peó, utilitzat per la regla dels cinquanta moviments.

6. Número de moviment complet: el nombre dels moviments complets. Comença a l'1 i s'incrementa després del moviment de les negres.

Per a la nostra xarxa neuronal, cal tenir en compte que només ens influeixen paràmetres que es tinguin en compte a l’hora d’avaluar el joc, per tant, el nombre de moviments no cal que els fem servir, ja que s’usen per a declarar taules o per a representar partides en motors de joc. Tenint triats els paràmetres que cal representar per la xarxa, cal codificar-los.

Per a la codificació, he representat els valors com a bits, on cadascun d’aquests tindrà un sentit diferent. La primera codificació pensada s’ajusta a la representació que fa un FEN i ha estat codificada de la manera següent:

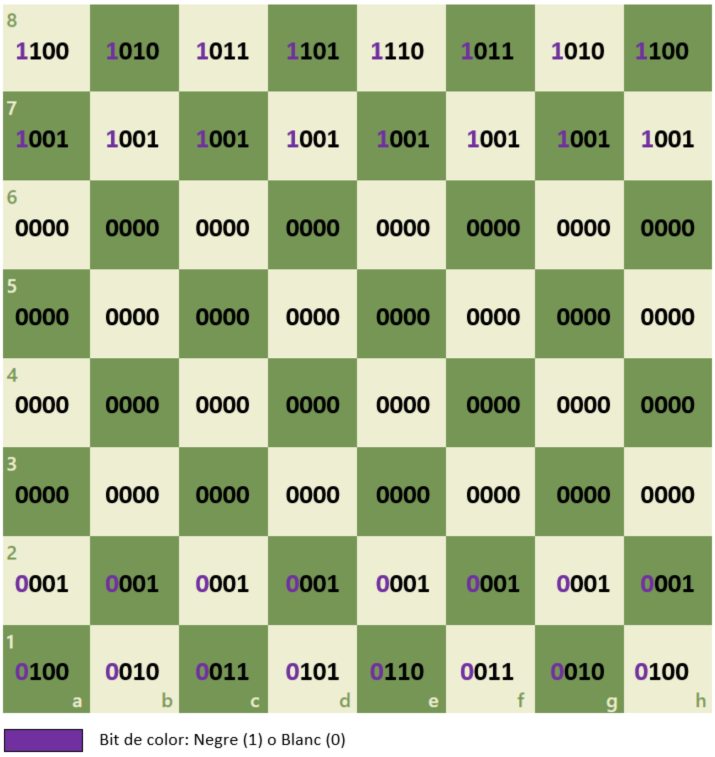
1. 4 bits per a representar les peces. Els tres de menys representaran un valor que identificarà la peça, mentre que el de més pes, servirà per a mostrar el color d’aquesta sent blanca (0) o negra (1). L’esplai en blanc estarà representat pel 0. La codificació de les peces és la següent:
    a. Buit → 000
    b. Peó → 001
    c. Cavall → 010
    d. Alfil → 011
    e. Torre → 100
    f. Reina → 101
    g. Rei → 110

2. El torn del jugador vindrà representat per 1 bit on estarà a 0 si és torn de blanques i a 1 si és torn de negres.

3. Els drets d’enroc es representaran amb 4 bits. Si cap dels dos bàndols té la capacitat d'enrocar, aquest valor serà “0000. En cas contrari, aquest camp contindrà un o més bits a 1 (com en la representació FEN “KQkq”): "1000" si les blanques poden enrocar al flanc de rei, "0100" si les blanques poden enrocar al flanc de dama, "0010" si les negres poden enrocar al flanc de rei, i "0001" si les negres poden enrocar al flanc de dama, per exemple, tenir tots els drets d’enroc disponibles es codificaria com “1111”.

4. La captura al pas es codificarà amb 7 bits, els tres de major pes per a la columna (sent “000” la “a” i “111” la “h”) i els tres següents la fila (sent “000” la 1 i “111” la 8), finalment. Si no hi ha cap casella de destinació al pas, s’utilitza el bit de menys pes i es codifica com “0000001“.

Com a referència, així es veuria la posició inicial si substituïm cada peça amb el valor de la seva codificació:



Il·lustració 3. Tauler on les peces s'han substituït per la seva codificació

Només caldria afegir els bits de:

- Torn: 1
- Drets d’enroc: 1111
- Captura al pas: 0000001

Finalment, la traducció del FEN a la codificació d'en bits de la posició inicial seria:

1100101010111101111010111010110010011001100110011001100110011001000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000010001000100010001000
1000101000010001101010110001100100100011110000001.

Aquesta codificació ens dona una entrada de 268 bits. Tot i que és una codificació bona, n'he volgut provar una altra. En aquesta segona tots els valors queden inclosos en el taulell, és a dir, afegint una mica més d'informació a cada casella es pot tenir tot el necessari, sense haver de codificar, per una banda, el taulell i seguidament les dades sobre aquest (com enroc o captura al pas).

Aquesta segona codificació intenta col·locar tota la informació a la casella, és a dir, cada casella identifica quina peça té al seu interior, de qui és el torn, si la peça disposa de drets d'enroc o si disposa de captura al pas. Per a fer-ho cada casella disposa de 7 bits (4 per a les variables i 3 per a la peça) distribuïts de la manera següent:

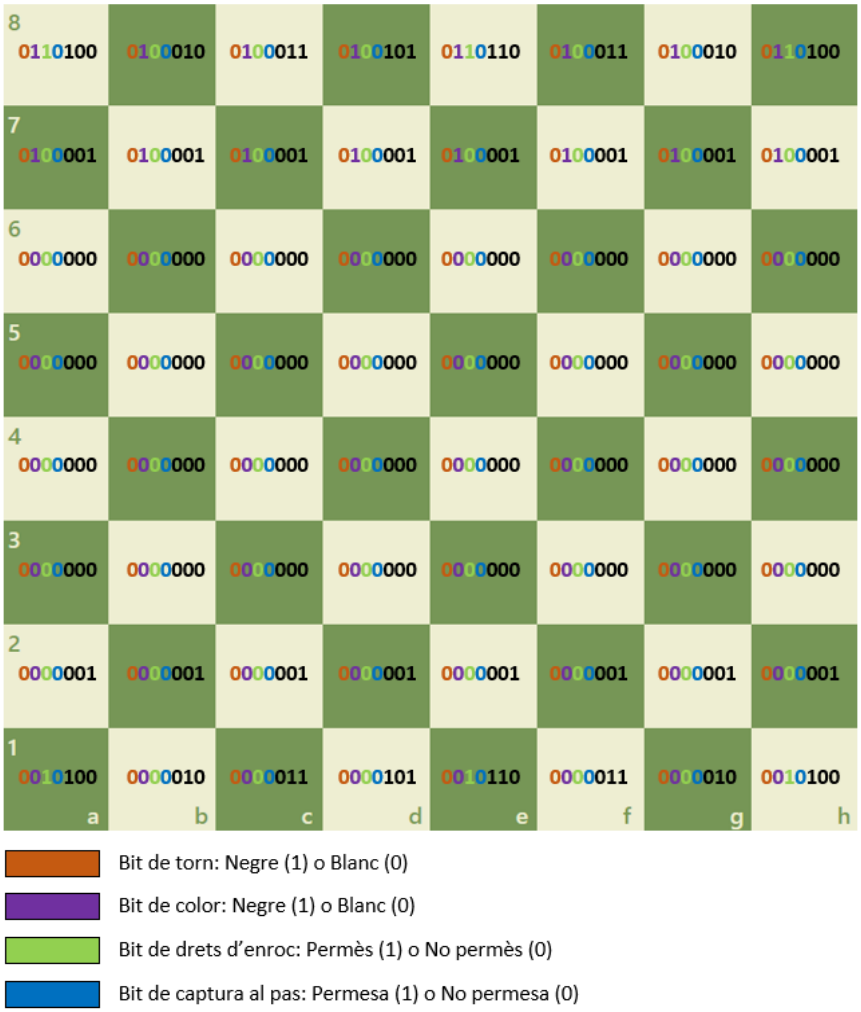
Casella → YYYY – XXX

- Bits de variables – YYYY – tcdp
  - Torn(t) – Informa del jugador que ha de moure: blanc (0) o negre (1)
  - Color(c) – Color de la peça que conté: blanc o buit (0) o negre (1)
  - Drets d'enroc(d):
    - Tot el que no sigui torre o rei: 0.
    - Torre o rei si es poden enrocar (1), en cas contrari (0).
  - Captura al pas(p):
    - Tot el que no sigui peó: 0.
    - Si el peó es pot capturar al pas (1), en cas contrari (0).
- Bits de peces:
  - Buit → 000
  - Peó → 001
  - Cavall → 010
  - Alfil → 011
  - Torre → 100
  - Reina → 101
  - Rei → 110

Pel fet que no totes les peces tenen tots els bits de decisió, per exemple, el peó sempre tindrà el bit d'enroc a 0, podem establir una codificació de 7 bits per a cada estat de la casella:

- Buit → t000-000
- Peó → tc0p-001
- Cavall → tc00-010
- Alfil → tc00-011
- Torre → tcd0-100
- Reina → tc00-101
- Rei → tcd0-110

Un cop establerta la nova codificació, podem veure com quedaria la posició inicial del taulell:



Il·lustració 4. Tauler on les peces s'han substituït per la seva segona codificació

Un cop la xarxa estigui dissenyada i llesta per a funcionar, s'executarà amb les mateixes condicions, però variant la codificació d'entrada per a comprovar quina de les dues té millors resultats i és seleccionada.

3.4 Codificació de les dades de sortida

En aquest apartat s'explica el tractament que reben les dades per a poder ser interpretades a la sortida de la xarxa neuronal. Com ja s'ha mencionat anteriorment, la xarxa classificarà les partides segons la seva avaluació, per tant, cal que la sortida representi els diferents rangs que podem tenir d'avaluacions i, així, la xarxa triarà quin s'adequa més a la posició d'entrada.

Igual que amb les dades d'entrada, no hi ha una codificació correcta de bits de sortida, aleshores, el més correcte serà identificar diferents i, posteriorment, testejar quina dona millors resultats. Podem representar els rangs de moltes maneres, per exemple, podem tenir bits per als rangs i un altre separat per al signe, una altra codificació seria tenir tots els rangs separats (en negatius o positius), una altra possibilitat seria representar el zero com un únic valor en el rang o incloure'l en algunes codificacions. Dintre de totes les possibles codificacions, les triades per a testejar han estat les següents:

1. En aquesta primera classificació, dividirem la sortida en 7 neurones, la primera d'elles identificarà el signe de l'avaluació mentre que la resta el rang on es troba el valor. La sortida quedarà distribuïda de la forma següent:
    a. El primer indicarà el signe de l'avaluació Positiu/Zero (0) o negatiu (1)
    b. Interval 1 → avaluació < 0.5
    c. Interval 2 → 0.5 <= avaluació >= < 1.5
    d. Interval 3 → 1.5 <= avaluació < 2.5
    e. Interval 4 → 2.5 <= avaluació < 3.5
    f. Interval 5 → 3.5 <= avaluació < 4.5
    g. Interval 6 → 4.5 <= avaluació

2. En la segona classificació provada, la intenció és separar les avaluacions en rangs incloent rangs pel signe, sense que aquest tingui la seva sortida a banda. A més, s'inclouen més rangs per a poder tenir la informació més classificada.
    a. Interval 1 → avaluació < 0.25
    b. Interval 2 → 0.25 <= avaluació < 0.5
    c. Interval 3 → 0.5 <= avaluació < 1
    d. Interval 4 → 1 <= avaluació < 1.5
    e. Interval 5 → 1.5 <= avaluació < 2.5
    f. Interval 6 → 2.5 <= avaluació < 3.5
    g. Interval 7 → 3.5 <= avaluació < 4.5
    h. Interval 8 → avaluació >= 4.5
    i. Interval 9 → -0.25 <= avaluació < 0
    j. Interval 10 → -0.25 <= avaluació > -0.5
    k. Interval 11 → -0.5 <= avaluació > -1
    l. Interval 12 → -1 <= avaluació > -1.5
    m. Interval 13 → -1.5 <= avaluació > -2.5
    n. Interval 14 → -2.5 <= avaluació > -3.5
    o. Interval 15 → -3.5 <= avaluació > -4.5
    p. Interval 16 → avaluació <= -4.5

3. La següent és una modificació de la segona on se li ha afegit un camp amb el 0 per separat, perquè pugui ser més precisa amb l'avaluació.
    a. Interval 1 → avaluació < 0.25
    b. Interval 2 → 0.25 <= avaluació < 0.5
    c. Interval 3 → 0.5 <= avaluació < 1
    d. Interval 4 → 1 <= avaluació < 1.5
    e. Interval 5 → 1.5 <= avaluació < 2.5
    f. Interval 6 → 2.5 <= avaluació < 3.5
    g. Interval 7 → 3.5 <= avaluació < 4.5
    h. Interval 8 → avaluació >= 4.5
    i. Interval 9 → -0.25 <= avaluació < 0
    j. Interval 10 → -0.25 <= avaluació > -0.5
    k. Interval 11 → -0.5 <= avaluació > -1
    l. Interval 12 → -1 <= avaluació > -1.5
    m. Interval 13 → -1.5 <= avaluació > -2.5
    n. Interval 14 → -2.5 <= avaluació > -3.5
    o. Interval 15 → -3.5 <= avaluació > -4.5
    p. Interval 16 → avaluació <= -4.5
    q. Interval 17 → avaluació = 0

4. En aquesta, s'ha seguit afegint atributs a l'avaluació amb la intenció de millorar el rendiment de la xarxa, en aquesta codificació s'afegeix el bit de signe a la sortida.
    a. Interval 1 → avaluació < 0.25
    b. Interval 2 → 0.25 <= avaluació < 0.5
    c. Interval 3 → 0.5 <= avaluació < 1
    d. Interval 4 → 1 <= avaluació < 1.5
    e. Interval 5 → 1.5 <= avaluació < 2.5
    f. Interval 6 → 2.5 <= avaluació < 3.5
    g. Interval 7 → 3.5 <= avaluació < 4.5
    h. Interval 8 → avaluació >= 4.5
    i. Interval 9 → -0.25 <= avaluació < 0
    j. Interval 10 → -0.25 <= avaluació > -0.5
    k. Interval 11 → -0.5 <= avaluació > -1
    l. Interval 12 → -1 <= avaluació > -1.5
    m. Interval 13 → -1.5 <= avaluació > -2.5
    n. Interval 14 → -2.5 <= avaluació > -3.5
    o. Interval 15 → -3.5 <= avaluació > -4.5
    p. Interval 16 → avaluació <= -4.5
    q. Interval 17 → avaluació = 0
    r. Interval 18 → Positiu/Zero (0) o negatiu (1)

5. Com es pot comprovar en les avaluacions anteriors, el primer interval també permetia comprovar si l'avaluació es negativa (ja que en alguns problemes la xarxa pot presentar millor rendiment si en comptes de triar una sola neurona de sortida ho ha de fer amb més d'una), en aquesta cinquena codificació s'elimina aquesta multi-sortida per a comprovar el rendiment.
    a. Interval 1 → 0 <= avaluació < 0.25
    b. Interval 2 → 0.25 <= avaluació < 0.5
    c. Interval 3 → 0.5 <= avaluació < 1
    d. Interval 4 → 1 <= avaluació < 1.5
    e. Interval 5 → 1.5 <= avaluació < 2.5
    f. Interval 6 → 2.5 <= avaluació < 3.5
    g. Interval 7 → 3.5 <= avaluació < 4.5
    h. Interval 8 → avaluació >= 4.5
    i. Interval 9 → -0.25 <= avaluació < 0
    j. Interval 10 → -0.25 <= avaluació > -0.5
    k. Interval 11 → -0.5 <= avaluació > -1
    l. Interval 12 → -1 <= avaluació > -1.5
    m. Interval 13 → -1.5 <= avaluació > -2.5
    n. Interval 14 → -2.5 <= avaluació > -3.5
    o. Interval 15 → -3.5 <= avaluació > -4.5
    p. Interval 16 → avaluació <= -4.5
    q. Interval 17 → avaluació = 0

6. Per a la sisena codificació de test afegim més rangs a la sortida per a precisar més la sortida.
    a. Interval 1 → avaluació < 0.25
    b. Interval 2 → 0.25 <= avaluació < 0.5
    c. Interval 3 → 0.5 <= avaluació < 0.75

d. Interval 4 → 0.75 <= avaluació < 1
e. Interval 5 → 1 <= avaluació < 1.25
f. Interval 6 → 1.25 <= avaluació < 1.5
g. Interval 7 → 1.5 <= avaluació < 2.5
h. Interval 8 → 2.5 <= avaluació < 3.5
i. Interval 9 → 3.5 <= avaluació < 4.5
j. Interval 10 → avaluació >= 4.5
k. Interval 11 → -0.25 <= avaluació < 0
l. Interval 12 → -0.25 <= avaluació > -0.5
m. Interval 13 → -0.5 <= avaluació > -0.75
n. Interval 14 → -0.75 <= avaluació > -1
o. Interval 15 → -1 <= avaluació > -1.25
p. Interval 16 → -1.25 <= avaluació > -1.5
q. Interval 17 → -1.5 <= avaluació > -2.5
r. Interval 18 → -2.5 <= avaluació > -3.5
s. Interval 19 → -3.5 <= avaluació > -4.5
t. Interval 20 → avaluació <= -4.5

7. Per a l'última codificació, afegim una sortida a la codificació anterior per a comprovar el 0.
    a. Interval 1 → avaluació < 0.25
    b. Interval 2 → 0.25 <= avaluació < 0.5
    c. Interval 3 → 0.5 <= avaluació < 0.75
    d. Interval 4 → 0.75 <= avaluació < 1
    e. Interval 5 → 1 <= avaluació < 1.25
    f. Interval 6 → 1.25 <= avaluació < 1.5
    g. Interval 7 → 1.5 <= avaluació < 2.5
    h. Interval 8 → 2.5 <= avaluació < 3.5
    i. Interval 9 → 3.5 <= avaluació < 4.5
    j. Interval 10 → avaluació >= 4.5
    k. Interval 11 → -0.25 <= avaluació < 0
    l. Interval 12 → -0.25 <= avaluació > -0.5
    m. Interval 13 → -0.5 <= avaluació > -0.75
    n. Interval 14 → -0.75 <= avaluació > -1
    o. Interval 15 → -1 <= avaluació > -1.25
    p. Interval 16 → -1.25 <= avaluació > -1.5
    q. Interval 17 → -1.5 <= avaluació > -2.5
    r. Interval 18 → -2.5 <= avaluació > -3.5
    s. Interval 19 → -3.5 <= avaluació > -4.5
    t. Interval 20 → avaluació <= -4.5
    u. Interval 21 → avaluació = 0

Igual que amb les codificacions de sortida, un cop tinguem la xarxa dissenyada, es faran proves amb les diferents codificacions per a comprovar quina presenta millor rendiment.

3.5 Construcció de la xarxa neuronal

En aquest apartat s'inicia el procés de construcció de la xarxa que processarà totes les dades i calcularà el resultat final. Com s'ha dit reiteradament, no es pot saber amb certesa quin disseny de la xarxa serà el més precís, ja que és un problema que no ha estat tractat anteriorment i pot ser que la xarxa més ideal sigui una combinació entre les infinites que existeixen.

Per a crear la xarxa, igual que per a la resta de codi es fa servir Python, en concret les biblioteques TensorFlow i Keras.

TensorFlow és una plataforma de codi obert desenvolupada per Google que s'utilitza àmpliament en l'aprenentatge automàtic i el deeplearning. TensorFlow permet crear i entrenar models d'aprenentatge automàtic de manera eficient, amb una gran quantitat d'eines i funcionalitats disponibles per a simplificar el procés. És altament escalable, cosa que significa que pot manejar grans conjunts de dades i es pot executar en una àmplia varietat de maquinari.

D'altra banda, Keras és una biblioteca de Python d'alt nivell per a la creació de xarxes neuronals artificials. Keras s'executa sobre TensorFlow i proporciona una API simplificada i fàcil d'usar per a crear i entrenar models d'aprenentatge profund. Keras també admet una varietat d'arquitectures de xarxes neuronals, incloent-hi xarxes convolucionals, xarxes recurrents i xarxes de retroalimentació.

En resum, TensorFlow proporciona una base sòlida per a l'aprenentatge profund, mentre que Keras simplifica el procés de construcció i entrenament de models d'aprenentatge profund en Python. Junts, són una combinació poderosa per a desenvolupar aplicacions d'aprenentatge profund.

En aquest treball, he usat Keras per a crear i entrenar el perceptró multicapa que utilitzaré. Per a crear aquesta xarxa Keras l'estructura en capes, aquestes es divideixen en tres tipus:

- D'entrada: Com el nom indica és la primera capa que trobem i és per on es reben les dades. El nombre de neurones d'aquesta capa bé donat per les dades d'entrada.
- Ocultes: Són les capes que es troben entre les dades d'entrada i sortida. Són les capes que formen la complexitat de la xarxa i les que segons el nombre de capes i neurones per capa fan que la xarxa pugui comprendre millor les dades i, per tant, ser més precisa.
- De sortida: És l'última capa d'una xarxa neuronal. Igual que la capa de sortida, aquesta capa té el nombre de neurones fixat per la codificació de les dades de sortida.

Per a poder trobar la xarxa que millor s'ajusti al problema que estem tractant, he fet el mateix que amb les dades d'entrada i les de sortida, tenir diverses codificacions i provar posteriorment el seu funcionament per a veure quina presenta la millor precisió. Les arquitectures de xarxa triades han estat les següents:

1. Primer una xarxa només amb una capa d'entrada i una de sortida, tot i que, segons la teoria no ha de ser l'òptim, val la pena provar-ho per a comprovar-ho.
    i. Xarxa 1
        - Capa d'entrada: 64 o 67 neurones (segons la codificació triada)
        - Capa sortida: Entre 16 i 21 neurones (segons la codificació de sortida)

2. En les següents xarxes, s'afegeixen capes ocultes. El que se sol fer amb aquest tipus de xarxes és afegir alguna capa amb més neurones que la capa d'entrada, aquesta capa està pensada per a poder entendre i captar les relacions que existeixin entre les diferents neurones de la capa d'entrada. Les arquitectures de xarxa pensades han estat les següents:
    i. Xarxa 2
        - Capa d'entrada: 64 o 67 neurones (segons la codificació triada)
        - Capa oculta 1: 128 neurones
        - Capa sortida: Entre 16 i 21 neurones (segons la codificació de sortida)
    ii. Xarxa 3
        - Capa d'entrada: 64 o 67 neurones (segons la codificació triada)
        - Capa oculta 1: 128 neurones
        - Capa oculta 2: 64 neurones
        - Capa sortida: Entre 16 i 21 neurones (segons la codificació de sortida)
    iii. Xarxa 4
        - Capa d'entrada: 64 o 67 neurones (segons la codificació triada)
        - Capa oculta 1: 128 neurones
        - Capa oculta 2: 64 neurones
        - Capa oculta 3: 32 neurones
        - Capa oculta 4: 256 neurones
        - Capa sortida: Entre 16 i 21 neurones (segons la codificació de sortida)
    iv. Xarxa 5
        - Capa d'entrada: 64 o 67 neurones (segons la codificació triada)
        - Capa oculta 1: 256 neurones
        - Capa oculta 2: 128 neurones
        - Capa oculta 3: 64 neurones
        - Capa oculta 4: 32 neurones
        - Capa sortida: Entre 16 i 21 neurones (segons la codificació de sortida)
    v. Xarxa 6
        - Capa d'entrada: 64 o 67 neurones (segons la codificació triada)
        - Capa oculta 1: 128 neurones
        - Capa oculta 2: 256 neurones
        - Capa oculta 3: 64 neurones
        - Capa oculta 4: 32 neurones
        - Capa oculta 3: 64 neurones
        - Capa oculta 4: 16 neurones
        - Capa sortida: Entre 16 i 21 neurones (segons la codificació de sortida)
    vi. Xarxa 7
        - Capa d'entrada: 64 o 67 neurones (segons la codificació triada)
        - Capa oculta 1: 256 neurones
        - Capa sortida: Entre 16 i 21 neurones (segons la codificació de sortida)

A cada neurona cal aplicar-li una funció d'activació per a produir una sortida no lineal. La funció d'activació s'aplica a la suma ponderada de les entrades a la neurona, més el biaix, i produeix la sortida de la neurona. A causa de les característiques de classificació de la xarxa que s'està buscant construir, la funció d'activació triada ha estat la sigmoide, que permet establir a cada neurona de sortida un valor binari per a identificar si l'entrada pertany a aquell grup o no.

Aquestes seran totes les arquitectures que seran comparades i posades a test per a veure quina obté millors resultats. En cas de tenir dues xarxes que tenen un rendiment similar, es triarà la més senzilla, ja que vol dir que serà més veloç a l'hora de fer càlculs.

3.6 Selecció de les codificacions i validació de model

En aquest últim apartat es documenten totes les proves necessàries per a poder seleccionar el millor disseny de la xarxa, per a fer això, el que es fa és primer entrenar les xarxes.

Aquest entrenament consisteix a donar-li a la xarxa un nombre elevat d'entrades i la sortida esperada amb cadascuna d'aquestes. Aleshores, la xarxa recorre aquest conjunt de dades diverses vegades i, en cada iteració, va ajustant els pesos de les neurones per a poder afinar el resultat i fer-se cada cop més precisa. Al final d'aquest entrenament es proporciona el percentatge d'encert al qual ha arribat la xarxa.

Un cop entrenada, cal comprovar que la xarxa no ha patit un sobre-ajust. El sobre-ajust en l'entrenament (també conegut com a overfitting en anglès) és un problema en el qual el model s'ajusta massa a les dades d'entrenament i perd la capacitat de generalitzar i fer prediccions precises en noves dades. En altres paraules, quan un MLP es sobre-ajusta, aprèn patrons i relacions específiques en les dades d'entrenament que no són rellevants per a dades noves i no vistes abans. Això pot ocórrer si el model és massa complex en relació amb la grandària de les dades d'entrenament, o si el model s'entrena massa temps amb les mateixes dades.

Per a comprovar que realment el model està entrenat es comprova que la sortida és correcta amb un segon joc de dades, les dades de test, aquestes dades han de ser diferents de les dades d'entrenament, per tant, són completament noves i ens ajuden a verificar la xarxa.

Per a comprovar quina xarxa és la millor, el que faré serà comparar el percentatge d'encert (tant amb les dades d'entrenament, com amb les dades de test) de totes les xarxes per a poder trobar la més adient. Per a fer-ho, s'han utilitzat 2 milions de dades per a l'entrenament i 100 mil dades per a la verificació.

Abans de començar amb les proves cal configurar els paràmetres d'aprenentatge per a optimitzar la xarxa, això es coneix com compilar la xarxa. Per a fer la compilació, cal ajustar 3 paràmetres principals:

- Loss (pèrdua): És una funció que s'utilitza per a mesurar la discrepància entre les prediccions del model i les etiquetes reals de les dades d'entrenament. L'objectiu del model és minimitzar aquesta funció de pèrdua durant l'entrenament per a millorar la precisió de les prediccions. En el cas d'un MLP classificador, l'elecció de la funció de pèrdua depèn del tipus de problema de classificació que s'estigui abordant. En el cas que s'està tractant, a l'estar classificant segons diferents paràmetres, la funció de pèrdua elegida és la "categorical_crossentropy". L'entropia creuada categòrica mesura la discrepància entre la distribució de probabilitat predita pel model i la distribució de probabilitat real de les etiquetes de les dades d'entrenament. Aquesta funció de pèrdua és adequada per a problemes de classificació perquè penalitza fortament les prediccions incorrectes i afavoreix les prediccions correctes amb alta confiança.

- Optimizer (optimitzador): És un algorisme que s'utilitza per a ajustar els pesos de les neurones en cada iteració durant l'entrenament del model amb la finalitat de minimitzar la funció de pèrdua.
  Keras proporciona varis optimitzadors predefinits, com Adam, SGD (descens de gradient estocàstic) i RMSprop, entre altres. L'elecció de l'optimitzador adequat depèn del problema específic que s'està abordant, així com de les característiques del conjunt de dades i de l'arquitectura del model. Per a saber quin optimitzador utilitzar, s'han fet proves executant una combinació arbitrària d'entrada i sortida amb tots els combinadors que s'ofereixen. Com tots donaven el mateix resultat positiu, s'ha triat "Adam", ja que, en general, l'optimitzador Adam és una bona elecció per a la majoria dels problemes d'aprenentatge profund, incloent-hi la classificació, la detecció d'objectes i el processament del llenguatge natural. És un optimitzador adaptatiu que ajusta la taxa d'aprenentatge dels paràmetres de manera automàtica, la qual cosa ajuda a accelerar la convergència i a evitar que el model s'embussi en mínims locals.

- Metrics (mètriques): Són funcions que s'utilitzen per a avaluar el rendiment del model durant i després de l'entrenament. Les mètriques s'utilitzen per a mesurar la precisió del model en un conjunt de dades de prova i per a comparar diferents models entre si. Dintre de totes les que proporciona Keras, la triada ha estat la precisió, ja que, com és obvi, el que volem és que el model sigui el més encertat possible.

Un cop configurada la xarxa, es poden començar les proves. El que es farà serà provar totes les combinacions possibles. Per a fer això, es prepara un bucle on es creuen les 7 arquitectures de xarxa pensades amb les 7 codificacions de sortides, això per a cadascuna de les dues combinacions d'entrada. En aquesta part, compararem les que presenten millors percentatges amb les dades d'entrenament i amb les de test. Les taules completes amb la informació de tots els tests es poden veure a l'annex A.

Les millors combinacions per a la codificació amb tota la informació a les caselles han estat:

Arquitectura de xarxa	Funció d'avaluació	Resultat entrenament	Resultat test
Xarxa 1	avaluació 1	91.15%	89.53%
Xarxa 2	avaluació 1	91.15%	89.53%
Xarxa 3	avaluació 1	91.15%	89.53%
Xarxa 4	avaluació 1	91.15%	89.53%
Xarxa 5	avaluació 1	91.15%	89.53%
Xarxa 6	avaluació 1	91.15%	89.53%
Xarxa 7	avaluació 1	91.15%	89.53%
Xarxa 1	avaluació 6	91.15%	89.53%
Xarxa 2	avaluació 6	91.15%	89.53%
Xarxa 3	avaluació 6	91.15%	89.53%
Xarxa 4	avaluació 6	91.15%	89.53%
Xarxa 5	avaluació 6	91.15%	89.53%
Xarxa 6	avaluació 6	91.15%	89.53%
Xarxa 7	avaluació 6	91.15%	89.53%

Taula 1 Resultats de les millors codificacions per a la codificació amb la informació a les caselles

Per a la codificació amb les dades de les peces i la informació de la partida separades:

Arquitectura de xarxa	Funció d'avaluació	Resultat entrenament	Resultat test
Xarxa 2	avaluació 1	91.15%	89.53%
Xarxa 3	avaluació 1	91.15%	89.53%
Xarxa 4	avaluació 1	91.15%	89.53%
Xarxa 5	avaluació 1	91.15%	89.53%
Xarxa 6	avaluació 1	91.15%	89.53%
Xarxa 7	avaluació 1	91.15%	89.53%
Xarxa 2	avaluació 6	91.15%	89.53%
Xarxa 3	avaluació 6	91.15%	89.53%
Xarxa 4	avaluació 6	91.15%	89.53%
Xarxa 5	avaluació 6	91.15%	89.53%
Xarxa 6	avaluació 6	91.15%	89.53%
Xarxa 7	avaluació 6	91.15%	89.53%

Taula 2 Resultats de les millors codificacions per a la codificació amb camps separats

Com es pot veure s'ha obtingut amb moltes codificacions un valor d'encert amb les dades d'entrenament del 91,15%, mentre que amb les dades de test un 89,53%. Això és un resultat molt satisfactori, ja que es troba en un rang molt elevat d'encert i, tenint en compte que tampoc existeixen més xarxes com a aquesta per a comparar, podem dir que és un resultat satisfactori. Ara toca classificar les xarxes i triar l'òptima.

El primer que es compararà per eliminar la meitat de codificacions serà la codificació d'entrada. Com que els percentatges es troben en igualtat de condicions, el que es farà serà comparar per dimensions (ja que entre més petites, més velocitat) i per utilitat.

Segons aquests criteris, l'entrada més petita és la codificació amb totes les dades al tauler, ja que només necessita 64 neurones per a l'entrada i, a més, és la que considero que representa les dades d'una forma més compacta. Aquest criteri ens ha reduït les xarxes a 14.

A continuació, de les 14 restants, podem veure que són totes les xarxes amb la codificació de sortida 1 o 6. Si tornem a veure com eren aquestes avaluacions:

La 1 és de la següent manera:

a. El primer indicarà el signe de l'avaluació Positiu/Zero (0) o negatiu (1)
b. Interval 1 → avaluació < 0.5
c. Interval 2 → 0.5 <= avaluació >= < 1.5
d. Interval 3 → 1.5 <= avaluació < 2.5
e. Interval 4 → 2.5 <= avaluació < 3.5
f. Interval 5 → 3.5 <= avaluació < 4.5
g. Interval 6 → 4.5 <= avaluació

Mentre que la 6:

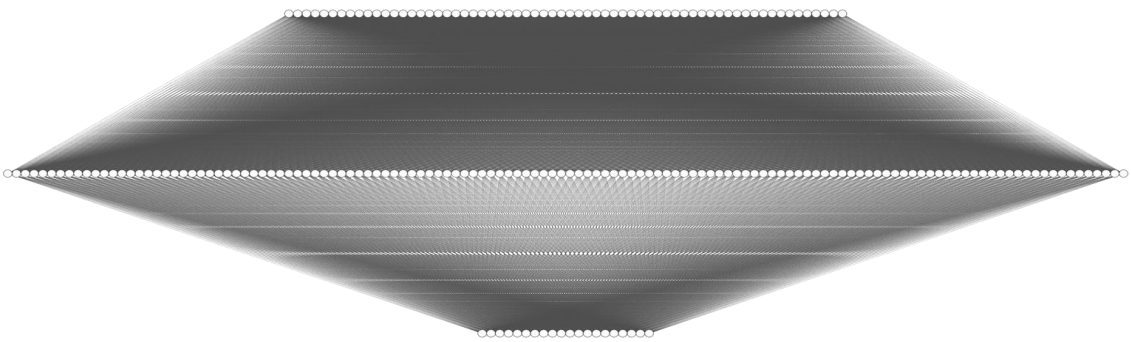
a. Interval 1 → avaluació < 0.25
b. Interval 2 → 0.25 <= avaluació < 0.5
c. Interval 3 → 0.5 <= avaluació < 0.75
d. Interval 4 → 0.75 <= avaluació < 1
e. Interval 5 → 1 <= avaluació < 1.25
f. Interval 6 → 1.25 <= avaluació < 1.5
g. Interval 7 → 1.5 <= avaluació < 2.5

h. Interval 8 → 2.5 <= avaluació < 3.5
i. Interval 9 → 3.5 <= avaluació < 4.5
j. Interval 10 → avaluació >= 4.5
k. Interval 11 → -0.25 <= avaluació < 0
l. Interval 12 → -0.25 <= avaluació > -0.5
m. Interval 13 → -0.5 <= avaluació > -0.75
n. Interval 14 → -0.75 <= avaluació > -1
o. Interval 15 → -1 <= avaluació > -1.25
p. Interval 16 → -1.25 <= avaluació > -1.5
q. Interval 17 → -1.5 <= avaluació > -2.5
r. Interval 18 → -2.5 <= avaluació > -3.5
s. Interval 19 → -3.5 <= avaluació > -4.5
t. Interval 20 → avaluació <= -4.5

En aquest cas, no podem seguir el criteri de la que menys dimensions tingui, ja que també ens interessa veure quines dades representen. Com es pot veure clarament, la codificació 6 presenta molts més intervals de sortida, cosa que fa que la codificació sigui molt més precisa a l'hora d'indicar quina és l'avaluació de la posició. Tenint en compte això, finalment la codificació triada ha estat la 6.

Un cop seleccionada la codificació d'entrada i de sortida, es redueix l'elecció entre les 7 arquitectures de xarxa plantejades inicialment. Sobre l'arquitectura, el fet de tenir més o menys capes només ajuda a entendre millor les relacions del conjunt de dades d'entrada, per tant, si totes les configuracions presenten una igual precisió, és fàcilment comprensible que la millor és la més petita, ja que aquesta presentarà una complicació en el càlcul més reduïda, ja que el nombre total de neurones per les quals haurà de passar la informació serà menor. Les xarxes que presenten menor mida són la 6 i la 1, ja que la 1 no presenta capes ocultes, això podria significar una menor eficiència en algunes posicions, per tant, per la petita diferència que presentarien en temps de resposta i mida, és més adient seleccionar l'arquitectura 6, ja que aquesta possible correcció en algunes posicions (tot i que no s'hagi donat el cas en la mostra) és preferible a l'ínfima major velocitat que presenta l'arquitectura 1.

Finalment, l'estructura de la nostra xarxa MLP, és la codificació d'entrada amb totes les peces del tauler (64 neurones), amb una capa oculta més amplia per a comprendre bé les relacions que pugui aparèixer entre les variables (128 neurones) i la codificació de sortida 6 (20 neurones), ja que és la que més informació proporciona per al problema plantejat. Aquesta xarxa i les seves connexions es poden veure representades en la figura següent:



Il·lustració 5. Representació de la xarxa final seleccionada

4. Xarxa per a valorar moviments

4.1 Nou disseny

En aquesta nova fase la intenció és modificar el model per apropar-nos més a l'objectiu plantejat. El següent pas és millorar la xarxa per a poder avaluar moviments com a tal i no posicions. Per a poder iniciar la creació d'aquesta nova xarxa, cal primer entendre que és un moviment en escacs, ja que no deixa de ser res més que el canvi d'una posició a una altra, per tant, podem avaluar un moviment com la diferència entre l'avaluació de la posició inicial amb la de la posició final després de realitzar el moviment. Amb aquesta premissa clara, es pot començar a plantejar el nou disseny de la xarxa.

Aleshores, podem veure que per a avaluar un moviment necessitarem saber els següents aspectes sobre aquest:

- Fen de la posició inicial
- Fen de la posició final
- Diferència d'avaluació entre les dues posicions

Amb aquests paràmetres ja es poden anar plantejant els primers canvis a realitzar a la xarxa. A l'entrada, caldrà doblar les neurones necessàries per a introduir les dades, ja que necessitem dues posicions com la del MLP anterior per al nou MLP. A la sortida, el rang de sortides pot ser el mateix, ja que segueix representant una avaluació, tot i que en aquest cas, serà la del moviment. Caldrà afegir doncs una nova neurona a la sortida en la qual representarem si aquest moviment es troba dintre del rang de mat forçat (si els dos jugadors juguen correctament el mat és inevitable), així també s'afegeix nova informació que no estava inclosa en la xarxa anterior.

4.2 Creació de la base de dades

Igual que en la xarxa anterior, en aquesta també cal generar una base de dades, ja que no podem fer servir la que teníem perquè les dades d'entrada són noves. Per a generar el nou CSV amb la informació que servirà per a alimentar la xarxa farem servir també el conjunt de partides (en format PGN) que ja tenim amb anterioritat, però ara, el que caldrà es recorre cada partida i guardar la informació del nou moviment. Per fer això, vaig crear un script en Python que comença amb la posició inicial de les peces, i per a cada moviment, es genera la posició en FEN del tauler en l'estat inicial i després de fer el moviment. A més, per a aquests dos estats del tauler es genera la seva avaluació (també fent servir stockfish) i s'emmagatzema el valor segons si les posicions estan en mat forçat (que es representa com el nombre de moviments per a acabar en escac i mat, en negatiu si es per negres o positiu si es per blanc) o no seguint el criteri següent:

- Si a la posició 1 tenim avaluació numèrica i a la 2 també:
  L'avaluació del moviment serà definida per l'operació:

$$avaluació\ final = avaluació\ 2 - avaluació\ 1$$

  El resultat d'aquesta fórmula ens mostrarà com ha afectat aquest moviment a l'avaluació de la partida.

- Si a la posició 1 tenim avaluació numèrica i a la 2 hi ha mat forçat:

Això voldrà dir que el moviment ha portat a ser mat, per tant, l'avaluació es calcularà de la forma següent:

$$avaluació\ final = \frac{1000}{avaluació\ 2} - avaluació\ 1 \qquad (1)$$

El que s'està fent amb la fórmula és equiparar una posició que ja és mat forçat amb una que no ho és. Per fer això, se li dona un valor molt elevat (més elevat entre més s'apropa a ser escac i mat) a la posició que correspondria el mat i se li resta el valor de la posició inicial, per a poder representar així el canvi que ha generat aquest moviment ha estat molt gran.

- Si a la posició 1 tenim mat forçat i a la 2 avaluació numèrica:
  Això voldrà dir que el moviment, com en el cas anterior, ha de tenir un valor numèric elevat que mostri el canvi radical que ha suposat per la posició, aquest és definit per la fórmula següent:

$$avaluació\ final = avaluació\ 2 - \frac{1000}{avaluació\ 1} \qquad (2)$$

- Si a la posició 1 tenim mat forçat i a la 2 també:
  L'avaluació del moviment serà definida per l'operació:

$$avaluació\ final = avaluació\ 2 - avaluació\ 1 \qquad (3)$$

El resultat d'aquesta fórmula ens donarà el nombre de moviments en què ha disminuït o ha augmentat el mat forçat que hi ha en la posició.

Cal entendre que no s'ha d'interpretar un moviment com a bo o dolent segons el signe que tingui, ja que això només representa com afecta l'estat de la partida (negatiu van guanyant negres, positiu blanques), la condició d'aquests moviments (bo o dolent) serà definida per qui fa el moviment (un moviment amb una avaluació positiva serà un bon moviment si el fan les blanques, però un mal moviment si el fan les negres).

En la informació de la base de dades també generarem el bit que ens dirà si la posició és mat o no i que, com s'ha mencionat anteriorment, servirà per a poder generar les dades de sortida de la xarxa junt amb les avaluacions. Aquest bit serà el quart paràmetre de les dades que es guardaran al CSV i vindrà generat per el FEN de la segona posició, si aquesta és mat, el bit estarà a 1, mentre que si no ho és, estarà a 0.

Per últim, com generar les dades de la base de dades és una funció lenta pel que fa a temps perquè és costosa computacionalment, aprofitarem la generació d'aquesta informació per a afegir informació que pot ser necessària en un futur. Així que, per últim, la informació que s'afegirà serà la de la peça que ha estat moguda, es guardarà com un valor numèric entre 1 i 6 amb el següent criteri:

- El peó serà 1

- El cavall serà 2
- L'alfil serà 3
- La torre serà 4
- La reina serà 5
- El rei serà 6

Un cop definits tots els camps que formaran part de la base, només cal programar l'script. Per a fer-ho, s'ha modificat el que es feia servir en la generació de la base anterior amb els càlculs per a trobar les noves dades. A més, pel fet que l'anterior va tenir una durada d'execució molt elevada i aquesta nova informació presentava més complicacions en l'àmbit computacional, s'ha afegit l'ús de threads per al tractament dels fitxers PGN i els càlculs corresponents així, es permet augmentar la velocitat de generació de la base de dades en uns 4 cops. Finalment, després del tractament de tots els fitxers, s'ha creat la base de dades (en format CSV) amb la nova informació. Dels fitxers mencionats en l'apartat anterior que contenien les partides en PGN s'han extret 2.856.516 posicions per a entrenar la xarxa.

4.3 Procediment de construcció, d'entrenament i validació del model

Un cop més, cal validar la xarxa. Com ja s'ha demostrat pel MLP anterior quines eren les millors codificacions per a l'entrada i la sortida, i degut també a què s'ha augmentat el nombre de neurones d'aquestes, per al nou MLP només cal comprovar que s'ha triat l'arquitectura de xarxa correcta.
Per a això, es dissenyen noves arquitectures basades en les anteriors, però modificant les capes d'entrada i sortida i obtenim les següents xarxes a provar:

i. Xarxa 1
   - Capa d'entrada: 128 neurones
   - Capa sortida: 21 neurones

ii. Xarxa 2
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa sortida: 21 neurones

iii. Xarxa 3
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa oculta 2: 128 neurones
   - Capa sortida: 21 neurones

iv. Xarxa 4
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa oculta 2: 128 neurones
   - Capa oculta 3: 64 neurones
   - Capa oculta 4: 256 neurones
   - Capa sortida: 21 neurones

v. Xarxa 5
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa oculta 2: 128 neurones
   - Capa oculta 3: 64 neurones
   - Capa oculta 4: 32 neurones
   - Capa sortida: 21 neurones

vi. Xarxa 6
- Capa d'entrada: 128 neurones
- Capa oculta 1: 128 neurones
- Capa oculta 2: 256 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 32 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 16 neurones
- Capa sortida: 21 neurones

vii. Xarxa 7
- Capa d'entrada: 128 neurones
- Capa oculta 1: 512 neurones
- Capa sortida: 21 neurones

Finalment, es torna comprova la seva efectivitat per a la codificació d'entrada i sortida triada. Els resultats han estat els següents:

Arquitectura de xarxa	Funció d'avaluació	Resultat entrenament	Resultat test
Xarxa 1	avaluació 6	99.46%	99.54%
Xarxa 2	avaluació 6	99.46%	99.54%
Xarxa 3	avaluació 6	99.46%	99.54%
Xarxa 4	avaluació 6	99.46%	99.54%
Xarxa 5	avaluació 6	99.46%	99.54%
Xarxa 6	avaluació 6	99.46%	99.54%
Xarxa 7	avaluació 6	99.46%	99.54%

Taula 3 Resultats òptims per a la codificació final de la xarxa inicial

Com es pot apreciar, totes les xarxes han tingut una precisió molt elevada, millorant l'aconseguida per la primera xarxa, això és degut al fet que en augmentar el nombre d'entrades, la xarxa es capaç de comprendre amb més facilitat la relació entre aquestes i millorar els càlculs dels pesos fins a tenir precisions molt altes. Pel que fa a l'arquitectura definitiva, seguint els mateixos criteris que amb la xarxa anterior, la triada ha estat l'opció 2. Així que, en resum, la segons xarxa queda formada pels paràmetres següents:

- Codificació d'entrada: 128 neurones. 64 per a representar les caselles de la primera posició i 64 per a representar les caselles del taulell amb la segona posició.
- Arquitectura de la xarxa:
  - Capa d'entrada: 128 neurones
  - Capa oculta 1: 256 neurones
  - Capa sortida: 21 neurones

- Codificació de sortida
  - Interval 1 → avaluació < 0.25
  - Interval 2 → 0.25 <= avaluació < 0.5
  - Interval 3 → 0.5 <= avaluació < 0.75
  - Interval 4 → 0.75 <= avaluació < 1
  - Interval 5 → 1 <= avaluació < 1.25
  - Interval 6 → 1.25 <= avaluació < 1.5
  - Interval 7 → 1.5 <= avaluació < 2.5
  - Interval 8 → 2.5 <= avaluació < 3.5
  - Interval 9 → 3.5 <= avaluació < 4.5

- Interval 10 → avaluació >= 4.5
- Interval 11 → -0.25 <= avaluació < 0
- Interval 12 → -0.25 <= avaluació > -0.5
- Interval 13 → -0.5 <= avaluació > -0.75
- Interval 14 → -0.75 <= avaluació > -1
- Interval 15 → -1 <= avaluació > -1.25
- Interval 16 → -1.25 <= avaluació > -1.5
- Interval 17 → -1.5 <= avaluació > -2.5
- Interval 18 → -2.5 <= avaluació > -3.5
- Interval 19 → -3.5 <= avaluació > -4.5
- Interval 20 → avaluació <= -4.5
- Neurona de mat

5. Xarxa explicant moviments

5.1 Nou disseny plantejat

En aquest nou apartat es modifica la xarxa anterior per a afegir nous paràmetres que permetran entendre la posició resultant després dels moviments i, per tant, què aconsegueix aquest moviment. Per a fer aquesta explicació he triat uns paràmetres inicials que poden ser senzills d'entendre per a qualsevol persona que es s'estigui iniciant en el món dels escacs, ja que, com he mencionat anteriorment, la intenció d'aquesta xarxa és explicar els moviments a un nivell bàsic perquè una persona principiant pugui entendre que està fent. Per a fer això, els paràmetres triats han estat:

- Control de caselles: S'entén el control de caselles com el nombre de caselles que veu una peça des d'una posició en concret.
- Captura: Permet saber si un moviment ha estat realitzat per a capturar una peça.
- Escac: Permet saber si un moviment ha estat forçat o no, ja que és per a defensar un escac
- Posició en el taulell: Permet saber la posició que ocupa la peça respecte al centre del taulell.

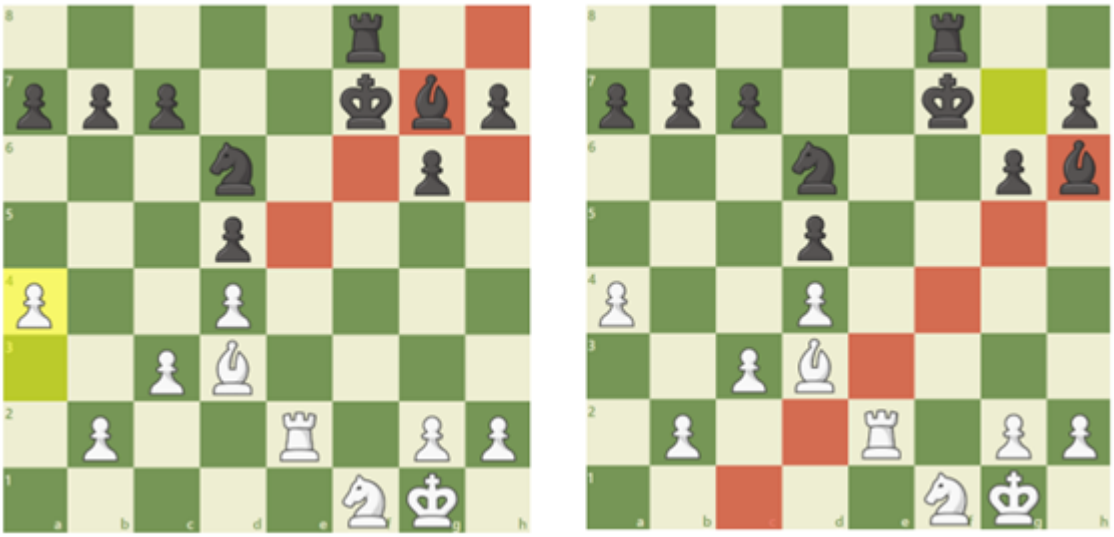
Per a afegir cadascun d'aquests camps primer cal veure com calcular-los i codificar-los per a poder generar les sortides esperades i així entrenar la xarxa. La codificació triada ha estat la següent:

Control de caselles

Aquest camp pot tenir dos valors:

- Positiu: La peça moguda controla (veu des de la seva posició) el mateix nombre de peces o més des de la seva nova posició respecta l'anterior posició abans del moviment.
- Negatiu: La peça moguda controla un nombre inferior de caselles des de la seva casella actual respecte la posició que tenia abans del moviment.

Per a fer el càlcul d'aquest paràmetre, s'utilitza el fen de les posicions inicials i final per a poder trobar on es troba la peça abans i després de fer el moviment, un cop obtingudes les dues caselles, utilitzant la llibreria Python-chess, es genera la posició i amb la funció *board.attackers()* s'obtenen el nombre de caselles atacades des de les posicions donades, un cop obtingudes el nombre de caselles, amb una simple resta es pot obtenir un valor representant el control de les caselles del moviment. Aquest més gran o igual a 0 si el control de caselles ha estat positiu o més petit 0 si el control de caselles ha estat negatiu. Per a representar visualment aquest càlcul, es pot veure en la imatge següent:



Il·lustració 6. Posició inicial i final després del moviment d'un alfil

Com es pot veure, en la primera posició (esquerra) l'alfil es troba a g7 i controla 5 caselles (marcades en vermell), posteriorment, després de moure'l a h6, controla 6 caselles. La valoració del control de caselles és d'1, per tant, aquest moviment és positiu.

Captura

Per a fer les comprovacions respecte a una captura, no cal accedir a cap llibreria externa, per a comprovar si una peça ha estat capturada, només cal comprovar el nombre de lletres que apareixen en el FEN de la posició inicial i final, si aquest es veu reduït, això indica que una peça ha desaparegut, per tant, només pot venir donat perquè hi ha hagut una captura.

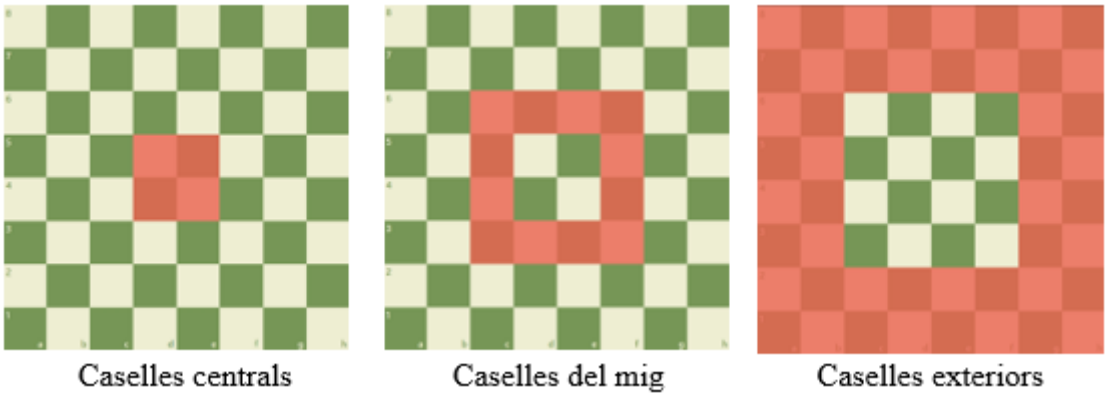
Escac

Per a fer la comprovació d'escac, se simula la posició inicial amb la llibreria Python-chess i, aplicant la funció *is_check()* a aquesta posició ens retorna un booleà conforma ens trobem en escac o no.

Posició del taulell

Com s'ha comentat a l'inici, les caselles d'un tauler es poden classificar en tres categories (mostrats també a la il·lustració 7):

- Centre: Les quatre caselles del mig (D4,E4,D5,E5)
- Mig: Les dues files i columnes de caselles que envolten les caselles centrals.
- Exterior: Les caselles que estan en contacte directe amb el límit del taulell



Il·lustració 7. Nom de les caselles segons la seva posició al taulell

Per a implementar la classificació de les posicions el que es fa és comparar les dues posicions FEN donades i trobar quina de les lletres (que representa una peça) s'ha mogut. Per a posicions d'enroc on es mouen dues peces, es compara la peça moguda amb la calculada prèviament (juntament amb les posicions FEN i l'avaluació) a la base de dades mencionada a l'apartat 4.2. Un cop obtinguda la peça que s'ha mogut, es mira la posició d'aquesta en el taulell i així es poden obtenir les coordenades per a saber en quina categoria es troba.

Un cop identificats aquests nous paràmetres afegirem les neurones necessàries a la sortida de la xarxa. La nova informació vindrà codificada de la manera següent:

- 1 neurona pel control:
  - Control positiu: La sortida serà 0.
  - Control negatiu: La sortida serà 1.
- 1 neurona per captura
  - No existeix captura: Sortida a 0.
  - Hi ha captura: Sortida a 1.
- 1 neurona per escac
  - No hi ha escac: Sortida a 0.
  - Existeix escac: Sortida a 1.
- 3 neurones per la posició del taulell
  - Neurona centre: Valor a 1 si el moviment és a l'exterior o 0 en cas contrari.
  - Neurona mig: Valor a 1 si el moviment és a l'exterior o 0 en cas contrari.
  - Neurona externa: Valor a 1 si el moviment és a l'exterior o 0 en cas contrari.

Tenint la nova sortida codificada, i sense necessitat de canviar l'entrada, es tenen els paràmetres necessaris per a poder començar a entrenar la xarxa.

5.2 Procediment de construcció, d'entrenament i validació del model

Com s'ha de fer en totes les xarxes, si canvia el nombre de neurones de sortida, cal fer modificacions a l'estructura per a intentar trobar la que millor resultats aporti i ajudi a la interpretació de les connexions existents entre les neurones d'entrada. Com aquesta nova xarxa presenta un major nombre de neurones de sortida, es proven noves codificacions per a, un cop més, trobar l'òptima i aconseguir millorar el percentatge d'encerts. Les noves codificacions provades per a la nova xarxa han estat:

i. Xarxa 1

- Capa d'entrada: 128 neurones
- Capa sortida: 27 neurones

ii. Xarxa 2
- Capa d'entrada: 128 neurones
- Capa oculta 1: 256 neurones
- Capa sortida: 27 neurones

iii. Xarxa 3
- Capa d'entrada: 128 neurones
- Capa oculta 1: 256 neurones
- Capa oculta 2: 128 neurones
- Capa sortida: 27 neurones

iv. Xarxa 4
- Capa d'entrada: 128 neurones
- Capa oculta 1: 256 neurones
- Capa oculta 2: 128 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 256 neurones
- Capa sortida: 27 neurones

v. Xarxa 5
- Capa d'entrada: 128 neurones
- Capa oculta 1: 256 neurones
- Capa oculta 2: 128 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 32 neurones
- Capa sortida: 27 neurones

vi. Xarxa 6
- Capa d'entrada: 128 neurones
- Capa oculta 1: 128 neurones
- Capa oculta 2: 256 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 32 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 16 neurones
- Capa sortida: 27 neurones

vii. Xarxa 7
- Capa d'entrada: 128 neurones
- Capa oculta 1: 512 neurones
- Capa sortida: 27 neurones

Finalment, es torna comprova la seva efectivitat per a la codificació d'entrada i sortida triada. Els resultats han estat els següents:

Arquitectura de xarxa	Resultat entrenament	Resultat test
Xarxa 1	96,35%	96,42%
Xarxa 2	96,35%	96,42%
Xarxa 3	96,35%	96,42%
Xarxa 4	96,35%	96,42%

Xarxa 5	96,35%	96,42%
Xarxa 6	96,35%	96,42%
Xarxa 7	96,35%	96,42%

Taula 4 Resultats òptims per a la codificació final de la xarxa amb explicacions

Com es pot veure, totes les xarxes presenten un resultat d'entrenament i de test superior al 96%. Aquesta petita baixada del percentatge d'encert respecte a la xarxa anterior és deguda a l'increment de les variables de sortida , cosa que fa que la xarxa hagi d'interpretar moltes més connexions entre les dades i pot portar a cometre algun error més. Tot i haver disminuït el percentatge respecte de l'anterior, es pot afirmar que un percentatge superior al 80% representa una xarxa ben configurada i preparada per a fer la seva feina, per tant, es pot afirmar que aquests resultats (molt superiors) són més que satisfactoris per a la xarxa creada. Seguint el criteri de les xarxes anteriors, dintre de les configuracions amb millor percentatge, es tria la que millors qualitats aporta (velocitat de processament i menor complexitat), per tant, l'arquitectura seleccionada ha estat l'explicada en la xarxa 2.

Finalment, en tenir ja tota l'estructura de la xarxa muntada i saber quina presenta millors resultats, es repeteix l'entrenament i es guarda, utilitzant una funció de la mateixa llibreria Keras, el valor resultant dels pesos en un fitxer h5 (aquest format és una manera d'organitzar i emmagatzemar dades en una estructura jeràrquica similar a un sistema d'arxius.). Aquest fitxer ens servirà per a, posteriorment, carregar la xarxa de manera ràpida i poder fer prediccions encertades sense haver de tornar a entrenar-la de zero.

6. Ampliació dels algorismes i noves xarxes d'explicació

6.1 Nova xarxa d'explicacions

Com s'ha mostrat en els resultats de la xarxa anterior, en augmentar el nombre de neurones de sortida, el percentatge d'encert es veu afectat, això ha portat a haver de trobar una altra forma d'interpretar més paràmetres, ja que, entre millor explicació d'una posició es vulgui donar, més atributs d'aquesta s'hauran de tenir en compte, per tant, la forma de representar noves variables interessants per a l'explicació ha de fer-se d'una altra manera, no és, pel que fa a rendiment, el més idoni seguir-ho fent amb la mateixa xarxa

És per això que per a representar nous paràmetres que no són fàcils calcular al moment, es pot utilitzar la mateixa codificació de la posició en una nova xarxa especialment dissenyada per a aquest paràmetre.

D'aquí surt la idea de seleccionar un nou aspecte dels escacs per a ser interpretat per a aquesta nova xarxa que, junt amb la inicial, ens ajudaran a poder explicar millor cada moviment realitzat. El paràmetre triat ha estat les columnes lliures.

En els escacs, les columnes lliures es refereixen a les columnes del tauler que no estan obstruïdes per peces pròpies o de l'oponent. Aquestes columnes exerceixen un paper crucial en el joc, ja que brinden una sèrie d'avantatges estratègics i tàctiques. Les raons per les quals és important i beneficiós moure peces cap a les columnes lliures:

1. Control del centre: El control del centre del tauler és fonamental en els escacs, ja que proporciona una posició centralitzada des de la qual es poden llançar atacs i maniobres en totes les direccions. Moure peces cap a les columnes lliures permet un major control del centre i facilita l'ocupació de caselles clau.
2. Desenvolupament de peces: Moure peces cap a les columnes lliures és una forma efectiva de desenvolupar-les i millorar la seva activitat. En moure una peça des de la seva posició inicial cap a una columna lliure, se li atorga major mobilitat i s'amplien les seves possibilitats d'influir en el joc.
3. Connexió de peces: En moure peces cap a les columnes lliures, es facilita la connexió entre elles. La connexió entre les peces és important per a establir una cooperació eficient i aprofitar el seu potencial combinat.
4. Atacs a peces enemigues: Les columnes lliures brinden l'oportunitat de realitzar atacs directes sobre les peces enemigues. En moure una peça cap a una columna lliure, es pot apuntar a les files i diagonals enemigues, la qual cosa pot conduir a amenaces tàctiques i oportunitats de captura.
5. Pressió sobre el rei enemic: Moure peces cap a les columnes lliures pot ajudar a crear una pressió addicional sobre el rei enemic. En ocupar columnes obertes prop del rei enemic, es generen amenaces potencials i s'augmenta el risc per a l'oponent.

En resum, les columnes lliures són importants en els escacs perquè permeten el control del centre, el desenvolupament de peces, la connexió entre elles, els atacs a peces enemigues i la pressió sobre el rei contrari. Moure peces cap a les columnes lliures és una estratègia comunament utilitzada per a aprofitar aquests avantatges i millorar la posició i el joc en general. És per això que és un factor clau en l'aprenentatge bàsic dels escacs i un factor a tenir en compte en l'explicació que donarà el nostre algorisme.

Per a generar la nova xarxa i com que al final totes les xarxes s'ajuntaran en un mateix programa, s'utilitzarà la mateixa codificació d'entrada que en la resta de xarxes, ja que ja ha quedat demostrat que és efectiva per a representar les posicions del taulell. Per a la sortida, només caldrà una neurona que representarà un valor binari (0 o 1) indicant si el moviment ha estat cap a una casella que es troba en una columna lliure o no. Tenint clara la codificació d'entrada i sortida, cal generar les dades per a poder entrenar i testejar la xarxa. Per a l'entrada, ja disposem de les posicions i de la funció de codificació de l'entrada a partir dels FEN. Pel que fa a la sortida, es generarà per a cada parell de posicions un valor binari. Per fer això, es crea una funció en Python que (utilitzant de nou la llibreria Python-chess i els FEN de les posicions que ja tenim a la base de dades) serà capaç de generar la posició final i comprovar si columna on es troba la casella final té algun peó (només es tenen en compte peons, ja que són les úniques peces que encara volent no es poden moure fàcilment entre columnes, per tant, es considera que bloquegen el pas tant de peces aliades com enemigues).

Per últim, i com s'ha estat fent amb totes les xarxes, cal trobar l'arquitectura de xarxa idònia per a la nova sortida, ja que com ha canviat molt (d'unes 20 neurones en les anteriors a 1 neurona en la nova) no es pot afirmar que les arquitectures anteriors funcionin. Per a la nova xarxa es plantegen les arquitectures següents:

i. Xarxa 1
   - Capa d'entrada: 128 neurones
   - Capa sortida: 1 neurona

ii. Xarxa 2
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa sortida: 1 neurona

iii. Xarxa 3
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa oculta 2: 128 neurones
   - Capa sortida: 1 neurona

iv. Xarxa 4
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa oculta 2: 128 neurones
   - Capa oculta 3: 64 neurones
   - Capa oculta 4: 256 neurones
   - Capa sortida: 1 neurona

v. Xarxa 5
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 256 neurones
   - Capa oculta 2: 128 neurones
   - Capa oculta 3: 64 neurones
   - Capa oculta 4: 32 neurones
   - Capa sortida: 1 neurona

vi. Xarxa 6
   - Capa d'entrada: 128 neurones
   - Capa oculta 1: 128 neurones
   - Capa oculta 2: 256 neurones
   - Capa oculta 3: 64 neurones
   - Capa oculta 4: 32 neurones

- Capa oculta 3: 64 neurones
- Capa oculta 4: 16 neurones
- Capa sortida: 1 neurona

vii. Xarxa 7
- Capa d'entrada: 128 neurones
- Capa oculta 1: 64 neurones
- Capa sortida: 1 neurona

viii. Xarxa 8
- Capa d'entrada: 128 neurones
- Capa oculta 1: 256 neurones
- Capa oculta 2: 128 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 32 neurones
- Capa oculta 5: 16 neurones
- Capa sortida: 1 neurona

ix. Xarxa 9
- Capa d'entrada: 128 neurones
- Capa oculta 1: 256 neurones
- Capa oculta 2: 128 neurones
- Capa oculta 3: 64 neurones
- Capa oculta 4: 32 neurones
- Capa oculta 3: 16 neurones
- Capa oculta 4: 8 neurones
- Capa sortida: 1 neurona

Per les següents arquitectures presentades els resultats han estat els següents:

Arquitectura de xarxa	Resultat entrenament	Resultat test
Xarxa 5	82,48%	77,38%
Xarxa 4	82,44%	77,36%
Xarxa 9	82,46%	77,29%
Xarxa 7	82,38%	77,22%
Xarxa 3	82,44%	77,13%
Xarxa 2	82,36%	77,00%
Xarxa 6	82,40%	76,75%
Xarxa 8	82,46%	76,67%
Xarxa 1	81,55%	76,03%

Taula 5 Resultats òptims de les diferents arquitectures de la xarxa d'explicació del control

Com ràpid es pot concloure de la taula, l'arquitectura que millors resultats presenta, tant en entrenament com en test, és l'arquitectura 5, per tant, aquesta ha estat l'escollida per a la configuració de la xarxa de columna lliure.

Avaluant els resultats, es pot veure com s'ha obtingut un percentatge d'encert molt elevat, per tant, es pot afirmar que la xarxa funciona i és vàlida per a realitzar la funció per a la qual ha estat dissenyada.

6.2 Algorisme per a extreure informació de mobilitat del taulell

En contrapartida a les xarxes neuronals, existeix també el recurs clàssic de la creació d'un algorisme per a extreure informació. Com s'ha anat fent fins ara, aquest algorisme es dissenya per a generar les sortides de la xarxa de dades i que aquestes serveixen per a comparar amb la sortida de la xarxa i anar refinant els pesos. Això es així, perquè aquests algorismes tenen un temps elevat de càlcul per a treure tots els valors que es requereixen per a l'explicació, per això, es generen un cop i s'usen per a entrenar la xarxa que, un cop entrenada, serà capaç de reproduir aquests resultats de forma quasi-immediata i sense gaire error. Però, d'altra banda, si el que volem es algun aspecte de la partida en concret que és senzill de calcular amb un algorisme, no passar per tot el procés d'entrenar una xarxa per a generar aquest resultat. És per això, que alguns aspectes dels moviments també vindran donats per un algorisme i no generats per una xarxa.

Seguint aquest raonament, el següent aspecte a tenir en compte és la restricció dels moviments del rival i ampliació del teu nombre de moviments. En els escacs, és fonamental comprendre la importància de moure les teves peces de manera estratègica per a restringir el moviment de les peces rivals i augmentar la mobilitat de les teves pròpies peces. Algunes de les raons clau de per què aquesta estratègia és important són:

1. Control de l'espai: En moure les teves peces de manera efectiva, pots controlar l'espai en el tauler. En ocupar caselles clau i limitar el moviment de les peces enemigues, pots restringir les opcions del teu oponent i reduir la seva llibertat de moviment. Això pot ajudar-te a establir una posició més sòlida i facilitar les teves pròpies maniobres.
2. Amenaces i tàctiques: En restringir les opcions de moviment de les peces rivals, pots crear amenaces i oportunitats tàctiques. En col·locar les teves peces en posicions que ataquin o amenacin les peces enemigues, obligues el teu oponent a respondre i a prendre decisions defensives. Això pot resultar en la captura de peces enemigues, guany de material o la creació de febleses en la posició de l'oponent.
3. Bloqueig i obstrucció: En moure les teves peces de manera estratègica, pots bloquejar o obstruir el moviment de les peces enemigues. Això pot ser especialment útil en situacions en les quals el teu oponent té una peça molt activa o amenaçadora. En bloquejar el seu camí i limitar el seu abast, pots neutralitzar la seva efectivitat i minimitzar la seva influència en el joc.
4. Coordinació de peces: En moure les teves peces per a restringir el moviment enemic, també pots augmentar la coordinació entre les teves pròpies peces. En establir una xarxa de suport i cooperació entre les teves peces, pots maximitzar el seu potencial combinat i crear amenaces més poderoses. La mobilitat augmentada de les teves pròpies peces també et permet respondre de manera més flexible als moviments de l'oponent i adaptar-te a l'evolució de la partida.
5. Seguretat del rei: Moure les teves peces per a restringir el moviment de les peces rivals també pot contribuir a la seguretat del teu rei. En limitar l'espai d'acció de l'oponent, redueixes les possibles amenaces directes contra el teu rei i crees una barrera defensiva més sòlida.

En general, moure les teves peces de manera que restringeixin el moviment de les peces enemigues i augmentin la mobilitat de les teves pròpies peces és una estratègia essencial en els escacs. Aquesta tàctica et permet establir un control efectiu del tauler, crear amenaces, coordinar les teves peces de manera més eficient i garantir la seguretat del teu rei.

Amb el propòsit de comprovar l'efectivitat d'un moviment respecte a la restricció de mobilitat que aporta, es crea un algorisme que, donades les dues posicions abans i després de fer el moviment, simula la posició amb la llibreria Python-chess i amb la funció *legal_moves()*, és capaç d'obtenir el nombre de moviments per a cada color en la posició. Un cop obtinguts els nombres de moviments, la funció calcula la diferència entre els moviments possibles d'un jugador abans i després del moviment i retorna el nombre de moviments que ha guanyat (valors positius) o ha perdut (valor negatiu) cada jugador, en el cas de ser 0, significa que el jugador té el mateix nombre de jugades disponibles que abans de fer el moviment. Amb aquest nou valor calculat i sabent qui ha fet el moviment es podran donar moltes més explicacions respecte del perquè de la valoració d'un moviment.

7. Construcció de l'algorisme final

En aquest apartat, es construeix l'algorisme final que serà el resultat de la combinació del tot el mostrat anteriorment i crearà la sortida de text que permetrà donar una idea al jugador de com és una posició. Aquest algorisme tindrà el següent funcionament:

1. S'utilitzarà la xarxa d'explicació de moviments per a generar la informació del moviment i les primeres dades del taulell.
2. Es comprovarà que la predicció és correcta.
3. S'utilitzarà la xarxa de columnes buides per a generar més informació.
4. Amb l'algorisme de restricció de moviments s'obtindrà la informació restant del taulell.
5. Amb tots els atributs donats es generarà un text únic per a la posició.

Generació de la valoració de la posició i comprovació de la predicció

En aquest primer pas, s'inicialitza la xarxa amb el fitxer h5 generat anteriorment, i es prepara per a retornar els valors de sortida de la xarxa. El codi d'aquesta funció, es pot veure a continuació:

```
def MLPexplanations(self, fen1, fen2):

      inputs = self.convertTuple((fen1,fen2))

      inputs = inputs.astype('float32') / 127

      model = load_model('./ModelFiles/model.h5')

      predictions = model.predict(inputs, verbose=0)

      return predictions[0]
```

Com es pot veure, la funció té d'entrada els valors dels dos FEN de les posicions, transforma aquests valors als seus valors numèrics i se'ls dona el format adequat i es carrega el model. Finalment, s'utilitza aquest model per a fer la predicció de com seran les característiques del moviment i es retornen.

Un cop s'ha predit el valor i abans de ser utilitzat per generar text, es comprovarà si és correcte (entenent per correcte que no s'hagi donat dues valoracions al moviment o s'hagi dit que una peça està a més d'una posició en el taulell), en cas afirmatiu, es continuarà amb el normal funcionament de l'algorisme, en cas contrari, a partir dels dos FEN, es generarà la sortida amb ús d'algorismes (els mateixos que han servit per a generar els valors d'entrenament de la xarxa), a causa del gran percentatge d'encert d'aquesta xarxa s'espera que la generació dels valors de sortida sigui quasi immediat i no s'hagi de fer gaire ús de l'altre càlcul, ja que és més costos i ineficient.

Determinació de columnes buides

El següent pas de l'algorisme serà utilitzar la segona xarxa neuronal per a saber si el moviment ha estat cap a una casella que es troba a una columna lliure. Per a fer això el sistema és igual que en el codi anterior, amb la variació que els pesos carregats hauran de ser els de la segona xarxa neuronal, a més, en aquest cas es retornarà un únic valor binari, no una taula. A diferència de la xarxa anterior, amb aquesta no es podrà comprovar si hi ha hagut un error, ja que només tenim un valor de sortida.

Obtenció de valor de restricció de moviments

L'últim pas en l'obtenció de les dades del taulell és utilitzar l'últim algorisme creat per a entendre qui i com control el tauler.

```
def restrictingMoves(self, fen1, fen2):

      board1 = chess.Board(fen1)

      board2 = chess.Board(fen2)

      board1.turn = board2.turn = chess.WHITE

      difW = len(list(board2.legal_moves))-
             len(list(board1.legal_moves))

      board1.turn = board2.turn = chess.BLACK

      difB = len(list(board2.legal_moves))-
             len(list(board1.legal_moves))

      return difW,difB
```

Com es pot veure, aquest algorisme utilitza el FEN de les posicions per a generar el taulell i calcular el nombre de moviments legals per a cada posició i cada jugador, posteriorment calcula si aquest nombre de moviments augmenta, disminueix o es queda igual i ho retorna com a resultat.

Aquest és l'últim pas en l'obtenció de les dades necessàries per a explicar la posició, per tant, un cop les tenim totes i s'ha comprovat que són correcte, el següent pas serà formar text a partir d'elles.

Creació del text amb les conclusions extretes

Finalment, l'última part d'aquest algorisme s'encarrega d'extreure els resultats i construir una cadena de text per a representar tota la informació mencionada. La forma de crear aquest text ha estat encadenant condicionals, es podria veure com un arbre binari on la primera decisió és si ha estat un moviment bo o dolent per al jugador que té el torn, a continuació, es van mirant altres aspectes, amb el context que va agafant la frase (negatiu o positiu) es van adaptant els connectors per a formar un text coherent, instructiu i que no només és explicatiu sinó que s'humanitza el caràcter de la frase. A continuació, es pot veure un exemple de codi:

```
if (fen1.split()[1] == "w" and dicEvals[ind][1] == 0) or
(fen1.split()[1] == "b" and dicEvals[ind][1] == 1):

      frase_expl = "El moviment realitzat és favorable per tu."

      if escac:

            frase_expl = frase_expl + "Has aturat l'escac"

                  if captura:

                        frase_expl = frase_expl + " i capturat
                        una peça"
```

```
                    if mate: frase_expl = frase_expl + ",
                    però segueixes en mat."

                    else: frase_expl = frase_expl + "."

          else:

                    if mate: frase_expl = frase_expl + ",
                    però segueixes en mat."

                    else: frase_expl = frase_expl + "i
                    millores la posició en general."
```

En aquest tros de codi es pot veure com es comença a construir la frase. Primer es comprova qui està movent i si l'avaluació del moviment l'afavoreix, si es així, la frase començarà amb el text: "El moviment realitzat és favorable per tu. ". El següent punt a mirar és si ens trobem en escac, ja que com s'ha explicat, això implicaria que ens hem vist obligats en certa manera a fer aquest moviment, en cas d'estar en escac, uniríem el fragment anterior amb el text "Has aturat l'escac", en cas contrari, no afegiríem cap text a la frase. A continuació, mirem si aquest moviment captura una peça, en cas positiu, afegiríem el text "i capturat una peça", en cas contrari, no afegiríem res. Per últim, en aquest tros de codi, també es mira si el jugador es troba en mat. En cas positiu, tot i que ha fet un moviment bo, l'informaríem del seu estat afegint: ", però segueixes en mat.", en cas contrari, no caldria afegir res més. Aleshores, en una posició o fem un moviment favorable, aturem escac amb una captura, però estem en mate, l'explicació que se'ns donaria del moviment és la següent:

"El moviment realitzat és favorable per tu. Has aturat l'escac i capturat una peça, però segueixes en mat."

Aquest seria un exemple en una posició sense tenir en compte tots els paràmetres. Si tenim en compte tots l'aspecte que s'han treballat per a un moviment en concret, tenim una combinació de:

- 2 frases segons el moviment és bo o dolent.
- 2 per saber si estàs en escac.
- 2 frases per les posicions amb mat.
- 2 frases per les posicions amb captura.
- 3 frases per a la posició al taulell.
- 10 opcions diferents pel nombre de moviments.
- 2 pel control del taulell
- 2 per la columna lliure.

En conclusió, amb els atributs tractants, es poden formar 1920 combinacions d'explicacions diferents per als moviments d'una partida.

8. Creació de la GUI

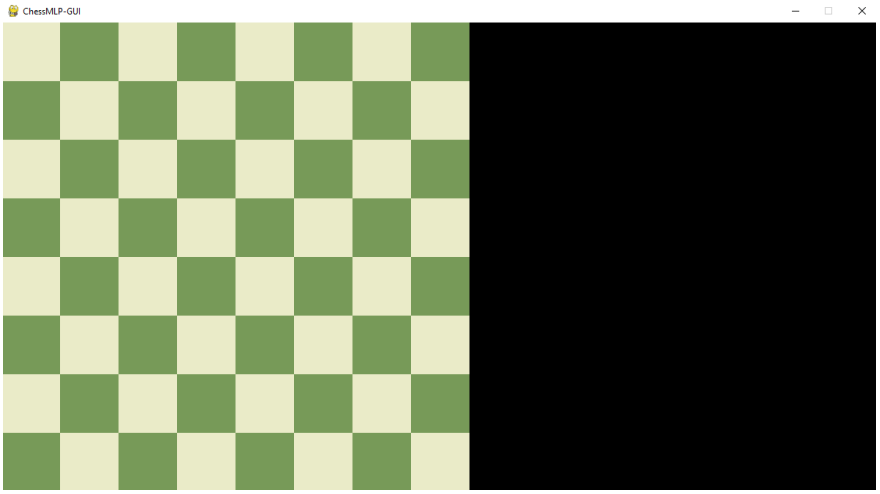
Per últim, per a poder utilitzar de forma senzilla l'algorisme d'explicació de partides, es crea una interfície gràfica d'usuari (GUI en anglès) per a poder jugar escacs i analitzar en qualsevol moment el moviment realitzat amb l'algorisme creat. Per a aquesta tasca s'ha utilitzat Python amb la llibreria pygame, aquesta llibreria s'utilitza per a crear videojocs i aplicacions multimèdia interactives. Proporciona eines i funcions per al desenvolupament de gràfics, so, animació, detecció de col·lisions i maneig d'esdeveniments, entre altres capacitats útils per a la creació de jocs. El desenvolupament d'aquesta GUI es divideix en les parts següents:

8.1 Implementació del taulell

En la implementació d'un motor per a poder jugar, la primera part es poder visualitzar el taulell amb les peces per a poder començar a tenir una base. En aquest cas, el primer que es crea és la finestra, aquesta constarà de dues parts:

- El taulell amb les 64 caselles corresponents.
- Espai addicional per a poder implementar funcions extres o afegir botons.

La finestra inicial es veu de la manera següent:

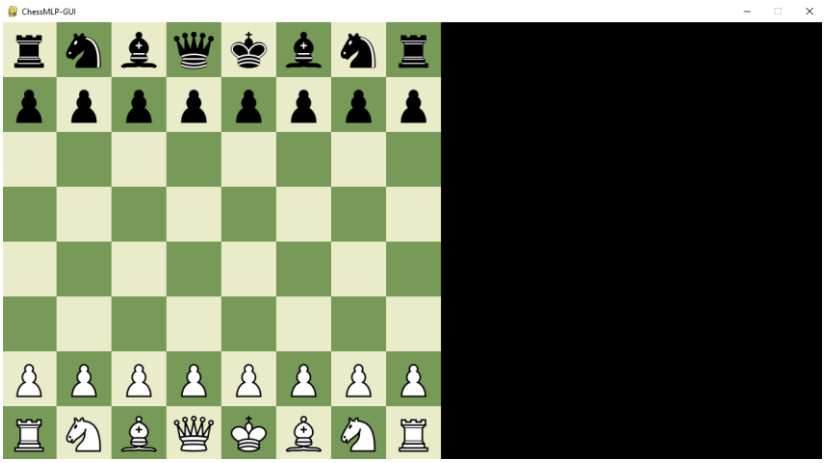


Il·lustració 8. Vista base de la GUI

El següent que cal fer és afegir totes les peces, per a poder tenir tota la part estètica muntada i començar a treballar amb la lògica de joc.

Per a afegir les peces, es recorre cada casella del taulell i es comprova si aquesta posició en el FEN conté una peça, si és així, s'afegeix a la casella.

Aquest pas aconsegueix mostrar el taulell de la següent manera:



Il·lustració 9. Vista de la GUI amb les peces incloses

Amb l'addició de les peces, es completa la creació de la part principal estètica del taulell, el següent serà afegir la lògica de joc.

8.2 Implementació de la lògica de joc

La implementació de la lògica de joc ha de tenir en compte molts factors, alguns d'ells poden ser:

- Representació del tauler: El motor d'escacs ha de tenir una representació interna del tauler d'escacs, que inclogui informació sobre la posició de les peces, l'estat del joc i altres regles específiques dels escacs. Aquesta representació ha de ser eficient i permetre un accés ràpid a la informació necessària per a la presa de decisions.

- Generació de moviments legals: El motor ha de ser capaç de generar tots els moviments legals possibles per a una posició donada. Això implica tenir en compte tant les regles, com els moviments de les diferents peces, les captures, l'enroc, el moviment del peó al pas i la promoció del peó. Generar i validar els moviments legals pot ser complicat a causa de les múltiples regles i restriccions en el joc.

- Optimització i eficiència: Per a fer un motor ràpid i que sigui servible cal optimitzar diferents aspectes, com la representació del tauler, els algorismes de cerca i la gestió de la memòria. L'optimització és essencial perquè el motor pugui prendre decisions en un temps raonable.

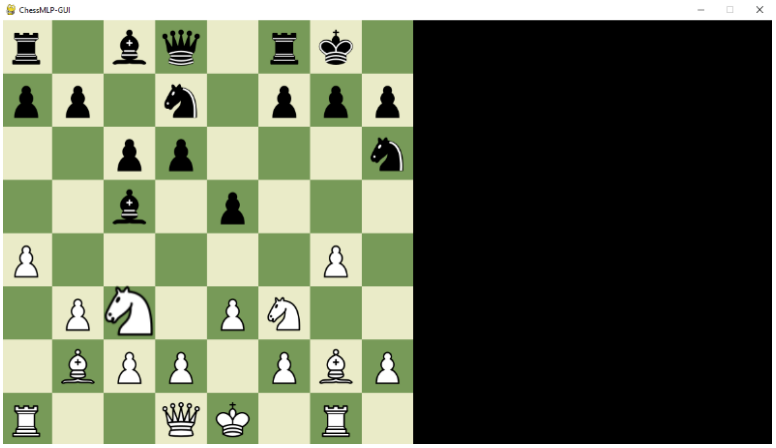
A causa de la complexitat de tots aquests components, i tenint en compte que no es troben en l'objectiu del treball, sinó que són un medi per a mostrar els resultats, s'ha decidit fer servir una llibreria ja creada i optimitzada per a portar tota la lògica de joc. Perquè ja s'ha fet servir en algun altre punt del treball, s'ha optat per a fer servir la llibreria Python-chess. Aquesta llibreria permet inicialitzar una partida des de la posició d'inici, anar seguint els moviments, validar la seva correctesa i detectar quan la partida s'ha acabat.

Un cop triat el motor amb el qual es portarà la lògica, cal implementar la funció del moviment de peces. Aquesta implementació es fa de la següent manera:

1. El joc detecta que s'ha clicat a sobre del taulell i detecta sobre quina casella, si aquesta casella conté una peça, es passa al punt següent, si no, no es fa res.

2. En seleccionar la peça, s'augmenten les seves dimensions per a fer sensació d'estar agafada i se segueix el moviment d'aquesta pel taulell.
3. Un cop es deixa la peça, es detecta en quina casella s'ha fet i es comprova si el moviment es correcte, en cas afirmatiu, la peça es mourà a la nova posició, en cas negatiu, tornarà a la casella inicial.

Aquesta lògica se seguirà fins que es detectin alguna de les condicions de final de partida, si és així s'acabarà el joc i es reiniciarà a la posició inicial. En la imatge següent es mostra la posició en meitat d'una partida i en meitat d'un moviment:



Il·lustració 10. Vista de la GUI amb la implementació dels moviments

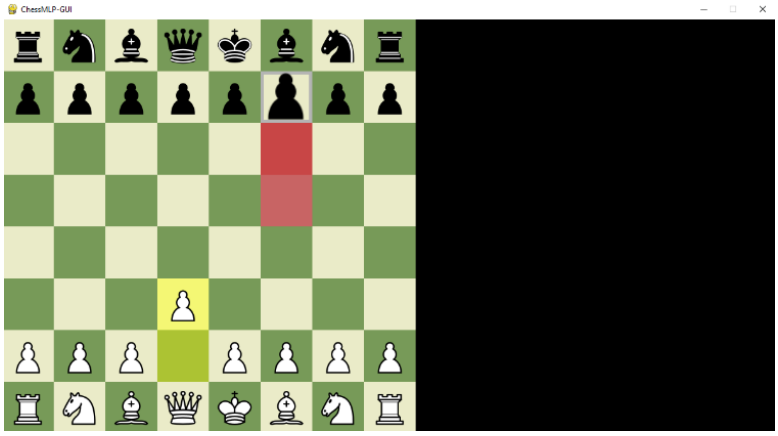
8.3 Creació de funcions extres

En aquest apartat, s'afegiran totes les funcions necessàries per a millora la qualitat del joc i afegir funcionalitats útils a l'hora de jugar i fer anàlisi de partides.

Millores visuals

- S'implementa, quan es va a moure, la visualització dels possibles moviments legals que té una peça, això permet a algú no experimentat amb el joc veure ràpidament on poden moure les seves peces i facilitar el càlcul de jugades.
- Es ressalta el moviment més recent del rival per a facilitar el joc, això permet veure fàcilment el canvi respecte a la posició anterior, i ser facilitar la visualització de les intencions del rival.
- Es ressalta la casella sobre la qual es troba el cursor per a facilitar la navegació i reconeixement d'aquest sobre el taulell, aquesta millora no facilita directament el joc, però dona una sensació molt més agradable al jugador i permet reconèixer fàcilment la posició del ratolí.

Un cop afegides les tres funcionalitats noves, una posició qualsevol en el taulell es veuria de la següent manera:



Il·lustració 11. Vista de la GUI amb implementacions visuals

Avaluació de la posició

Com en la majoria de motors actuals d'escacs, s'afegeix una avaluació de la posició per a poder ajudar a entendre millora qui té l'avantatge. Per a fer-ho, s'utilitza stockfish. Per a cada nou moviment, es crearà el FEN, serà avaluat i mostrat per pantalla.

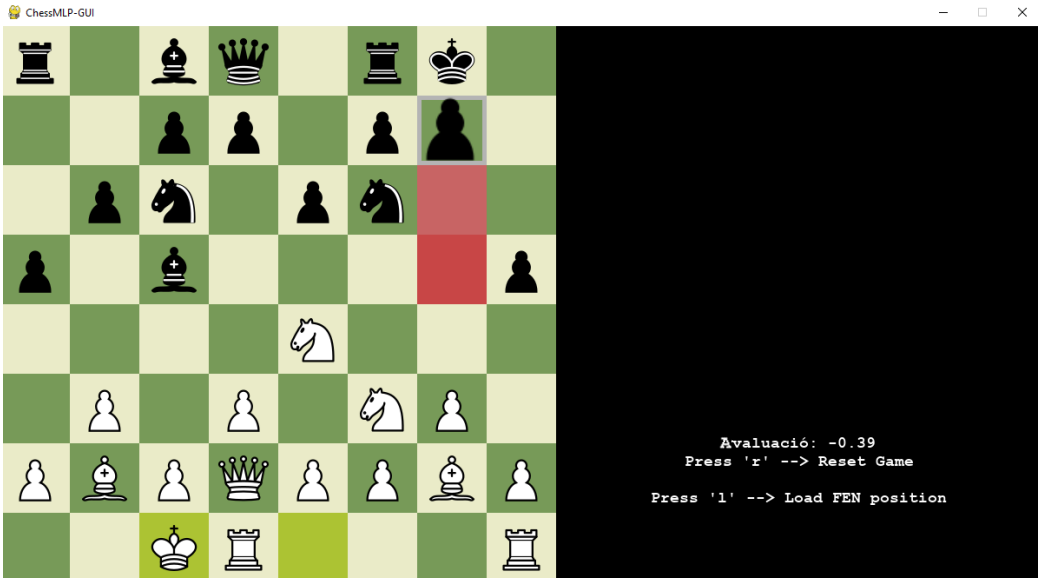
Reinici de la partida

S'afegeix la possibilitat de poder reiniciar la partida en qualsevol moment, això permet tornar a començar des de la posició inicial en cas d'estar analitzant una partida o reiniciar de zero en cas d'estar jugant contra algú i que aquest es rendeixi.

Càrrega de posicions

S'implementa la funcionalitat de carregar una posició a partir del seu FEN, el programa és capaç de llegir aquesta posició, representar-la en el tauler i jugar des d'aquest punt. Aquesta funció facilita molt l'anàlisi de certes posicions, ja que ajuda a arribar ràpidament al moment de la partida que es vol treballar, sense haver de jugar tots els moviments previs.

La implementació d'aquestes tres funcions, fan que la nova vista del programa sigui de la forma següent:



Il·lustració 12. Vista de la GUI amb funcions implementades. Groc: darrer moviment. Vermell: destins possibles

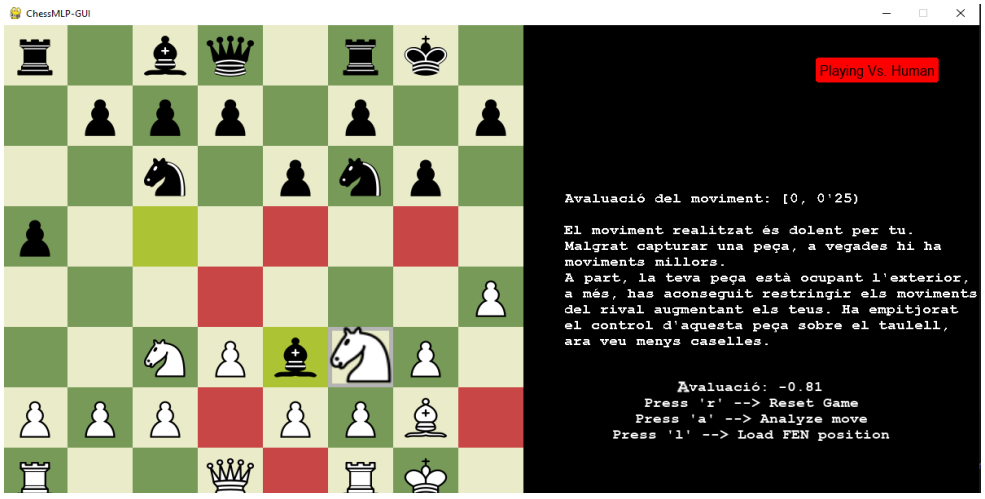
Possibilitat de jugar contra l'ordinador

La següent funcionalitat serà la possibilitat de triar entre de jugar contra l'ordinador o contra un altre jugador. Quan es tingui el mode vs. ordinador, s'utilitzarà l'algorisme creat per a generar els moviments i triar el millor possible. Aquest mode pot servir com a entrenament per a una persona que no tingui la possibilitat de jugar contra algú físicament.

8.4 Implementació de l'explicació de moviments

Per últim, s'afegeix l'algorisme explicat en el punt anterior que permet donar una valoració a un moviment i explicar el perquè d'aquesta. Per a fer això, igual que les altres funcionalitats, s'haurà de prémer la tecla indicada, el programa generarà el FEN de la posició abans i després del moviment i cridarà a la funció de generació d'expilació, aquesta, amb una velocitat quasi immediata, mostrarà per pantalla la valoració i el perquè d'aquesta.

Finalment, la vista del programa final amb totes les funcionalitats incloses és la següent:



Il·lustració 13. Vista de la GUI acabada

9. Conclusions

En aquest Treball de Fi de Grau s'han aconseguit satisfer tots els objectius proposats, la qual cosa representa un pas en la direcció correcta en la comprensió i aplicació de les xarxes neuronals en l'anàlisi de moviments d'escacs. També, he adquirit un sòlid coneixement dels fonaments teòrics de les xarxes neuronals artificials i la seva implementació en Python. Això m'ha permès comprendre i aplicar de manera efectiva les biblioteques com TensorFlow o Keras per a desenvolupar la xarxa neuronal.

Mitjançant una exhaustiva recerca i selecció de tècniques i algorismes, he aconseguit implementar una xarxa neuronal que analitza i classifica els moviments d'escacs com a bons o dolents. La xarxa neuronal ha demostrat un bon rendiment i capacitat per a explicar les raons darrere de les avaluacions. El conjunt de dades recopilat i preparat per a l'entrenament i validació de la xarxa neuronal ha estat adequat i representatiu dels moviments d'escacs. Això va permetre obtenir resultats significatius i de confiança durant les proves i avaluacions. El rendiment de la xarxa neuronal ha estat avaluat utilitzant mètriques apropiades, i s'han comparat els resultats amb les anàlisis realitzades per experts en escacs. Els resultats obtinguts demostren que la xarxa neuronal és capaç de realitzar avaluacions consistents i precises.

La interfície gràfica d'usuari (GUI) desenvolupada és intuïtiva i atractiva, la qual cosa facilita la interacció dels usuaris amb la xarxa neuronal. Els usuaris poden ingressar moviments d'escacs i obtenir explicacions clares i comprensibles sobre per què es consideren bons o dolents.

S'ha documentat adequadament tot el procés de desenvolupament, des del disseny de la xarxa neuronal fins a la metodologia d'entrenament, els resultats obtinguts i les conclusions aconseguides. Això garanteix la reproductibilitat i comprensió del treball realitzat, i proporciona una base sòlida per a futures recerques en el mateix camp.

En resum, aquest TFG ha demostrat amb èxit la viabilitat i eficàcia d'utilitzar una xarxa neuronal per a explicar la qualitat dels moviments d'escacs. Els objectius formatius s'han aconseguit íntegrament, la qual cosa ha permès adquirir habilitats i coneixements valuosos en l'àmbit de la intel·ligència artificial. Aquest projecte constitueix una contribució significativa al camp i estableix les bases per a futures recerques i desenvolupaments en aquesta àrea.

10. Bibliografia web

"Llibreria Python-chess" https://python-chess.readthedocs.io/en/latest/ (accés 25 d'abril 2023)

"Notació PGN" https://en.wikipedia.org/wiki/Portable_Game_Notation (accés 29 d'abril 2023)

"Stockfish a Python" https://pypi.org/project/stockfish/ (accés 29 d'abril 2023)

"Informació sobre Stockfish" https://en.wikipedia.org/wiki/Stockfish_(chess) (accés 30 d'abril 2023)

"Descarregar Stockfish" https://stockfishchess.org/ (accés 1 de maig 2023)

"Pygame a Python" https://pypi.org/project/pygame/ (accés 2 de maig 2023)

"Llibreria Pygame" https://www.pygame.org/news (accés 2 de maig 2023)

"Tipus de xarxes neuronals" https://www.mygreatlearning.com/blog/types-of-neural-networks/ (accés 2 de maig 2023)

"Entrenament de xarxes neuronals"
https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network (accés 3 de maig 2023)

"Què és un MLP?" https://subscription.packtpub.com/book/programming/9781838821654/1/ch01lvl1sec04/3-multilayer perceptron-mlp (accés 3 de maig 2023)

"Xarxes neuronals amb Keras" https://medium.com/data-science-365/acquire-understand-and-prepare-the-mnist-dataset-3d71a84e07e7 (accés 4 de maig 2023)

"MLP amb Keras i Tensorflow" https://towardsdatascience.com/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10 (accés 4 de maig 2023)

"Implementar perceptrons multicapa" https://medium.com/@artjovianprojects/deep-learning-project-multilayer-perceptron-e34017941918 (accés 4 de maig 2023)

"Bases de dades de PGN" https://www.pgnmentor.com/files.html#players (accés 6 de maig 2023)

"PGN a Python" https://python-chess.readthedocs.io/en/latest/pgn.html (accés 7 de maig 2023)

Annex A: Resultats de les proves amb les arquitectures de xarxa

Taules amb els percentatges d'encert de les codificacions d'entrada i les diferents combinacions d'arquitectura i de sortida.

- Taula completa amb les 49 combinacions de xarxes i sortides per a la codificació amb tota la informació a les caselles:

Arquitectura de xarxa	Funció d'avaluació	Resultat entrenament	Resultat test
Xarxa 1	avaluació 1	91.15%	89.53%
Xarxa 2	avaluació 1	91.15%	89.53%
Xarxa 3	avaluació 1	91.15%	89.53%
Xarxa 4	avaluació 1	91.15%	89.53%
Xarxa 5	avaluació 1	91.15%	89.53%
Xarxa 6	avaluació 1	91.15%	89.53%
Xarxa 7	avaluació 1	91.15%	89.53%
Xarxa 1	avaluació 2	68.72%	61.65%
Xarxa 2	avaluació 2	68.71%	61.59%
Xarxa 3	avaluació 2	68.71%	61.59%
Xarxa 4	avaluació 2	68.71%	61.59%
Xarxa 5	avaluació 2	68.71%	61.59%
Xarxa 6	avaluació 2	68.71%	61.59%
Xarxa 7	avaluació 2	68.71%	61.59%
Xarxa 5	avaluació 3	57.90%	48.12%
Xarxa 3	avaluació 3	57.79%	48.02%
Xarxa 6	avaluació 3	57.78%	47.67%
Xarxa 4	avaluació 3	57.72%	47.65%
Xarxa 2	avaluació 3	57.52%	47.44%
Xarxa 7	avaluació 3	57.57%	47.38%
Xarxa 1	avaluació 3	57.00%	46.80%
Xarxa 1	avaluació 4	59.82%	50.72%
Xarxa 2	avaluació 4	56.68%	46.25%
Xarxa 3	avaluació 4	56.68%	46.25%
Xarxa 4	avaluació 4	56.68%	46.25%
Xarxa 5	avaluació 4	56.68%	46.25%
Xarxa 6	avaluació 4	56.68%	46.25%
Xarxa 7	avaluació 4	56.68%	46.25%
Xarxa 1	avaluació 5	85.50%	81.48%
Xarxa 2	avaluació 5	85.50%	81.48%
Xarxa 3	avaluació 5	85.50%	81.48%
Xarxa 4	avaluació 5	85.50%	81.48%
Xarxa 5	avaluació 5	85.50%	81.48%
Xarxa 6	avaluació 5	85.50%	81.48%
Xarxa 7	avaluació 5	85.50%	81.48%
Xarxa 1	avaluació 6	91.15%	89.53%
Xarxa 2	avaluació 6	91.15%	89.53%
Xarxa 3	avaluació 6	91.15%	89.53%

Xarxa 4	avaluació 6	91.15%	89.53%
Xarxa 5	avaluació 6	91.15%	89.53%
Xarxa 6	avaluació 6	91.15%	89.53%
Xarxa 7	avaluació 6	91.15%	89.53%
Xarxa 1	avaluació 7	85.50%	81.48%
Xarxa 2	avaluació 7	85.50%	81.48%
Xarxa 3	avaluació 7	85.50%	81.48%
Xarxa 4	avaluació 7	85.50%	81.48%
Xarxa 5	avaluació 7	85.50%	81.48%
Xarxa 6	avaluació 7	85.50%	81.48%
Xarxa 7	avaluació 7	85.50%	81.48%

Taula 6 Tots els resultats de les combinacions d'arquitectures i avaluacions per a la codificació d'entrada amb tota la informació a les caselles

- Taula completa amb les 49 combinacions de xarxes i sortides per a la codificació amb les dades de les peces i la informació de la partida separades:

Arquitectura de xarxa	Funció d'avaluació	Resultat entrenament	Resultat test
Xarxa 1	avaluació 1	91.08%	89.34%
Xarxa 2	avaluació 1	91.15%	89.53%
Xarxa 3	avaluació 1	91.15%	89.53%
Xarxa 4	avaluació 1	91.15%	89.53%
Xarxa 5	avaluació 1	91.15%	89.53%
Xarxa 6	avaluació 1	91.15%	89.53%
Xarxa 7	avaluació 1	91.15%	89.53%
Xarxa 1	avaluació 2	68.78%	61.61%
Xarxa 2	avaluació 2	68.71%	61.59%
Xarxa 3	avaluació 2	68.71%	61.59%
Xarxa 4	avaluació 2	68.71%	61.59%
Xarxa 5	avaluació 2	68.71%	61.59%
Xarxa 6	avaluació 2	68.71%	61.59%
Xarxa 7	avaluació 2	68.71%	61.59%
Xarxa 1	avaluació 3	57.56%	47.55%
Xarxa 2	avaluació 3	57.71%	47.64%
Xarxa 3	avaluació 3	57.98%	48.02%
Xarxa 4	avaluació 3	57.87%	48.10%
Xarxa 5	avaluació 3	57.95%	48.17%
Xarxa 6	avaluació 3	56.68%	46.25%
Xarxa 7	avaluació 3	57.71%	47.93%
Xarxa 1	avaluació 4	60.75%	52.18%
Xarxa 2	avaluació 4	56.68%	46.25%
Xarxa 3	avaluació 4	56.68%	46.25%
Xarxa 4	avaluació 4	56.68%	46.25%
Xarxa 5	avaluació 4	56.68%	46.25%
Xarxa 6	avaluació 4	56.68%	46.25%
Xarxa 7	avaluació 4	56.68%	46.25%
Xarxa 1	avaluació 5	85.46%	81.37%
Xarxa 2	avaluació 5	85.50%	81.48%

Xarxa 3	avaluació 5	85.50%	81.48%
Xarxa 4	avaluació 5	85.50%	81.48%
Xarxa 5	avaluació 5	85.50%	81.48%
Xarxa 6	avaluació 5	85.50%	81.48%
Xarxa 7	avaluació 5	85.50%	81.48%
Xarxa 1	avaluació 6	91.08%	89.33%
Xarxa 2	avaluació 6	91.15%	89.53%
Xarxa 3	avaluació 6	91.15%	89.53%
Xarxa 4	avaluació 6	91.15%	89.53%
Xarxa 5	avaluació 6	91.15%	89.53%
Xarxa 6	avaluació 6	91.15%	89.53%
Xarxa 7	avaluació 6	91.15%	89.53%
Xarxa 1	avaluació 7	85.46%	81.37%
Xarxa 2	avaluació 7	85.50%	81.48%
Xarxa 3	avaluació 7	85.50%	81.48%
Xarxa 4	avaluació 7	85.50%	81.48%
Xarxa 5	avaluació 7	85.50%	81.48%
Xarxa 6	avaluació 7	85.50%	81.48%
Xarxa 7	avaluació 7	85.50%	81.48%

Taula 7 Tots els resultats de les combinacions d'arquitectures i avaluacions per a la codificació d'entrada amb la informació del taulell en camps diferents

Annex B: Accessibilitat al codi

Tot la informació i codi sobre l'entrenament i creació de les xarxes es pot trobar al repositori de GitHub següent:

https://github.com/sergial273/TFG-Perceptro

El codi i les instruccions per a l'execució de la interfície gràfica es poden trobar en el repositori següent:

https://github.com/sergial273/TFG-MLPexplicacions-GUI