Classificação Binária de Vantagem Posicional no Xadrez Com Modelo Perceptron Multicamadas

Lucas Mairon O. Camilo
Bacharelado em Engenharia de Software
Universidade Federal Rural do Semi-árido (UFERSA)
Rio Grande do Norte, Brasil
lucasmairon2@gmail.com

Rosana C. B. Rego
Departamento de Engenharias e Tecnologia
Universidade Federal Rural do Semi-Árido (UFERSA)
Rio Grande do Norte, Brasil
rosana.rego@ufersa.edu.br

***Resumo*—O xadrez é um jogo de tabuleiro que possui um grande número de posições diferentes, assim, evidenciando que há a possibilidade de, somente com a observação humana, existir dificuldades quanto a verificação de qual lado está com mais vantagem. Assim, o presente trabalho traz o treinamento de um modelo de inteligência artificial, baseado em perceptron multicamadas, que tem o papel de, dado uma posição do tabuleiro, utilizando classificação binária, indicar qual lado do tabuleiro está com mais vantagem. Trazendo ao final os resultados do treinamento do modelo, com acurácia de 79.42%, precisão de 78.93%, recall de 80.36% e F1 score de 79.64%, o que evidencia um progresso na rede.**

***Index Terms*—xadrez, inteligência artificial, modelagem, simulação, aprendizado de máquina**

I. Introdução

O xadrez é um jogo de mesa mundialmente conhecido e jogado, possuindo um tabuleiro com 64 espaços organizados em uma matriz 8×8, e uma grande diversidade de jogadas possíveis. Onde, por meio delas, constrói-se a vantagem no tabuleiro, seja com as peças brancas ou pretas [1]. Isso se dá pelo fato de que cada peça do tabuleiro tem um valor específico de pontos. Tradicionalmente, utiliza-se o sistema de valoração de Reinfeld onde o peão tem valor $v_{\text{peão}} = 1$, cavalo e bispo $v_{\text{cavalo}} = v_{\text{bispo}} = 3$, torre $v_{\text{torre}} = 5$, e dama $v_{\text{dama}} = 9$, enquanto o rei tem valor infinito por ser a peça decisiva do jogo.

O tabuleiro de xadrez pode ser representado de forma visual, assim como citado anteriormente, sendo algo similar a Figura 1, que representa um tabuleiro em seu estado inicial.

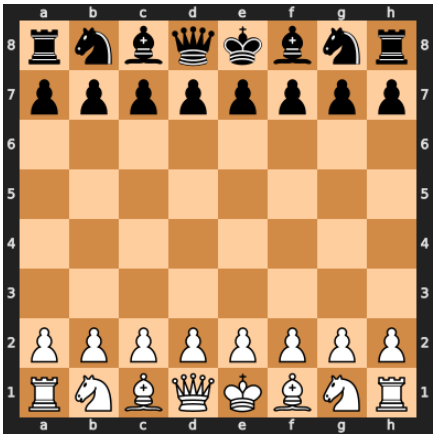

Figura 1. Tabuleiro de xadrez na sua posição original.

Porém, essa não é a única representação possível do tabuleiro, também existe a representação em FEN (*Forsyth-Edwards Notation*), uma sequência de caracteres que define as posições das peças em um estado da partida [2]. Formalmente, uma string FEN pode ser representada como uma tupla FEN $= (P, T, R, E, H, M)$, onde P é a configuração das peças, T indica o turno, R representa os direitos de roque, E a casa *en passant*, H o contador de meio-movimentos, e M o número do movimento completo. Um exemplo disso é a Figura 2, que representa o código FEN: rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R w KQkq c6 0 2. Onde os caracteres indicam em que posições as peças estão, turno de quem deve jogar, a disponibilidade de algumas jogadas, e outras informações, em uma sequencia que deve ser lida da esquerda para a direita.

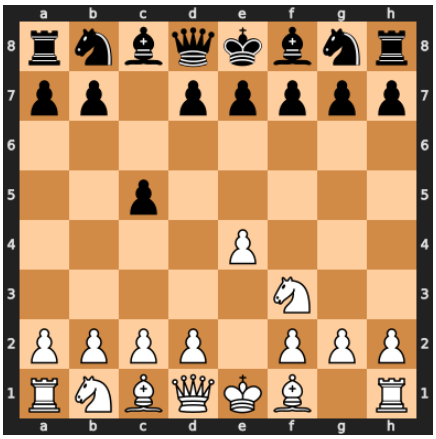

Figura 2. Tabuleiro de xadrez em uma posição específica.

Desta forma, utilizando o FEN citado acima podemos observar barras com sequências de caracteres, com cada barra representando uma linha do tabuleiro e os caracteres indicando

as peças que estão nos espaços entre elas, tendo cada linha 8 casas disponíveis. Logo, os caracteres como r, n, b, q, k e também suas versões em maiúsculo, trazem a inicial da peça que está no espaço, em inglês, como, por exemplo: *rook* (torre, em português) sendo r, *knight* (cavalo) sendo n, e outros. Assim, as letras minusculas indicam as peças pretas e as maiúsculas as brancas. Também observa-se a presença de números entre os caracteres, o que indica a quantidade de casas vazias presentes naquele intervalo [3].

Além disso, os caracteres finais como w, KQkq c6 0 e 2, indicam a ordem de quem é a vez de jogar, neste caso w(indicando a vez das brancas ou *white*), disponibilidade de realizar o roque, tendo KQkq como representação roque disponível tanto das brancas quanto das pretas, c6 representando a casa onde é possível realizar *en passant*, uma jogada especial do tabuleiro, o 0 indicando a quantidade de lances realizados após um movimento de peão ou peça capturada e por fim o 2, trazendo a informação de qual lance o jogo está [3].

Assim, a probabilidade de vitória em um tabuleiro de xadrez não se limita apenas a pontuação das peças, também pode incluir a chance de ganhar nos próximos lances, dependendo das jogadas que serão realizadas em seguida [1]. Matematicamente, pode-se expressar a avaliação de uma posição p como uma função $E(p)$, onde:

$$E(p) = \sum_i v_i \cdot n_i + f_{\text{posicional}}(p) + f_{\text{tática}}(p) \tag{1}$$

onde v_i é o valor da peça tipo i, n_i é a quantidade de peças desse tipo, $f_{\text{posicional}}$ captura vantagens posicionais e $f_{\text{tática}}$ representa oportunidades táticas. Com isso, o presente trabalho pretende traz o treinamento de um modelo de inteligência artificial para calcular, em uma posição selecionada do xadrez, entre as peças pretas e brancas, qual tem mais possibilidade de ganhar a partida.

O interesse quanto a este estudo se deu da curiosidade do presente autor em entender mais sobre as jogadas de xadrez, além de poder, por meio do modelo treinado, aprender mais sobre quais movimentos são mais adequados para uma posição. Também podendo impactar a comunidade de jogadores por meio do mecanismo.

Desta forma, os objetivos do presente trabalho são voltados a criação e treinamento de um modelo de inteligência artificial capaz de, por meio de posições do tabuleiro de xadrez, indicar qual dos lados está com vantagem, se são as peças pretas ou brancas.

II. Levantamento Bibliográfico

O uso de inteligência artificial para interações quanto as jogadas do xadrez também é discutido por meio de outros trabalhos, destacando assim a contribuição que a proposta pode trazer. Assim, os próximos parágrafos trazem trabalhos que possam contribuir e impactar com a presente proposta.

Silver et al. [4] destacaram o xadrez quanto aos testes de inteligência artificial. Assim, utilizando o AlphaZero para demonstrar que modelos de inteligência artificial treinados exclusivamente por jogadas do tabuleiro, sem conhecimento humano, pode vencer até mesmo campeões mundiais. Desta forma, destacando o potencial do aprendizado de máquina na área.

Sabatelli et al. [5] trazem um trabalho com a ideia de treinar modelos de inteligência artificial para avaliar posições de xadrez com base no stockfish. Desta formando, utilizando regressão para a resolução de problema, como também comparando algumas arquiteturas.

Pires et al. [6] apresentam a avaliação do desempenho de diferentes algoritmos de aprendizado de máquina quanto a predição do resultado em relação a uma partida de xadrez, tendo com base apenas uma posição do tabuleiro. Demonstrando assim como o tipo de rede proposto no presente trabalho pode realmente aprender padrões do tabuleiro e ser treinado com certa precisão.

Lai [7] traz o destaque quanto a organização de dados, já que, assim como o presente trabalho, o autor também decidiu fazer a passagem de dados quanto as posições sem utilizar imagens de estados do tabuleiro, introduzindo assim parâmetros a mais para auxiliar no processo. Assim, se torna evidente que esse tipo de abordagem também pode ser funcional quanto ao treinamento de redes neurais nesse tipo de solução.

III. Modelo de Rede Neural Proposto

A. Detalhamento do Problema

Dado a complexidade das jogadas de xadrez, a olho humano, muitas vezes torna-se difícil conseguir identificar qual o lado está com a vantagem em relação à posição das peças[3]. Portanto, o modelo desenvolvido pretende resolver o problema de identificação de vantagem estratégica em posições do tabuleiro de xadrez, utilizando classificação binária para isso.

Formalmente, o problema pode ser definido como: dado um conjunto de dados $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, onde $\mathbf{x}_i \in \mathbb{R}^{70}$ representa a codificação de uma posição do tabuleiro e $y_i \in \{0, 1\}$ é o rótulo da classe (0 para vantagem das pretas, 1 para vantagem das brancas), deseja-se aprender uma função $f_\theta : \mathbb{R}^{70} \to [0, 1]$ parametrizada por θ que minimize o erro de classificação sobre o conjunto de dados. O objetivo é encontrar os parâmetros ótimos:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} \left[L(f_\theta(\mathbf{x}), y) \right] \tag{2}$$

onde L é a função de perda escolhida (Binary Cross-Entropy) e $\mathbb{E}$ representa o valor esperado em relação à distribuição dos dados.

B. Organização dos dados

Os dados utilizados para o treino do modelo estão disponíveis por meio do link: https://www.kaggle.com/datasets/ronakbadhe/chess-evaluations. O dataset utilizado possui cerca de 16 milhões de dados distribuídos em duas colunas, sendo elas a FEN, representando uma posição do tabuleiro, e *evaluations*(avaliações), contendo o valor da posição.

Dos quase 16 milhões de dados foram filtrados 250 mil, com a intenção de coletar uma grande número de dados, sem prejudicar significativamente o desempenho da rede. Os

dados da coluna de avaliações do *dataset* foram convertidos em escala binária por meio da função:

$$y = \begin{cases} 1, & \text{se } e > 0 \text{ (vantagem das brancas)} \\ 0, & \text{se } e < 0 \text{ (vantagem das pretas)} \end{cases} \tag{3}$$

onde e representa o valor de avaliação da posição, descartando assim posições equilibradas ($e = 0$). Desta forma, os dados foram balanceados por meio do mecanismo de *undersampling*, voltado a redução dos dados da classe majoritária para igualar a quantidade de dados presentes em ambas as classes[8].

Seja $\mathcal{D}_0 = \{(\mathbf{x}_i, 0)\}$ o conjunto de exemplos da classe 0 e $\mathcal{D}_1 = \{(\mathbf{x}_j, 1)\}$ da classe 1, com $|\mathcal{D}_0| \neq |\mathcal{D}_1|$. O *undersampling* cria um subconjunto balanceado $\mathcal{D}_{\text{bal}}$ tal que:

$$|\mathcal{D}_{\text{bal}} \cap \mathcal{D}_0| = |\mathcal{D}_{\text{bal}} \cap \mathcal{D}_1| = \min(|\mathcal{D}_0|, |\mathcal{D}_1|) \tag{4}$$

gerando um balanceamento de 50/50 dos dados.

Além disso, os dados da coluna FEN também passaram pelo processo de conversão, visto que são sequências de caracteres. Desta forma, cada posição FEN do tabuleiro se tornou um vetor de características $\mathbf{x} \in \mathbb{R}^{70}$, onde dentre essas 70 dimensões, 64 são destinados às peças ocupantes dos espaços do tabuleiro e o restante voltado a valores extras, como turno e vez, para indicar quem tem prioridade de jogada, saldo material, que indica as vantagens em relação aos valores das peças, e por fim a possibilidade de roque, uma jogada popular no xadrez e a mobilidade, voltado ao número de jogadas legais para a posição.

O conjunto de dados balanceado $\mathcal{D}_{\text{bal}}$ foi particionado em três subconjuntos disjuntos:

$$\mathcal{D}_{\text{bal}} = \mathcal{D}_{\text{treino}} \cup \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{teste}}, \quad \mathcal{D}_i \cap \mathcal{D}_j = \emptyset \text{ para } i \neq j \tag{5}$$

utilizados respectivamente para treinamento, validação durante o treinamento e avaliação final do modelo.

Desta forma, define-se uma função de mapeamento $f : \mathbb{R}^{70} \to [0, 1]$ onde:

$$f(\mathbf{x}) = \hat{y} = \begin{cases} > 0.5, & \text{indica vantagem das brancas} \\ < 0.5, & \text{indica vantagem das pretas} \end{cases} \tag{6}$$

onde $\mathbf{x}$ é a representação vetorial da posição FEN e $\hat{y}$ é a predição da rede neural.

C. Arquitetura do modelo

O modelo representa uma rede perceptron multicamadas (MLP), sendo assim um modelo sequencial. A arquitetura utilizada tem uma estrutura de 4 camadas, sendo a primeira camada com 128 neurônios, representando uma camada de entrada de dados, duas camadas ocultas de 64 neurônios cada, e uma camada de saída com apenas um neurônio, assim como indicado na Figura 3.

Matematicamente, a saída de cada camada l pode ser expressa como:

$$\mathbf{h}^{(l)} = \sigma^{(l)} \left(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} \right) \tag{7}$$

onde $\mathbf{W}^{(l)}$ são os pesos da camada l, $\mathbf{b}^{(l)}$ é o vetor de viés, $\mathbf{h}^{(l-1)}$ é a saída da camada anterior, e $\sigma^{(l)}$ é a função de ativação.

Para as camadas de entrada e ocultas, utiliza-se a função de ativação ReLU (Rectified Linear Unit):

$$\text{ReLU}(z) = \max(0, z) = \begin{cases} z, & \text{se } z > 0 \\ 0, & \text{caso contrário} \end{cases} \tag{8}$$

A camada de saída utiliza a função sigmoid para produzir uma probabilidade no intervalo $[0, 1]$:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{9}$$

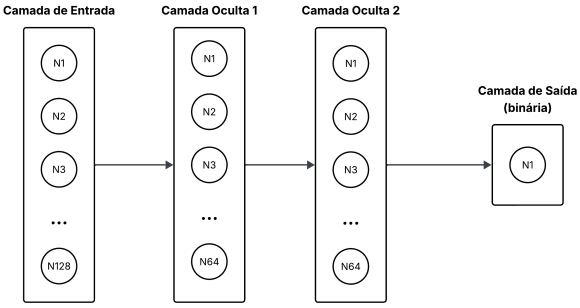

Figura 3. Representação das camadas da rede.

Dentre as camadas citadas também estão presentes as de *dropout* e *batch normalization*, utilizadas visando um maior desempenho da rede em relação a overfitting e as métricas em geral. O *dropout* aplica a operação:

$$\mathbf{h}_{\text{drop}} = \mathbf{h} \odot \mathbf{m}, \quad \mathbf{m} \sim \text{Bernoulli}(p) \tag{10}$$

onde $\odot$ representa a multiplicação elemento a elemento, $\mathbf{m}$ é uma máscara binária e p é a probabilidade de manter um neurônio ativo.

A *batch normalization* normaliza as ativações:

$$\text{BN}(\mathbf{h}) = \gamma \frac{\mathbf{h} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \tag{11}$$

onde μ_B e σ_B^2 são a média e variância do lote, γ e β são parâmetros aprendíveis, e ϵ é uma constante para estabilidade numérica.

D. Treinamento

O treinamento da rede foi realizado com 100 épocas, utilizando o mecanismo de parada forçada (*early stopping*) monitorando a validação da função perda. O critério de parada é definido como:

$$\text{parar se } L_{\text{val}}(t) > L_{\text{val}}(t - k) \text{ para todo } k \in \{1, 2, \ldots, p\} \tag{12}$$

onde $L_{\text{val}}(t)$ é a perda de validação na época t e p é a paciência do algoritmo (número de épocas sem melhoria antes de parar).

A função de perda utilizada é a *Binary Cross-Entropy*, definida como:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \quad (13)$$

onde N é o número de exemplos, y_i é o rótulo verdadeiro e $\hat{y}_i$ é a predição para o exemplo i.

Para evitar *overfitting*, aplica-se regularização L2 aos pesos da rede:

$$L_{\text{total}} = L(\mathbf{y}, \hat{\mathbf{y}}) + \lambda \sum_{l} \|\mathbf{W}^{(l)}\|_2^2 \quad (14)$$

onde λ é o coeficiente de regularização e $\|\mathbf{W}^{(l)}\|_2^2 = \sum_{i,j}(W_{ij}^{(l)})^2$ é o quadrado da norma L2 dos pesos.

O otimizador Adam atualiza os pesos segundo as regras:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla_{\mathbf{W}} L \quad (15)$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)(\nabla_{\mathbf{W}} L)^2 \quad (16)$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \quad (17)$$

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \quad (18)$$

onde α é a taxa de aprendizado, β_1 e β_2 são os coeficientes de decaimento dos momentos, e ϵ é uma constante para estabilidade numérica. Os valores atribuídos a estes parâmetros para a execução das simulações estão detalhados na Tabela I. O *batch size* utilizado foi de 256 exemplos.

Tabela I
VALORES DOS HIPERPARÂMETROS UTILIZADOS NAS SIMULAÇÕES

Parâmetro	**Valor**
Taxa de aprendizado (α)	0.0001
Decaimento do momento 1 (β_1)	0.9
Decaimento do momento 2 (β_2)	0.999
Constante numérica (ϵ)	10^{-8}
Tamanho do lote (*batch size*)	256

Os gradientes $\nabla_{\mathbf{W}} L$ são computados eficientemente usando o algoritmo de retropropagação (*backpropagation*), baseado na regra da cadeia do cálculo:

$$\frac{\partial L}{\partial W_{ij}^{(l)}} = \frac{\partial L}{\partial h_i^{(l)}} \cdot \frac{\partial h_i^{(l)}}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} \cdot h_j^{(l-1)} \quad (19)$$

onde $\delta_i^{(l)} = \frac{\partial L}{\partial h_i^{(l)}}$ é o erro retropropagado para o neurônio i da camada l.

O código contendo o treinamento do modelo está disponível no GitHub por meio da licença MIT, no link: https://github.com/ClassNeuralNetwork/classification-chess.

A arquitetura completa pode ser expressa como uma composição de funções:

$$\mathbf{h}^{(1)} = \text{ReLU}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \quad (20)$$

$$\mathbf{h}^{(2)} = \text{ReLU}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \quad (21)$$

$$\mathbf{h}^{(3)} = \text{ReLU}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \quad (22)$$

$$f_\theta(\mathbf{x}) = \sigma(\mathbf{W}^{(4)}\mathbf{h}^{(3)} + \mathbf{b}^{(4)}) \quad (23)$$

onde $\theta = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{4}$ representa todos os parâmetros treináveis da rede.

IV. Resultados e Simulações

A Figura 4 exibe o resultado da função perda aplicada ao modelo, onde, o eixo x representa a quantidade de épocas executadas, enquanto o eixo y apresenta o valor da função perda. Demonstrando assim um treinamento com um potencial, já que a linha de treino está bem alinhada com a linha de validação do modelo. A convergência da função perda indica que o processo de otimização gradiente descendente está minimizando efetivamente a função objetivo:

$$L_{\text{total}}(\theta) = L_{\text{treino}}(\theta) + \lambda\Omega(\theta) \quad (24)$$

onde L_{treino} é a perda no conjunto de treinamento e $\Omega(\theta)$ é o termo de regularização.

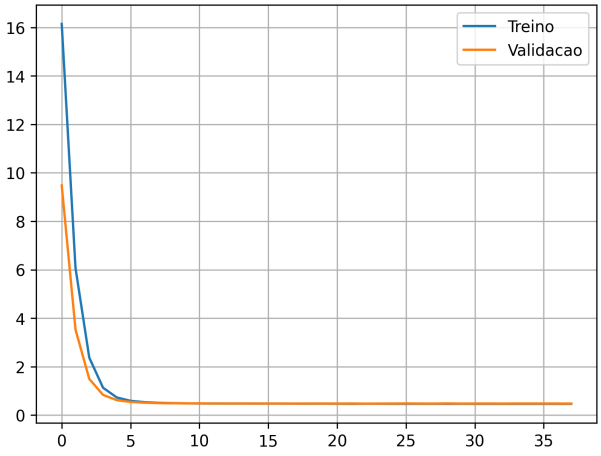

Figura 4. Função perda do modelo.

Por meio da Figura 5 podemos observar a matriz de confusão do modelo. A matriz de confusão é definida como:

$$C = \begin{bmatrix} \text{VP} & \text{FN} \\ \text{FP} & \text{VN} \end{bmatrix} \quad (25)$$

onde VP (verdadeiros positivos) representa predições corretas da classe 1, VN (verdadeiros negativos) da classe 0, FP (falsos positivos) são classificações incorretas como classe 1, e FN (falsos negativos) classificações incorretas como classe 0.

A matriz demonstra uma precisão considerável, ao conter a diagonal principal em destaque. Porém, ainda permitindo visualizar que o modelo está errando um considerável número de dados. Trazendo assim um número considerável tanto de falsos positivos quanto de falsos negativos, algo que se pretende investigar futuramente para ser melhorado.

Como é possível analisar por meio da Tabela II, as métricas, sendo elas acurácia, precisão, recall e f1-score, ficaram em torno de 0.78 a 0.80, indicando um bom desempenho da rede, mesmo que evidenciando a necessidade de fazer melhorias nela. Estas métricas são definidas matematicamente como:

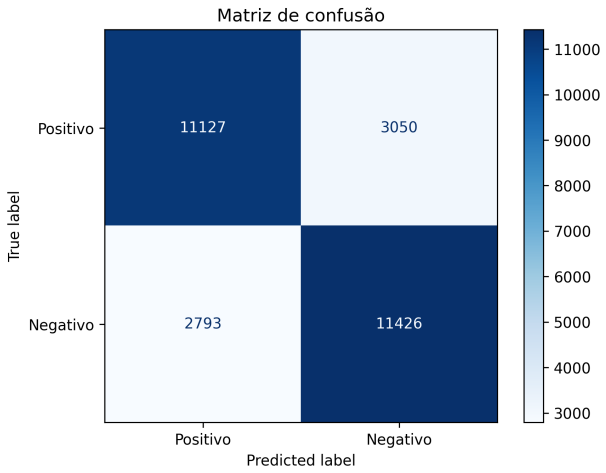

Figura 5. Matriz de confusão do modelo.

Acurácia: Proporção de predições corretas

$$\text{Acurácia} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{VN} + \text{FP} + \text{FN}} \tag{26}$$

Precisão: Proporção de predições positivas que são corretas

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{27}$$

Recall (Sensibilidade): Proporção de positivos verdadeiros corretamente identificados

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{28}$$

F1-Score: Média harmônica entre precisão e recall

$$\text{F1-Score} = 2 \cdot \frac{\text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}} = \frac{2\text{VP}}{2\text{VP} + \text{FP} + \text{FN}} \tag{29}$$

Tabela II
MÉTRICAS DE DESEMPENHO DO MODELO

Métrica	**Valor**
Acurácia	0.7942
Precisão	0.7893
Recall	0.8036
F1-score	0.7964

Tendo como base a Figura 6 podemos observar os impactos dos parâmetros passados no treinamento da rede, sendo eles obtidos ou por meio do FEN, com base no estado do tabuleiro, ou estando contidos nele. Assim, percebe-se o destaque que os parâmetros de saldo material, vez e mobilidade causam na rede, impactando diretamente para um treinamento melhor. Ao observar atentamente, também é possível notar que algumas casas, que representam os espaços do tabuleiro, geram impacto no treinamento, indicando assim que o treinamento não está sendo guiado apenas pelos parâmetros auxiliares.

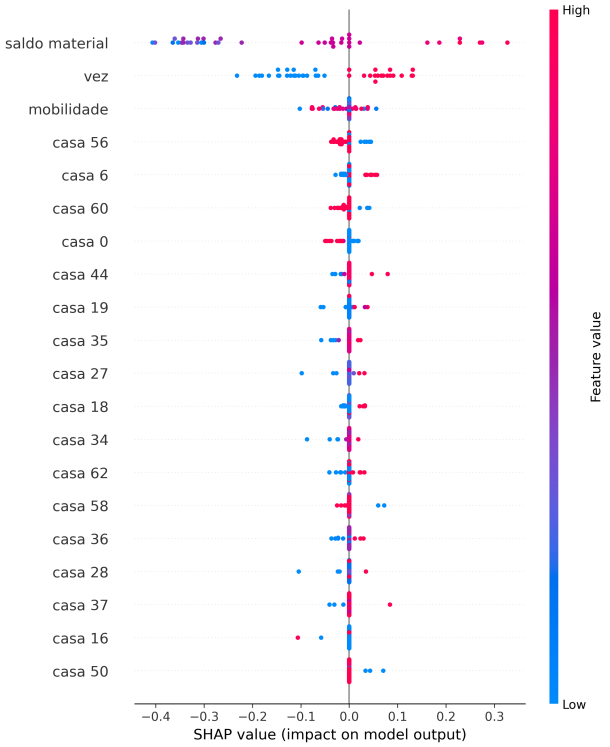

Figura 6. Impacto de parâmetros no treinamento da rede.

V. Conclusões

Por meio do exposto durante este trabalho percebe-se que o modelo já alcança bons desempenhos, podendo, ao ser aprimorado, ser utilizado para resolver problemas propostos. O modelo treinado atingiu uma taxa de acurácia de aproximadamente 79.42%, indicando que $P(\hat{y} = y) \approx 0.7942$ onde $\hat{y}$ é a predição do modelo e y é o rótulo verdadeiro. Este resultado demonstra que o modelo aprende padrões relevantes da estrutura posicional do xadrez, superando significativamente a baseline de classificação aleatória que teria acurácia de 50%.

Algumas das limitações quanto ao estudo realizado se trata da arquitetura da rede, já que, por ser uma perceptron multi camadas, não possui um desempenho adequado para o tipo de problema estudado quando comparado a arquiteturas mais sofisticadas como redes convolucionais ou transformers.

Como trabalhos futuros, pretende-se incrementar o modelo, almejando assim melhorar tanto suas métricas como os gráficos gerados para validação. Desta forma, melhorando os resultados do mesmo. Além disso, serão analisados novos tipos de redes para o problema, assim, podendo modificar o atual visando o aprimoramento do treinamento, já que pode se tornar mais eficiente.

Futuramente também pretende-se modificar a ideia para complementar as funcionalidades que o modelo poderá fazer, como, por exemplo, classificar qual próxima jogada seria a mais vantajosa, sugerir defesas e também apresentar justifica-

tivas quanto aos determinados tipos de respostas dados pelo modelo.

Uso de Inteligência Artificial

Em conformidade com as diretrizes de integridade científica e boas práticas acadêmicas recomendadas pela CAPES, os autores declaram que utilizaram ferramentas de inteligência artificial, incluindo Claude e Gemini, exclusivamente para auxiliar na melhoria da linguagem e legibilidade deste manuscrito. O uso dessas ferramentas foi realizado sob supervisão humana, e os autores revisaram e editaram cuidadosamente o conteúdo, assumindo total responsabilidade pelo texto final publicado.

Agradecimentos

Os autores agradecem à disciplina de Redes Neurais do curso de Engenharia de Software da Universidade Federal Rural do Semi-Árido (UFERSA), no âmbito da qual este trabalho foi desenvolvido.

Referências

[1] C. Castro, "Uma história cultural do xadrez," *Cadernos de teoria da comunicação, Rio de Janeiro*, vol. 1, no. 2, pp. 3–12, 1994.

[2] L. V. C. Zanella, "Tutor de xadrez, uma abordagem web," 2017.

[3] R. F. d. O. Cardozo, "Desenvolvimento de um agente de inteligência artificial para o jogo de xadrez baseado no algoritmo minimax," 2025.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[5] M. Sabatelli, F. Bidoia, V. Codreanu, and M. Wiering, "Learning to evaluate chess positions with deep neural networks and limited lookahead," in *7th International Conference on Pattern Recognition Applications and Methods*. SciTePress, 2018, pp. 276–283.

[6] H. Pires, H. Pedrini, R. Técnico-IC-PFG, and P. F. de Graduação, "Métodos de aprendizado de máquina aplicados ao jogo de xadrez," 2022.

[7] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *arXiv preprint arXiv:1509.01549*, 2015.

[8] M. S. Shelke, P. R. Deshmukh, and V. K. Shandilya, "A review on imbalanced data handling using undersampling and oversampling technique," *Int. J. Recent Trends Eng. Res*, vol. 3, no. 4, pp. 444–449, 2017.