Quais os fatores que determinam, aparti da analise, a diferença entre aprendizado de maquina e da inteligencia artificial ?

Orientador: Mauricio Neves Asenjo

Carolaine de Jesus Souza

Carolina de Jesus Souza

Renato Messias

Micael Robério

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

Campus Cubatão

Sumário:

XADREZ

O xadrez teve um papel importante na popularização da inteligência artificial, existem 2 eventos envolvendo xadrez que foram bastante relevantes para o desenvolvimento da IA como ciência.

O primeiro sendo a primeira vez que uma inteligência artificial foi capaz de vencer um campeão mundial, em 1996, o campeão mundial de xadrez Garry Kasparov aceitou participar de um match de 6 partidas contra o supercomputador da IBM, o Deep Blue, nesse primeiro confronto o campeão mundial conseguiu vencer por 4 a 2(3 vitórias de Kasparov, 2 empates e uma vitória do Deep Blue), em 1997, outro confronto foi realizado, e dessa vez a inteligência artificial se saiu vitoriosa, ganhando por 3,5 a 2,5(2 vitórias, 3 empates e 1 derrota), depois desse confronto, as ações da IBM dispararam e as IA viraram fundamentais na preparação de qualquer jogador profissional.

O segundo grande evento foi quando a primeira IA a aprender a jogar sozinha, o AlphaZero, derrotou com grande facilidade o Stockfish, a principal IA da época, em 100 partidas, o AlphaZero venceu 28, e empatou 72, esse evento foi muito importante para o desenvolvimento do Machine Learning(aprendizado de máquina), porque mostrou que além de uma Inteligência Artifical poder superar um humano, ela pode ter melhores resultados aprendendo sozinha.

O xadrez vem tendo um papel fundamental nas relações humanas, despertando raciocínio, sabendo as regras, podendo aplicar técnicas na vida da pessoa, concentração, nas diferenças de seus desenvolvimentos (ele pode ser desenvolvido de diversas maneiras, mas as três formas principais são: xadrez em php, inteligência artificial de xadrez e até mesmo desenvolvê-lo no Excel e jogá-lo pelo mesmo ou elaborar o jogo de xadrez com o auxílio do google jamboard.

Em sua totalidade, o xadrez busca unir os jogadores, pois se tem uma interação entre os dois. Com isso, os jogadores fazem o uso de diversas estratégias, visando ganhar a partida de forma justa e rápida. Levando em consideração que ele pode ser jogado em diversos meios, como no tabuleiro, pelo computador ou celular.

Regras :

OS movimentos da Peças de cada peça no xadrez demonstram uma singularidade, são eles:

-Peão (Pawn): Movimentos apenas para frente e somente uma casa por jogada, uma exceção é que apenas no primeiro movimento de cada peão, tem a oportunidade de se mover até duas casas a frente. Só podem eliminar peças que estão uma casa a frente à esquerda ou à direita.

-Torres (Tower): Podem se mover livremente para frente, para trás e ambos os lados, podendo se mover quantas casas vazias pude, podendo eliminar uma peça inimiga que esteja no caminho.

-Cavalos (Knight): Só podem se mover em “L”, três casas para qualquer lado e então uma casa a esquerda ou a direita. Só podem eliminar as peças inimigas que estiverem na última casa percorrida.

-Bispos (Bishop): Podem se mover livremente nas diagonais, quantas casa o jogador desejar, porém cada Bispo fica em uma cor diferente do tabuleiro, podendo eliminar uma peça inimiga que esteja no caminho.

-Rainha (Queen): Pode se mover livremente em todas as direções, quantas o jogador desejar, podendo eliminar uma peça inimiga que esteja no caminho.

-Rei (King): Pode se mover em todas as direções como a rainha, porém só pode mover uma casa, podendo eliminar uma peça inimiga próxima. Um rei não pode eliminar.

-Roque (Rook): Um movimento especial que só pode ser feito uma vez durante o jogo e movimenta duas peças de uma vez, onde o Rei e a Torre mais próxima. O movimento consiste no deslocamento lateral do rei na

primeira fileira em duas casas na direção na torre com a qual desejar "rocar", e a torre passa através do rei permanecendo na primeira casa após o "salto". A distância entre o rei e a torre antes do movimento determina se a jogada é classificada como roque grande ou roque pequeno.

São condições necessárias para a realização do movimento:

1. O rei e a torre devem estar na mesma fileira.
2. O rei e a torre não podem ter sido movidos previamente.
3. As casas entre o rei e a torre devem estar desocupadas.
4. O rei não pode ter estado em xeque anteriormente ao roque.

5. Nenhuma das casas onde o rei passará sofrem ameaças. De acordo com a FIDE, isto significa não haver ameaças para o rei na casa em que está, nas quais passará e na qual o rei ficará após o movimento.

6. O movimento do rei não pode resultar xeque.

Bruno Ribeiro, Fabiano Lucchese, Maycon Rocha e Vera Figueiredo; Inteligência Artificial em Jogos Digitais, apresentaram no artigo uma breve descrição dos principais mecanismos de inteligência artificial utilizados em jogos digitais. Em seguida, discutiram algumas das aplicações destes mecanismos em jogos, procuraram ainda apontar as possibilidades futuras de uso de inteligência artificial nesta área. Deram exemplos de como isso foi aplicado em um" jogo da velha", que é menos complexo que o xadrez, em sistemas em busca de arvores, os autores introduzem uma modelagem capaz de representar uma grande variedade de jogos em função de suas representações de estado, as regras pelas quais mudanças de estados podem ocorrer e o custo ou benefício que cada participante consegue depreender de cada estado.

INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial, que você vai ver por aí sendo citada apenas como IA (ou AI, de artificial intelligence), é um avanço tecnológico que permite que sistemas simulem uma inteligência similar à humana — indo além da programação de ordens específicas para tomar decisões de forma autônoma, baseadas em padrões de enormes bancos de dados. Vem continuar entendendo melhor o que é.

Algo tão complicado é também um campo de estudo acadêmico — que não começou ontem. Há algumas décadas, se estuda o que se chamou de “agentes inteligentes”, que percebem seu ambiente, entendem como podem operar e qual a melhor forma.
Credita-se ao professor John McCarthy o uso do termo pela primeira vez em 1956, em uma conferência de especialistas em Darmouth Colege, chamada “O Eros Eletrónico”, que definiu como “a ciência e a engenharia de produzir máquinas inteligentes”.
Assim, podemos definir inteligência artificial, no grosso modo, como a capacidade das máquinas de pensarem como seres humanos: aprender, perceber e decidir quais caminhos seguir, de forma racional, diante de determinadas situações.

Até então, os computadores precisavam de três grandes pilares para evoluir da computação simples para a atual, de inteligência artificial:

Bons modelos de dados para classificar, processar e analisar;

Acesso a grande quantidade de dados não processados;

Computação potente com custo acessível para processamento rápido e eficiente.

Com a evolução desses três segmentos, a inteligência artificial tornou-se finalmente possível com a fórmula: big data + computação em nuvem + bons modelos de dados.
Ou seja, a IA aprende como uma criança. Aos poucos, o sistema (a depender do objetivo para o qual ele foi criado) absorve, analisa e organiza os dados de forma a entender e identificar o que são objetos, pessoas, padrões e reações de todos os tipos.

Como assim, máquinas inteligentes?

Em sua essência, a Inteligência Artificial permite que os sistemas tomem decisões de forma independente, precisa e apoiada em dados digitais. O que, numa visão otimista, multiplica a capacidade racional do ser humano de resolver problemas práticos, simular situações, pensar em respostas ou, de forma mais ampla, potencializa a capacidade de ser inteligente.

Os economistas chamam isso de a quarta revolução industrial, marcada pela convergência de tecnologias digitais, físicas e biológicas — bagunçando as fronteiras das três áreas. E IA faz parte dessa próxima onda de inovação, trazendo grandes mudanças na maneira como pessoas e empresas se relacionam com tecnologia, compartilham dados e tomam decisões.

Máquinas reativas: são as formas mais antigas de inteligência artificial, que não possuem funcionalidade baseada em memória;

Memória limitada: conseguem aprender com base em dados históricos;

Teoria da mente: esse é o próximo nível de sistemas de IA que encontra-se em andamento;

“Autoconsciente”: a IA autoconsciente é uma formulação hipotética, que conseguirá compreender e evocar emoções, necessidades, crenças e, potencialmente, desejos próprios.

Agora, quando o assunto é a classificação técnica da inteligência artificial, devemos nos concentrar em três:

Inteligência artificial estreita (ANI): representa toda a IA existente, em que só pode realizar uma tarefa específica;

Inteligência geral artificial (AGI): se refere à capacidade da inteligência artificial geral aprender, perceber, compreender e funcionar completamente da mesma forma que um ser humano;

Superinteligência artificial (ASI): pode replicar a inteligência multifacetada dos seres humanos, possui uma memória maior, analisa dads rapidamente e possui capacidades de tomada de decisão.

REDE NAURAL

Os neurônios são as células que formam o cérebro. Elas são compostas basicamente por: (i) dendritos, que captam informações do ambiente ou de outras células, (ii) corpo celular ou Soma, responsável pelo processamento das informações, e (iii) axônio, para distribuir a informação processada para outros neurônios ou células do corpo. Entretanto, uma célula dificilmente trabalha sozinha. Quanto mais células trabalharem em conjunto, mais podem processar e mais eficaz torna-se o trabalho. Logo, para o melhor rendimento do sistema, são necessários muitos neurônios.

Dos Neurônios às Redes Neurais

Foi pensando em como os neurônios trabalham, que pesquisadores desenvolveram neurônios artificiais. As redes neurais artificiais consistem em um método para solucionar problemas de inteligência artificial, construindo um sistema que tenha circuitos que simulem o cérebro humano, inclusive seu comportamento, ou seja, aprendendo, errando e fazendo descobertas. São mais que isso, são técnicas computacionais que apresentam um modelo inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento a partir da experiência. Uma grande rede neural artificial pode ter centenas ou milhares de unidades de processamento, enquanto que o cérebro de um mamífero pode ter muitos bilhões de neurônios.

Uma Rede Neural Artificial (RNA) consiste em um grande número de elementos de processamento simples chamados neurônios, unidades, células ou nós. Cada neurônio está conectado a outros neurônios por meio de links direcionados, cada um com um peso associado. Os pesos representam informações que estão sendo usadas pela rede para resolver um problema. RNA podem ser aplicadas a uma ampla variedade de problemas, tais como armazenar e recuperar dados, classificar padrões, realizar mapeamentos em geral a partir de padrões de entrada, agrupando padrões similares, ou encontrar soluções para problemas de otimização.

Cada neurônio tem um estado interno, chamado de activity level, que é uma função das entradas que recebe. Tipicamente, um neurônio envia o activity level como um sinal para vários outros neurônios. É importante considerar, que um neurônio pode enviar apenas um sinal de cada vez, apesar de o sinal ser transmitido para vários outros neurônios.

Como um neurônio artificial é capaz de realizar um único processamento, cada entrada recebe somente um tipo de sinal ou informação, mas pode possuir várias entradas, e, então, perceber diferentes sinais. Assim, ligar vários neurônios similares em rede faz com que o sistema consiga processar mais informações e oferecer mais resultados.

De forma geral, a operação de um neurônio da rede se resume em:

Sinais são apresentados à entrada;

Cada sinal é multiplicado por um peso que indica sua influência na saída da unidade;

É feita a soma ponderada dos sinais que produz um nível de atividade;

Se esse nível excede um limite (threshold), a unidade produz uma saída.

Aplicações de Redes Neurais Artificiais

O estudo das redes neurais é um campo extremamente interdisciplinar, tanto no seu desenvolvimento quanto na sua aplicação. Uma breve listagem de algumas das áreas em que redes neurais estão sendo aplicadas mostra sua amplitude.

Processamento de Sinais

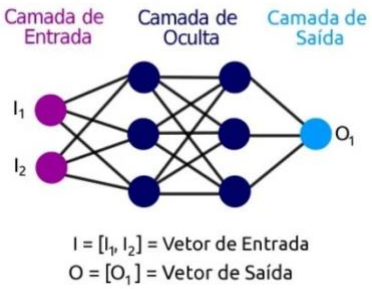
Existem muitas aplicações de redes neurais na área de processamento de sinais. Uma das primeiras aplicações comerciais foi (e ainda é) a supressão de ruído em linha telefônica.

Medicina

Um dos muitos exemplos de aplicação de redes neurais para a medicina, desenvolvido por Anderson et al. em meados da década de 80, é chamado de Instant Physician. A idéia por trás dessa aplicação é a formação de uma rede neural de memória associativa, para armazenar um grande número de registros médicos, os quais incluem informações sobre sintomas, diagnóstico e tratamento para um caso particular. Após o “treinamento”, a rede pode ser alimentada com uma entrada de um conjunto de sintomas, e então encontrará o melhor diagnóstico e tratamento.

Reconhecimento de Padrões

Muitos problemas interessantes estão na área de reconhecimento de padrões. Uma área em que muitos aplicativos de redes neurais têm sido desenvolvidos é o reconhecimento automático de caracteres manuscritos (dígitos ou letras). Além do reconhecimento de caracteres, as RNA podem ser aplicadas ao reconhecimento de padrões da voz. A grande variação nos timbres de voz e estilos de escrita, tornam esses problemas difíceis para as técnicas tradicionais, bons exemplos do tipo de processamento de informação que os seres humanos podem executar de forma relativamente fácil.



https://blogdozouza.wordpress.com/2019/12/09/como-codificar-uma-rede-neural-com-backpropagation-em-python-do-zero/

Aprendizagem de máquina

Machine learning também é um conceito com várias definições possíveis. Aqui vai uma que nos permite assimilar bem sua essência: aprendizado de máquina é um sistema que pode modificar seu comportamento autonomamente tendo como base a sua própria experiência — o treinamento que abordamos anteriormente. A interferência humana aqui é mínima.

A tal modificação comportamental consiste, basicamente, no estabelecimento de regras lógicas, vamos dizer assim, que visam melhorar o desempenho de uma tarefa ou, dependendo da aplicação, tomar a decisão mais apropriada para o contexto.

Para começar, é importante que sistemas do tipo façam análises com base em uma quantidade significativa de dados, coisa que os buscadores têm de sobra por conta dos milhões de acessos que recebem e que, consequentemente, servem de treinamento.

Outro aspecto ilustrado ali é a constante entrada de dados que favorece a identificação de novos padrões

Suponha que a palavra bravo passe a ser uma gíria associada a um movimento cultural. Com o machine learning, o mecanismo de pesquisa conseguirá identificar padrões que apontam para o novo significado do termo e, depois de algum tempo, estará apto a considerá-lo nos resultados das buscas.

Força bruta:

Algoritmos de força bruta são métodos diretos de resolver um problema que dependem de poder computacional puro e de tentar todas as possibilidades em vez de técnicas avançadas para melhorar a eficiência.

A força bruta é um algoritmo trivial mas de uso muito geral que consiste em enumerar todos os possíveis candidatos de uma solução e verificar se cada um satisfaz o problema.

Por exemplo, um algoritmo para encontrar os divisores de um número natural n é enumerar todos os inteiros de 1 a n, e verificar para cada um se ele dividido por n resulta em resto 0.

Esse algoritmo possui uma

implementação

muito simples, e sempre encontrará uma solução se ela existir. Entretanto, seu

custo computacional

é proporcional ao número de candidatos a solução, que, em problemas reais, tende a crescer exponencialmente. Portanto, a força bruta é tipicamente usada em problemas cujo tamanho é limitado, ou quando há uma

heurística

usada para reduzir o conjunto de candidatos para uma espaço aceitável. Também pode ser usado quando a simplicidade da implementação é mais importante que a velocidade de execução, como nos casos de aplicações críticas em que os erros de algoritmo possuem em sérias consequências.

Força bruta é uma tecnologia muito vista em ataques e invasões, o ataque de força bruta verifica cada possibilidade de senha até encontrar a correta.
Uma inteligencia artificial de xadrez de força bruta testa varias possibilidades de movimentos e avalia qual o melhor naquela posição, esse tipo de tecnologia usa o metodo minimax, que cria varias sequencias de jogadas, formando uma arvore de decisões para descobrir a jogada mais favoravel para a ia.
Machine Learning
Aprendizado de maquina é um metodo que ensina ao computador, atraves de uma base de dados, a como realizar uma tarefa.

Aprendizagem Por Reforço
A Aprendizagem Por Reforço é o treinamento de modelos de aprendizado de máquina para tomar uma sequência de decisões. O agente aprende a atingir uma meta em um ambiente incerto e potencialmente complexo. No aprendizado por reforço, o sistema de inteligência artificial enfrenta uma situação. O computador utiliza tentativa e erro para encontrar uma solução para o problema. Para que a máquina faça o que o programador deseja, a inteligência artificial recebe recompensas ou penalidades pelas ações que executa. Seu objetivo é maximizar a recompensa total.

Diferença entre os dois trabalhos
O machine learning é muito mais sofisticado, o número de possibilidades de jogadas no xadrez faz com que seja impossivel com que o computador avalie todas as posições, com o aprendizado de maquina o numero de

posições avaliadas diminui, porque atraves da base de dados que a Ia estudou ela tem noção de quais lances avaliar.

A linguagem de máquina não perde, ou ela ganha ou empata, ela aprende porque tem uma inteligência por trás dela , a de força bruta é um algoritmo que faz o que é programado para fazer. Ou seja, o algoritmos de linguagem de máquina é mais eficiente no caso do xadrez do que a força bruta(que não ganhou nenhuma partida) .De 30 partidas foram 7 empates e 23 vitórias da IA com Machine Learning.

Ambas as linguagens foram feitas em python, para fazer a de Machine Learning foi necessário usar tensorflow e algumas bibliotecas do python como pygame e mathlib, alem do google colab para fazer o treinamento, o tipo de rede neural foi a CNN(Convolutional neural network). Para fazer a outra inteligência artificial, foi necessário usar um método chamado "minimax"

Código Inteligência Artificial :

```
# -*- coding: utf-8 -*-
"""TensorflowAI.ipynb


Automatically generated by Colaboratory.


Original file is located at

https://colab.research.google.com/drive/1tCkp1Sv2LpxXZw0jEvczweeBswvvbTGa
"""
import chess
import chess.polyglot
import chess.svg
import chess.pgn
import chess.engine
import time
```

```
board = chess.Board()
board


pawntable = [
    0, 0, 0, 0, 0, 0, 0, 0,
    5, 10, 10, -20, -20, 10, 10, 5,
    5, -5, -10, 0, 0, -10, -5, 5,
    0, 0, 0, 20, 20, 0, 0, 0,
    5, 5, 10, 25, 25, 10, 5, 5,
    10, 10, 20, 30, 30, 20, 10, 10,
    50, 50, 50, 50, 50, 50, 50, 50,
    0, 0, 0, 0, 0, 0, 0, 0]


knightstable = [
    -50, -40, -30, -30, -30, -30, -40, -50,
    -40, -20, 0, 5, 5, 0, -20, -40,
    -30, 5, 10, 15, 15, 10, 5, -30,
    -30, 0, 15, 20, 20, 15, 0, -30,
    -30, 5, 15, 20, 20, 15, 5, -30,
    -30, 0, 10, 15, 15, 10, 0, -30,
    -40, -20, 0, 0, 0, 0, -20, -40,
    -50, -40, -30, -30, -30, -30, -40, -50]
bishopstable = [
    -20, -10, -10, -10, -10, -10, -10, -20,
    -10, 5, 0, 0, 0, 0, 5, -10,
    -10, 10, 10, 10, 10, 10, 10, -10,
    -10, 0, 10, 10, 10, 10, 0, -10,
    -10, 5, 5, 10, 10, 5, 5, -10,
```

```
    -10, 0, 5, 10, 10, 5, 0, -10,
    -10, 0, 0, 0, 0, 0, 0, -10,
    -20, -10, -10, -10, -10, -10, -10, -20]
rookstable = [
    0, 0, 0, 5, 5, 0, 0, 0,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    5, 10, 10, 10, 10, 10, 10, 5,
    0, 0, 0, 0, 0, 0, 0, 0]
queenstable = [
    -20, -10, -10, -5, -5, -10, -10, -20,
    -10, 0, 0, 0, 0, 0, 0, -10,
    -10, 5, 5, 5, 5, 5, 0, -10,
    0, 0, 5, 5, 5, 5, 0, -5,
    -5, 0, 5, 5, 5, 5, 0, -5,
    -10, 0, 5, 5, 5, 5, 0, -10,
    -10, 0, 0, 0, 0, 0, 0, -10,
    -20, -10, -10, -5, -5, -10, -10, -20]
kingstable = [
    20, 30, 10, 0, 0, 10, 30, 20,
    20, 20, 0, 0, 0, 0, 20, 20,
    -10, -20, -20, -20, -20, -20, -20, -10,
    -20, -30, -30, -40, -40, -30, -30, -20,
    -30, -40, -40, -50, -50, -40, -40, -30,
    -30, -40, -40, -50, -50, -40, -40, -30,
    -30, -40, -40, -50, -50, -40, -40, -30,
    -30, -40, -40, -50, -50, -40, -40, -30]
```

```
def endGame():
  if board.is_checkmate():
        if board.turn:
            return -9999
        else:
            return 9999
  if board.is_stalemate():
            return 0
  if board.is_insufficient_material():
            return 0


wp = len(board.pieces(chess.PAWN, chess.WHITE))
bp = len(board.pieces(chess.PAWN, chess.BLACK))
wn = len(board.pieces(chess.KNIGHT, chess.WHITE))
bn = len(board.pieces(chess.KNIGHT, chess.BLACK))
wb = len(board.pieces(chess.BISHOP, chess.WHITE))
bb = len(board.pieces(chess.BISHOP, chess.BLACK))
wr = len(board.pieces(chess.ROOK, chess.WHITE))
br = len(board.pieces(chess.ROOK, chess.BLACK))
wq = len(board.pieces(chess.QUEEN, chess.WHITE))
bq = len(board.pieces(chess.QUEEN, chess.BLACK))


material = 100 * (wp - bp) + 320 * (wn - bn) + 330 * (wb - bb) + 500 *
(wr - br) + 900 * (wq - bq)
pawnsq = sum([pawntable[i] for i in board.pieces(chess.PAWN,
chess.WHITE)])
pawnsq = pawnsq + sum([-pawntable[chess.square_mirror(i)]
                       for i in board.pieces(chess.PAWN, chess.BLACK)])
knightsq = sum([knightstable[i] for i in board.pieces(chess.KNIGHT,
chess.WHITE)])
knightsq = knightsq + sum([-knightstable[chess.square_mirror(i)]
```

```
                            for i in board.pieces(chess.KNIGHT,
chess.BLACK)])

bishopsq = sum([bishopstable[i] for i in board.pieces(chess.BISHOP,
chess.WHITE)])

bishopsq = bishopsq + sum([-bishopstable[chess.square_mirror(i)]

                            for i in board.pieces(chess.BISHOP,
chess.BLACK)])

rooksq = sum([rookstable[i] for i in board.pieces(chess.ROOK,
chess.WHITE)])

rooksq = rooksq + sum([-rookstable[chess.square_mirror(i)]

                        for i in board.pieces(chess.ROOK, chess.BLACK)])

queensq = sum([queenstable[i] for i in board.pieces(chess.QUEEN,
chess.WHITE)])

queensq = queensq + sum([-queenstable[chess.square_mirror(i)]

                          for i in board.pieces(chess.QUEEN,
chess.BLACK)])

kingsq = sum([kingstable[i] for i in board.pieces(chess.KING,
chess.WHITE)])

kingsq = kingsq + sum([-kingstable[chess.square_mirror(i)]

                        for i in board.pieces(chess.KING, chess.BLACK)])



def evaluate_board():

  eval = material + pawnsq + knightsq + bishopsq + rooksq + queensq +
kingsq

  if board.turn:

    return eval

  else:

    return -eval



def quiesce(alpha, beta):

    stand_pat = evaluate_board()

    if (stand_pat >= beta):

        return beta
```

```
    if (alpha < stand_pat):
        alpha = stand_pat
    for move in board.legal_moves:
        if board.is_capture(move):
            board.push_san(move)
            score = -quiesce(-beta, -alpha)
            board.pop()
    if (score >= beta):
                return beta
    if (score > alpha):
               alpha = score
    return alpha


def alphabeta(alpha, beta, depthleft):
    bestscore = -9999
    if (depthleft == 0):
        return quiesce(alpha, beta)
    for move in board.legal_moves:
        board.push_san(move)
        score = -alphabeta(-beta, -alpha, depthleft - 1)
        board.pop()
        if (score >= beta):
            return score
        if (score > bestscore):
            bestscore = score
        if (score > alpha):
            alpha = score
    return bestscore


def selectmove():
```

```
  try:
    move =
chess.polyglot.MemoryMappedReader("human.bin").weighted_choice(board).mov
e
    return move
  except:
    bestMove = chess.Move.null()
    bestValue = -99999
    alpha = -100000
    beta = 100000
    depth = 0
    for move in board.legal_moves:
        board.push_san(move)
        boardValue = -alphabeta(-beta, -alpha, depth - 1)
        if boardValue > bestValue:
            bestValue = boardValue
            bestMove = move
        if (boardValue > alpha):
            alpha = boardValue
        board.pop()
    return bestMove


    # Remaining imports
import traceback
from flask import Flask, Response, request
import webbrowser
# Searching Ai's Move
def aimove():
    move = selectmove()
    board.push_san(move)
# Searching Stockfish's Move
```

```
def stockfish():
    engine = chess.engine.SimpleEngine.popen_uci(
        "stockfish.exe")
    move = engine.play(board, chess.engine.Limit(time=0.1))
    board.push_san(move.move)
app = Flask(__name__)
# Front Page of the Flask Web Page
@app.route("/")
def main():
    global count, board
    ret = '<html><head>'
    ret += '<style>input {font-size: 20px; } button { font-size: 20px;
}</style>'
    ret += '</head><body>'
    ret += '<img width=510 height=510
src="/board.svg?%f"></img></br></br>' % time.time()
    ret += '<form action="/game/" method="post"><button name="New Game"
type="submit">New Game</button></form>'
    ret += '<form action="/undo/" method="post"><button name="Undo"
type="submit">Undo Last Move</button></form>'
    ret += '<form action="/move/"><input type="submit" value="Make Human
Move:"><input name="move" type="text"></input></form>'
    ret += '<form action="/dev/" method="post"><button name="Comp Move"
type="submit">Make Ai Move</button></form>'
    ret += '<form action="/engine/" method="post"><button name="Stockfish
Move" type="submit">Make Stockfish Move</button></form>'
    return ret
# Display Board
@app.route("/board.svg/")
def board():
    return Response(chess.svg.board(board=board, size=700),
mimetype='image/svg+xml')
# Human Move
@app.route("/move/")
```

```
def move():
    try:
        move = request.args.get('move', default="")
        board.push_san(move)
    except Exception:
        traceback.print_exc()
    return main()
# Make Ai’s Move
@app.route("/dev/", methods=['POST'])
def dev():
    try:
        aimove()
    except Exception:
        traceback.print_exc()
    return main()
# Make UCI Compatible engine's move
@app.route("/engine/", methods=['POST'])
def engine():
    try:
        stockfish()
    except Exception:
        traceback.print_exc()
    return main()
# New Game
@app.route("/game/", methods=['POST'])
def game():
    board.reset()
    return main()
# Undo
@app.route("/undo/", methods=['POST'])
def undo():
```

```
    try:
        board.pop()
    except Exception:
        traceback.print_exc()
    return main()


board = chess.Board()
app.run(debug=True)
```

Codigo da machine learning/ linguagem de maquina:

```
import chess
import chess.engine
import random
import numpy



# this function will create our x (board)
def random_board(max_depth=200):
  board = chess.Board()
  depth = random.randrange(0, max_depth)


  for _ in range(depth):
    all_moves = list(board.legal_moves)
    random_move = random.choice(all_moves)
    board.push(random_move)
    if board.is_game_over():
      break
```

```
  return board



# this function will create our f(x) (score)
def stockfish(board, depth):
  with chess.engine.SimpleEngine.popen_uci('stockfish/13/bin/stockfish')
as sf:
    result = sf.analyse(board, chess.engine.Limit(depth=depth))
    score = result['score'].white().score()
    return score
board = random_board()
board
```

print(stockfish(board, 10))

```
import tensorflow.keras.models as models
import tensorflow.keras.layers as layers
import tensorflow.keras.utils as utils
import tensorflow.keras.optimizers as optimizers



def build_model(conv_size, conv_depth):
  board3d = layers.Input(shape=(14, 8, 8))


  # adding the convolutional layers
  x = board3d
  for _ in range(conv_depth):
    x = layers.Conv2D(filters=conv_size, kernel_size=3, padding='same',
activation='relu')(x)
  x = layers.Flatten()(x)
```

```
  x = layers.Dense(64, 'relu')(x)
  x = layers.Dense(1, 'sigmoid')(x)


  return models.Model(inputs=board3d, outputs=x)
def build_model_residual(conv_size, conv_depth):
  board3d = layers.Input(shape=(14, 8, 8))


  # adding the convolutional layers
  x = layers.Conv2D(filters=conv_size, kernel_size=3,
padding='same')(board3d)
  for _ in range(conv_depth):
    previous = x
    x = layers.Conv2D(filters=conv_size, kernel_size=3,
padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv2D(filters=conv_size, kernel_size=3,
padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Add()([x, previous])
    x = layers.Activation('relu')(x)
  x = layers.Flatten()(x)
  x = layers.Dense(1, 'sigmoid')(x)


  return models.Model(inputs=board3d, outputs=x)
import tensorflow.keras.callbacks as callbacks



def get_dataset():
    container = numpy.load('dataset.npz')
```

```
    b, v = container['b'], container['v']
    v = numpy.asarray(v / abs(v).max() / 2 + 0.5, dtype=numpy.float32) # normalization (0 - 1)
    return b, v



x_train, y_train = get_dataset()
x_train.transpose()
print(x_train.shape)
print(y_train.shape)
from tensorflow.keras.callbacks import ModelCheckpoint
model.compile(optimizer=optimizers.Adam(5e-4), loss='mean_squared_error')
model.summary()
checkpoint_filepath = '/tmp/checkpoint/'
model_checkpointing_callback = ModelCheckpoint(
    filepath = checkpoint_filepath,
    save_best_only= True,
)
model.fit(x_train, y_train,
          batch_size=2048,
          epochs=1000,
          verbose=1,
          validation_split=0.1,
          callbacks=[callbacks.ReduceLROnPlateau(monitor='loss', patience=10),
                     callbacks.EarlyStopping(monitor='loss', patience=15, min_delta=1e-4),model_checkpointing_callback])


model.save('model.h5')
# used for the minimax algorithm
def minimax_eval(board):
```

```
  board3d = split_dims(board)
  board3d = numpy.expand_dims(board3d, 0)
  return model(board3d)[0][0]



def minimax(board, depth, alpha, beta, maximizing_player):
  if depth == 0 or board.is_game_over():
    return minimax_eval(board)

  if maximizing_player:
    max_eval = -numpy.inf
    for move in board.legal_moves:
      board.push(move)
      eval = minimax(board, depth - 1, alpha, beta, False)
      board.pop()
      max_eval = max(max_eval, eval)
      alpha = max(alpha, eval)
      if beta <= alpha:
        break
    return max_eval
  else:
    min_eval = numpy.inf
    for move in board.legal_moves:
      board.push(move)
      eval = minimax(board, depth - 1, alpha, beta, True)
      board.pop()
      min_eval = min(min_eval, eval)
      beta = min(beta, eval)
      if beta <= alpha:
        break
    return min_eval
```

```
# this is the actual function that gets the move from the neural network
def get_ai_move(board, depth):
  max_move = None
  max_eval = -numpy.inf


  for move in board.legal_moves:
    board.push(move)
    eval = minimax(board, depth - 1, -numpy.inf, numpy.inf, False)
    board.pop()
    if eval > max_eval:
      max_eval = eval
      max_move = move


  return max_move
# Testing code AI(white) vs Stockfish(black)
board = chess.Board()
from IPython.display import clear_output


with chess.engine.SimpleEngine.popen_uci('stockfish/7/bin/stockfish') as
engine:
    while True:
        clear_output(wait=True)
        move = get_ai_move(board, 1)
        board.push(move)
        print(f'\n{board}')
        if board.is_game_over():
            print('game_over')
            break
```

```
        move = engine.analyse(board, chess.engine.Limit(time=0.1),
info=chess.engine.INFO_PV)['pv'][0]
        board.push(move)
        print(f'\n{board}')
        if board.is_game_over():
            print('game_over')
            break
# Move by move testing code AI(white) vs Stockfish(black)
board = chess.Board()


with chess.engine.SimpleEngine.popen_uci('stockfish/13/bin/stockfish') as
engine:
    while True:
        move = get_ai_move(board, 1)
        board.push(move)
        print(f'\n{board}')
        if board.is_game_over():
            print('game_over')
            break
        move = engine.analyse(board, chess.engine.Limit(time=1),
info=chess.engine.INFO_PV)['pv'][0]
        board.push(move)
        print(f'\n{board}')
        if board.is_game_over():
            print('game_over')
            break
# Move by move testing code AI(white) vs Stockfish(black)
board = chess.Board()
from IPython.display import clear_output
```

```
with chess.engine.SimpleEngine.popen_uci('stockfish/13/bin/stockfish') as
engine:
    while True:
        clear_output(wait=True)
        move = get_ai_move(board, 1)
        board.push(move)
        print(move)
        print(f'\n{board}')
        if board.is_game_over():
            print('game_over')
            break
        input_var = input()
        move = chess.Move.from_uci(input_var)
        board.push(move)
        print(move)
        print(f'\n{board}')
        if board.is_game_over():
            print('game_over')
            break
```