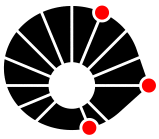
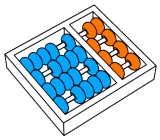




Métodos de Aprendizado de Máquina Aplicados ao Jogo de Xadrez

H. A. S. Pires *H. Pedrini*

Relatório Técnico - IC-PFG-22-32
Projeto Final de Graduação
2022 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Métodos de Aprendizado de Máquina Aplicados ao Jogo de Xadrez

Heigon Alafaire Soldera Pires* Hélio Pedrini†

Dezembro de 2022

Resumo

Desde a década de 50, são propostos métodos e soluções para programas serem capazes de jogar xadrez. Algoritmos de busca como o minimax são amplamente utilizados, em conjunto com funções de avaliações baseadas em heurísticas. Porém, estudos recentes na área, utilizando aprendizado de máquina, tem apresentado bons resultados. Este projeto tem como objetivo principal implementar métodos de aprendizado de máquina capazes de aprender a jogar xadrez, para assim, contribuir com estudos na área, ao entender quais são as vantagens e desvantagens em utilizar tais métodos. Para isso, o projeto apresenta uma contextualização histórica aos estudos desenvolvidos sobre o problema proposto e, para auxiliar a compreensão, alguns conceitos relacionados ao tema. O trabalho implementa um algoritmo clássico, amplamente utilizado para o desenvolvimento de programas modernos na área e, em sequência, implementa um método que substitui parte do algoritmo por um modelo de aprendizado de máquina que utiliza, em conjunto, a Rede de Crenças Profundas e a Rede Neural Siamesa. Inicialmente, o desenvolvimento das redes é realizado, com a coleta e o tratamento dos dados, sendo duas as fontes principais, a primeira é a plataforma de jogos online *Lichess* e a segunda são os jogos entre computadores do grupo CCRL. As arquiteturas das redes neurais são desenvolvidas e o processo de treinamento das mesmas é realizado e, por fim, os resultados finais obtidos são comparados entre as três abordagens. Finalmente, os benefícios que os modelos de aprendizado de máquina obtiveram são apresentados, mesmo não superando o método clássico, e novas propostas são sugeridas para avançar os estudos na área.

1 Introdução

O xadrez sempre obteve uma adoração por parte da matemática e computação. Desde a Máquina Analítica, proposta por Charles Babbage, Ada Lovelace já especulava sobre a capacidade da mesma de desenvolver tarefas complexas, como compor música ou jogar xadrez [1]. Sendo assim, o problema do *jogo de xadrez* foi amplamente estudado ao longo da história.

*Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP, Brasil, 13083-852.
†Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP, Brasil, 13083-852.

No início de 1950, tem-se o registro do primeiro artigo publicado sobre o assunto, *Programming a Computer for Playing Chess* por Claude Shannon [2], em que foram propostas algumas possíveis abordagens para sua solução, entre elas um algoritmo que utilizaria uma busca em árvore, com profundidade definida, em que o nó folha contém uma função de avaliação, composta por uma combinação linear de fatores que afetariam a qualidade da posição com valor positivo a uma das cores, que seria o jogador maximizador. Assim, a cada turno, essa cor escolheria o lance que maximiza a função de avaliação e a outra escolheria o lance que minimiza, semelhante ao algoritmo minimax com poda alfa-beta [3], hoje bem difundido em diversas *engines*, forma em que ficaram conhecidas as ferramentas computacionais capazes de jogar xadrez. Na mesma época, Alan Turing e David Champernowne desenvolveram a *Turochamp*, conhecida como a primeira *engine* desenvolvida, embora não capaz de executar em computador, em que cada passo do algoritmo foi simulado manualmente [4].

As *engines* desenvolvidas em computadores se iniciam por volta de 1970, como descrito em *Kasparov versus Deep Blue: Computer Chess Comes of Age* [5]. Embora o desafio de programar um computador a jogar xadrez não fosse muito grande, a qualidade do jogo era baixa, ao nível de jogadores amadores, Grande Mestres FIDE (maior titulação da Federação Internacional de Xadrez) acreditavam não ser possível superar essa qualidade. Em 1977, a Universidade Northwestern implementou o *CHESS* 4.6 [6] que chegou a ganhar de jogadores de níveis intermediário e alto. Assim, em 1997, a IBM desenvolveu o computador *Deep Blue* [7], máquina que utiliza diversos processadores para paralelizar a busca em árvore e analisar mais de 200 milhões de posições por segundo e conseguiu derrotar o, então campeão mundial, Garry Kasparov numa disputa de 6 partidas, com o resultado de 3,5 a 2,5.

Após esse feito, foi provado que as máquinas poderiam alcançar capacidades além das humanas no xadrez, entretanto, o *Deep Blue* foi um computador desenvolvido com o *hardware* específico para analisar centenas de milhões de posições. Dado isto, não era um *software* que podia simplesmente ser exportado para outro computador ou executar em computadores pessoais, assim, pesquisas continuaram para elevar o nível do jogo com a possibilidade de difundir esses programas.

Notando que o fator mais importante na qualidade de jogo era a função de avaliação, começaram a ser usados métodos de aprendizado de máquina para melhorar essa função, por exemplo, algoritmos genéticos para substituir os valores atribuídos aos diversos fatores de uma função [8], o que resultou em melhor qualidade, porém sendo ainda limitado ao uso de definição humana de quais fatores de uma partida deve-se considerar no cálculo da função.

Com o bom desempenho que o aprendizado por reforço demonstrou em jogos e com a difusão de redes neurais, algumas *engines* utilizaram aprendizado por reforço profundo [9], caso da *Giraffe* [10] em 2015, que também trocou o algoritmo minimax e busca baseada em profundidade por uma busca baseada em probabilidade, alcançando resultados de um Mestre Internacional FIDE (segunda maior titulação da Federação Internacional de Xadrez), entretanto, ainda não superando a capacidade humana.

A DeepMind, divisão da Google para inteligência artificial, em 2016, aplica o aprendizado por reforço e a busca em árvore Monte Carlo (MCTS) [11] ao jogo de tabuleiro chinês *go* para desenvolver a *AlphaGO* [12], superando o campeão mundial de *go* 19 anos após a *Deep Blue* derrotar o campeão mundial de xadrez, demonstrando a maior complexidade que esse

jogo possui em relação ao xadrez. Assim, ocorreram tentativas de aplicar esse algoritmo ao xadrez, porém, apesar de ser um jogo menos complexo, possui um caráter mais tático do que o jogo *go*, quando consideramos que o xadrez pode possuir muitas posições em que apenas um lance é possível para evitar uma derrota, por consequência, o fator probabilístico do algoritmo MCTS na análise da árvore poderia não analisar esse único lance que evita a derrota e, assim, o algoritmo não apresentava bons resultados.

Dessa maneira, os métodos de aprendizado por reforço estavam apresentando dificuldades no jogo de xadrez. Notando isso, ainda em 2016, foi desenvolvida a *DeepChess* [13] *engine* em que os desenvolvedores optaram por continuar utilizando o algoritmo minimax e apenas substituir a função de avaliação por duas arquiteturas de redes neurais, utilizaram um modelo de redes de crenças profundas (DBN) [14] que atua como um *autoencoder* ao realizar aprendizado não supervisionado para extrair características de uma posição, com duas posições de xadrez como entrada de duas DBNs e cada uma passando suas saídas como entradas para uma rede siamesa [15], a qual realizou um aprendizado supervisionado utilizando mais de 3 milhões de posições de jogos de xadrez com o rótulo de se o jogo resultou em vitória ou derrota para aprender a comparar duas posições e prever, qual teria maior probabilidade de levar a uma posição vantajosa. Essa abordagem alcançou resultado ao nível de Grande Mestre FIDE e, por ser treinada em jogos reais, apresentou a característica interessante de replicar escolhas que um humano com conhecimento avançado de xadrez tomaria, porém, ao se limitar a jogos humanos não seria possível atingir níveis superiores.

Em 2018, a DeepMind volta a utilizar o MCTS atualizado com melhorias e aprendizado por reforço, para desenvolver a *AlphaZero* [16], *engine* em que o treinamento foi totalmente baseado em jogos contra si mesmo, com algumas horas de treinamento superou níveis humanos e derrotou o *Stockfish* [17] repetidas vezes, a melhor *engine* de uso público atualmente e que, na época das partidas, estava na versão 8.

O *Stockfish* é uma *engine* de código aberta e disponível para uso público, sendo lançado em 2004 e idealizado para executar em CPU de diversos dispositivos e é amplamente difundido em diversas aplicações. Em 2020, após o avanço e o sucesso das redes neurais e sua derrota para o *AlphaZero*, lançou a versão 12, que parou de utilizar a função de avaliação clássica e trocou por uma arquitetura de rede neural que apresentou bons resultados no jogo de tabuleiro japonês *shogi*, conhecido como NNUE [18], por ser bem otimizada em cálculos de CPU, apresentou resultados melhores que a versão anterior e se manteve versátil. Isso foi importante para o *Stockfish*, devido à utilização do mesmo em dispositivos que podem possuir limitações de *hardware*, como aplicações em dispositivos móveis, jogos online e robótica.

Dada a variedade de soluções propostas e implementações para o problema, este projeto tem como principal objetivo avaliar os benefícios e prejuízos de se utilizar abordagens com o uso de métodos de aprendizado de máquina para desenvolver uma *engine* de xadrez. Com isso, tem como objetivo também, contribuir para os estudos de tal área, ao estudar e realizar comparações entre um método clássico, proposto na década de 50. Assim, caracteriza-se por um escopo que já vem sendo amplamente utilizado e estudado, e um novo método, recentemente proposto, que utiliza redes neurais em sua implementação. Por fim, almeja concluir também as vantagens e limitações desse novo modelo e propondo possíveis trabalhos futuros para o avanço dos estudos realizados neste projeto. Para isso, foram estudadas as soluções

mencionadas e a evolução da computação aplicada ao jogo de xadrez, para implementar, utilizando a linguagem de programação Python e suas bibliotecas, os algoritmos propostos e o ambiente capaz de realizar jogos de xadrez e executar os algoritmos, sendo assim, possível coletar métricas dos resultados obtidos para a formulação da análise final.

Para apresentação clara do estudo realizado, o texto se organiza da seguinte forma. A presente Seção 1 apresenta uma revisão histórica sobre os estudos aplicados na área da matemática e computação sobre jogos de tabuleiro, em especial, o xadrez, apresentando os desafios e a motivação para realizar um estudo sobre aprendizado de máquina nessa área, além de possuir os objetivos do projeto. A Seção 2 descreve alguns fundamentos e conceitos utilizados para a implementação do programa proposto, elucidando o algoritmo clássico amplamente utilizado, sendo ele o minimax com poda alfa-beta, apresentando também as arquiteturas de redes neurais presentes neste projeto. A Seção 3, a qual é subdividida segundo os métodos e materiais utilizados, inicia apresentando como foi desenvolvida a interface gráfica utilizada para testes. Apresenta também, a implementação do algoritmo minimax com poda alfa-beta e os resultados obtidos no primeiro dos tópicos. A Seção 4 apresenta a abordagem utilizando redes neurais, explicitando as arquiteturas implementadas, a coleta de dados para os treinamentos realizados e os resultados obtidos. Por fim, nesta seção ainda, são comparados os resultados de ambas as abordagens. Finalmente, a Seção 5 apresenta as considerações finais e a conclusão do projeto, além de indicar possíveis trabalhos futuros a serem realizados.

2 Conceitos Relacionados

Esta seção descreve brevemente conceitos relevantes associados ao tópico investigado neste trabalho.

2.1 Conceitos e Termos Relacionados ao Jogo de Xadrez

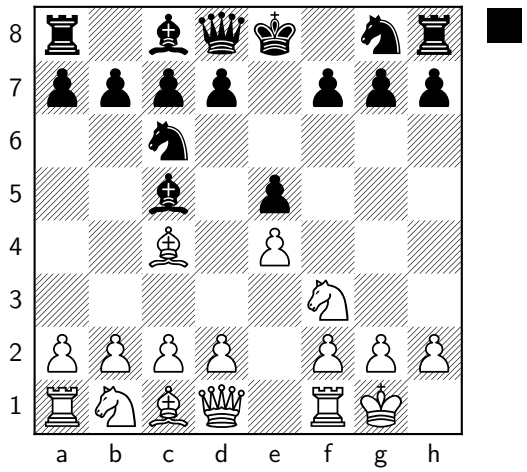
Para a plena compreensão deste projeto, além do conhecimento básico das regras do xadrez, como movimento das peças, posição inicial e o que é cheque-mate, será necessário familiaridade com alguns outros conceitos e termos. Então, esta seção contém explicações sobre os principais formatos de jogo, como é realizada a avaliação de um jogador e o que são as titulações, conceitos importantes para análises, as três fases do jogo e algumas notações relevantes.

Em partidas oficiais, os jogadores possuem tempo limitado para realizar os seus lances e a quantidade de tempo disponível determina o formato do jogo, como também, a qualidade das partidas realizadas e média de erros cometidos. Nas partidas clássicas, cada jogador possui cerca de 2 horas e 30 minutos para jogarem, totalizando cerca de 5 horas de jogo. Porém, as partidas mais jogadas hoje em dia, principalmente em plataformas online, possuem menos tempo. Os principais formatos são partidas rápidas, cada jogador possuindo cerca de 10 a 15 minutos, partidas *blitz* de 3 a 5 minutos e *bullet*, no qual, cada jogador possui 1 minuto.

Em seguida, são explicados conceitos que não fazem parte da regra do jogo, entretanto, são amplamente utilizados para a análise de uma partida. A primeira é a análise do material, a qual consiste nas peças presentes no tabuleiro e cada uma possui uma pontuação associada,

peões valem 1 ponto, cavalos e bispos 3 pontos, torres 5 pontos e a dama 9 pontos. A próxima análise é a mobilidade, esta constitui na quantidade de casas disponíveis para as peças moverem, sendo ainda mais importante, as 4 casas centrais (d4, d5, e4 e e5) estarem entre elas. Por último, a proteção do rei, esta, geralmente, compreende em realizar o lance chamado de roque, último lance da Figura 1. Neste único lance, duas peças são movimentadas, o rei move duas casas em direção a uma das torres e essa torre move para o outro lado do rei, dependendo da direção é chamado de roque curto ou roque longo.

Uma partida dividi-se em três fases e cada uma possui conceitos e desafios próprios, sendo elas a abertura, o meio-jogo e o final. A primeira fase, a abertura, constitui-se dos primeiros lances realizados no início da partida, sendo uma das fases que possui maior estudo teórico, entretanto, a escolha da abertura depende, principalmente, do estilo pessoal do jogador. A Figura 1 ilustra o exemplo da abertura italiana. Após essa fase, encontra-se o meio-jogo, onde ocorre a maioria das trocas de peças e as estratégias táticas de cada jogador, com o objetivo de levar para a próxima fase com vantagem. Por fim, a fase final, na qual ambos jogadores, provavelmente, possuirão pouco material e devem encurralar o rei adversário com o que possuem, caso um dos jogadores consiga, o jogo termina por cheque-mate ou desistência e, caso contrário, o jogo termina em empate.




```
1.e4 e5 2.Nf3 Nc6 3.Bc4 Bc5 4.O-O
```

Figura 1: Abertura Italiana com a notação PGN.

Uma forma de desenvolver as habilidades em cada fase do jogo, além de estudo teórico, é solucionar problemas, do inglês *puzzles*, durante o treino. Os problemas consistem em tabuleiros organizados com posições específicas, na qual, possuem uma única sequência que manterá vantagem para alguma das cores, assim, o desafio consiste em encontrar essa solução correta. Os problemas podem estar, também, acompanhados de determinados temas que definem, por exemplo, a que fase de jogo o mesmo trata.

Para avaliar o desempenho e nível dos jogadores, a Federação Internacional de Xadrez FIDE (*Fédération Internationale des Échecs*) utiliza o método de *Rating Elo*, desenvolvido

pelo físico Arpad Elo em 1978 [19]. Esse método atribui uma pontuação numérica a um jogador para medir sua habilidade em relação aos outros. As titulações são definidas através do *rating* FIDE de cada jogador, a titulação de Grande-Mestre é concedida após 2500 de *rating*.

Notações importantes do xadrez que são usados neste projeto são a PGN e a FEN. A notação PGN (*Portable Game Notation*) é utilizada para registrar os lances ocorridos na partida, como na Figura 1, em que está ilustrado o PGN da abertura italiana. Essa notação possui, também, um formato de arquivo digital, no qual pode salvar os lances, o tempo que cada lance ocorreu e outros informações dos jogadores. Além dessa, a notação FEN (*Forsyth-Edwards Notation*) proporciona uma forma de descrever uma posição estática com todas as informações necessárias para continuar a partida. Por exemplo, o FEN para descrever a posição na Figura 1 é a seguinte:

```
r1bqk1nr/pppp1ppp/2n5/2b1p3/2B1P3/5N2/PPPP1PPP/RNBQ1RK1 b kq - 5 4
```

Cada sequência de caracteres entre as barras representam uma fileira do tabuleiro, iniciando da fileira de número 8. As letras representam qual peça ocupa a casa, minúsculas para peças pretas e maiúsculas para brancas, e os números identificam quantidade de casas em branco na fileira. Após isso, a notação possui, respectivamente, uma letra para identificar qual jogador move nesse turno, quais roques estão disponíveis, se possui o movimento *en-passant* disponível, quantos movimentos realizados após a última captura ou lance de peão e, por fim, quantos turnos ambos jogadores concluíram até o momento.

2.2 Algoritmo Minimax com Poda Alfa-Beta

O algoritmo Minimax [20], pertencente às áreas de estudos em teoria da decisão [21] e teoria dos jogos [22], é amplamente utilizado em jogos baseados em turnos de dois jogadores, como o xadrez [23], para encontrar o melhor lance possível para cada um, assumindo que ambos os jogadores irão escolher o lance ótimo. Nomeia-se dessa forma por possuir o caráter de minimizar a perda máxima de uma determinada função de avaliação, normalmente, baseada em heurísticas [24] com valor positivo em relação a vantagem do jogador maximizador. Desta maneira, esse algoritmo consiste em uma busca recursiva de profundidade limitada na árvore de decisões, na qual, os nós folhas possuem o valor da função de avaliação. Cada nó percorrido equivale ao turno de um dos jogadores, sendo alternado entre o jogador maximizador, que irá escolher o caminho na árvore que leva ao maior valor possível, e o minimizador, que fará o oposto, como apresentado no Algoritmo 1.

Em sequência, analisaremos um exemplo de uma utilização do algoritmo com profundidade 3. A Figura 2 ilustra a árvore de decisões que será utilizada, em que os nós folhas possuem o valor da função de avaliação do estado do jogo no momento que o nó for alcançado e os demais serão preenchidos com o valor do melhor caminho a seguir para cada jogador, que se inicia com o maximizador e é alternado a cada nível de profundidade.

Realizando a busca recursivamente, cada nó é visitado e salvo nele o melhor valor de seus filhos. O jogador maximizador sempre escolherá o maior valor possível, enquanto o minimizador escolherá o oposto. Ao fim, tem-se a árvore preenchida como mostrado na Figura 3.

Algoritmo 1: Minimax.

```
Minimax(profundidade, maximizador)
    if profundidade = 0 then
        valor = função de avaliação
        return valor
    end
    else if maximizador = True then
        melhor_valor = −∞
        for filho do nó atual do
            valor = Minimax(profundidade - 1, False)
            melhor_valor = max(melhor_valor, valor)
        end
        return melhor_valor
    end
    else
        melhor_valor = +∞
        for filho do nó atual do
            valor = Minimax(profundidade - 1, True)
            melhor_valor = min(melhor_valor, valor)
        end
        return melhor_valor
    end
```

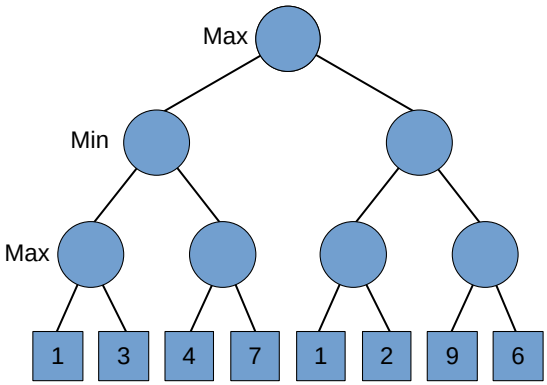


Figura 2: Árvore de decisões Minimax inicial. Inspirado na Figura 1 do artigo [25].

Assim, podemos ver que, mesmo existindo um nó folha com valor 9 na subárvore da direita, o melhor caminho a ser escolhido para o lance atual é o da esquerda (Figura 4), onde é possível alcançar o valor final 3, já que assumimos que o jogador oponente também irá escolher o caminho ótimo que não resultará no valor 9, mas no valor 2.

Apesar de eficiente para encontrar o melhor lance, existem variações desse algoritmo para diminuir o tempo de execução [26]. O mais difundido, atualmente, é a poda alfa-beta [27] utilizada para remover ramos da árvore sem modificar o resultado final. Ela se baseia no

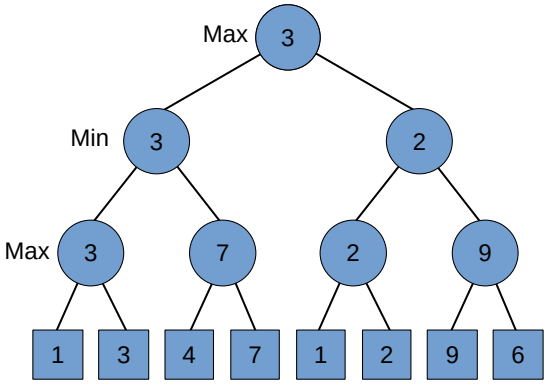


Figura 3: Árvore de decisões Minimax totalmente preenchida. Inspirado na Figura 2 do artigo [25].

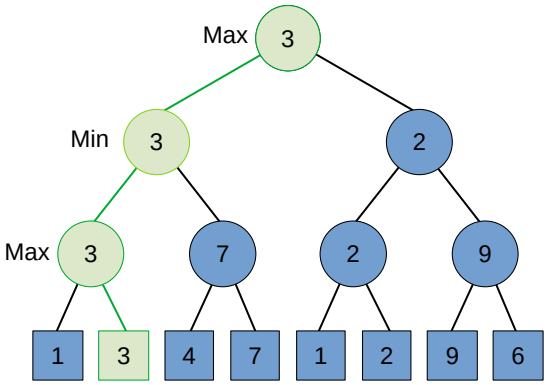


Figura 4: Árvore de decisões Minimax com o melhor caminho escolhido. Inspirado na Figura 2 do artigo [25].

princípio matemático: $\max\{\min\{5,7\},\min\{2,y\}\} = 5$, independentemente do valor de y, dado que $\min\{2,y\} \leq 2$. Sendo assim, o algoritmo utiliza dois novos parâmetros, o alfa para armazenar o melhor valor que o jogador maximizador pode obter até o momento e beta para o melhor valor do minimizador. Quando beta possuir um valor menor ou igual a alfa, não é mais necessário analisar os filhos do nó atual. Da mesma forma, a seguir teremos o pseudocódigo da poda alfa-beta (Algoritmo 2) e após isso, um exemplo.

A mesma árvore utilizada anteriormente (Figura 1) será preenchida, dessa vez, utilizando a poda alfa-beta (Figura 5), pode-se notar, em vermelho, ramos não analisados pelo algoritmo, pois no primeiro caso, na subárvore da esquerda, o jogador minimizador já havia encontrado o valor 3, ou seja, beta possuía o valor 3 quando o jogador maximizador encontrou o valor 4, assim, alfa recebeu o valor 4, como o $beta \leq alfa$, não seria mais necessário analisar os filhos desse nó. Na subárvore da direita, o mesmo acontece, quando o minimizador encontra o valor 2 e o maximizador já possui o valor 3 garantido, podando, assim, uma parcela ainda maior da árvore.

Algoritmo 2: Minimax com poda alfa-beta.

```
Minimax(profundidade, maximizador, alfa, beta)
    if profundidade = 0 then
        valor = função de avaliação
        return valor
    end
    else if maximizador = True then
        melhor_valor = −∞
        for filho do nó atual do
            valor = Minimax(profundidade - 1, False, alfa, beta)
            melhor_valor = max(melhor_valor, valor)
            alfa = max(alfa, melhor_valor)
            if beta ≤ alfa then
                break
            end
        end
        return melhor_valor
    end
    else
        melhor_valor = +∞
        for filho do nó atual do
            valor = Minimax(profundidade - 1, True, alfa, beta)
            melhor_valor = min(melhor_valor, valor)
            beta = min(beta, melhor_valor)
            if beta ≤ alfa then
                break
            end
        end
        return melhor_valor
    end
```

Por fim, é escolhido o mesmo caminho que anteriormente (Figura 6), entretanto, reduzindo a quantidade de análises realizadas. Há ainda outros estudos para melhoria do desempenho, como implementação utilizando paralelismo [28].

2.3 Aprendizado Profundo

O aprendizado profundo é uma das áreas de estudo de aprendizado de máquina [29], baseado em modelos computacionais, compostos de múltiplas camadas de processamento, capazes de aprender representações de dados com diversos níveis de abstração [30], nomeados de redes neurais.

Apesar de consistirem de camadas de processamento, existem diversas arquiteturas de redes neurais [31], que demonstraram vantagens e desvantagens em contextos diferentes. Componente importante para o aprendizado profundo são os algoritmos de retro-propagação,

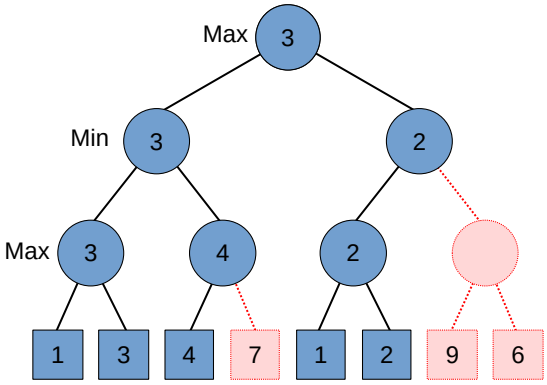


Figura 5: Árvore de decisões alfa-beta totalmente preenchida. Inspirado na Figura 4 do artigo [25].

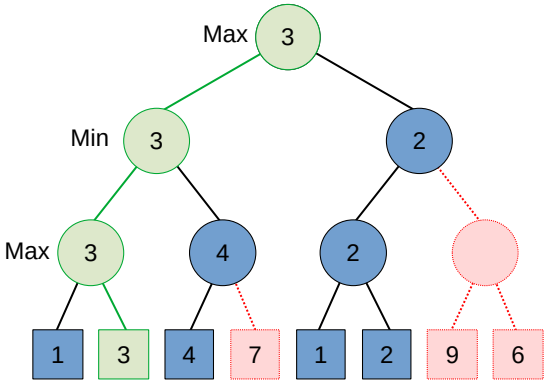


Figura 6: Árvore de decisões alfa-beta com o melhor caminho escolhido. Inspirado na Figura 4 do artigo [25].

do inglês *backpropagation* [32], usados para calcular o gradiente da função de perda [33] em relação aos pesos da rede para cada entrada, ajustando assim esses pesos para aproximar a saída real da saída esperada, portanto, treinando a rede neural. Assim como algoritmos de aprendizado de máquina, elas se dividem em aprendizado supervisionado, não supervisionado e semi supervisionado, entretanto, também se classificam em modelos discriminativos, capazes de discriminar entre diferentes instâncias dos dados, ou generativos, modelos que geram novas instâncias dos dados.

Neste projeto, a Rede de Crenças Profundas e a Rede Neural Siamesa foram utilizadas. Esta seção possui detalhes dos conceitos teóricos relacionados a cada uma.

2.3.1 Rede de Crenças Profundas

Em aprendizado de máquina, a extração de características [34] desempenha um papel importante, utilizando-se de diversas técnicas para desenvolver sistemas capazes de, automaticamente, aprender quais as características dos dados são relevantes para serem utilizadas

em outros métodos. Uma das técnicas utiliza *autoencoders* [35] para a extração mencionada, isto sendo, uma rede neural artificial usada para aprender uma codificação do dado de entrada, normalmente, através de aprendizado não supervisionado. O treinamento consiste em codificar a entrada, extraindo as principais características, e a validação e refinamento é feita ao reconstruir o dado a partir da codificação gerada, processo chamado de decodificação.

Em 2006, é proposta a Rede de Crenças Profundas, do inglês, *Deep Belief Network* (DBN) [36], na qual foi verificada a capacidade de aprender a reconstruir probabilisticamente os dados de entrada [37] e obteve bons resultados em extrair características [38]. A Figura 7 ilustra um exemplo da arquitetura de uma DBN, a qual consiste de diversas camadas concatenadas de Máquinas de Boltzmann Restritas (RBM) [39]. Uma RBM é constituída de duas camadas de neurônios, uma delas visível e a outra oculta, cada neurônio de uma camada é totalmente conectado a todos os neurônios de outra, entretanto, não existe nenhuma conexão entre neurônios da mesma camada. Sendo assim, o dado de entrada é codificado através das camadas ocultas da rede, para, no fim, obter-se uma saída menor, contendo a codificação da entrada com as principais características. A validação e o refinamento são realizados da mesma forma que um *autoencoder*, o dado de saída é então decodificado para ser o mais próximo possível do original (Figura 8).

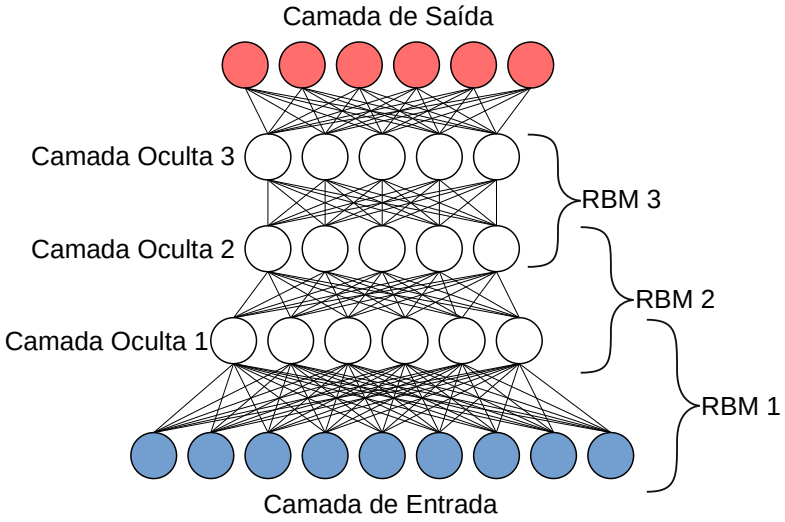


Figura 7: Arquitetura de uma Rede de Crenças Profundas. Inspirado na Figura 1 do artigo [38].

2.3.2 Rede Neural Siamesa

Uma Rede Neural Siamesa (SNN) consiste em uma arquitetura de redes neurais que possuem duas sub-redes iguais, ou seja, possuem a mesma arquitetura e os pesos compartilhados [40]. Elas são amplamente utilizadas para analisar similaridade entre duas entradas, como já demonstrado, pessoas tem facilidade em aprender rapidamente características de um número

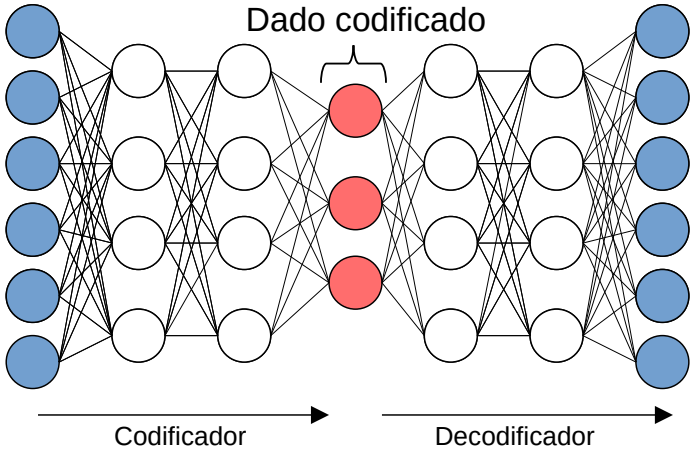


Figura 8: Treinamento de uma Rede de Crenças Profundas. Inspirado na Figura 5 do artigo [38].

pequeno de dados e identifica-las em similares [41], propriedade que métodos de aprendizado de máquina não possuem. No entanto, Redes Neurais Siamesas têm por objetivo suprimir essa deficiência ao necessitar de poucos dados [42] para apresentar resultados satisfatórios através de aprendizado supervisionado.

Essa arquitetura pode-se apresentar de diversas formas, pois as redes de pesos compartilhados, onde possuem a entrada dos dados, não exigem uma arquitetura definida, apenas necessita-se de duas redes idênticas, na Figura 9 é apresentado um exemplo de uma estrutura básica. Redes Siameses, no geral, além de apresentarem as redes de pesos compartilhados, apresentam também, camadas para unir ambas redes e calcular a diferença ou similaridade dos dados, podendo ser substituído por outros métodos e funções para calcular a similaridade baseada em algum critério desejado. Por fim, a saída pode conter dois elementos, como demonstrado, com um representando o valor verdade e outro falso para similaridade entre as entradas, entretanto, é comum, também, possuir apenas um elemento com o valor indicando o grau de similaridade. Como dito, é uma estrutura básica e pode ser alterada conforme o contexto desejado [43].

3 Material e Métodos

Esta seção se dedica aos métodos e materiais utilizados para o desenvolvimento do projeto, assim como os resultados obtidos em cada etapa de treinamento. Primeiramente, é apresentado o desenvolvimento da interface gráfica, assim como as bibliotecas usadas, para simular partidas de xadrez com a linguagem de programação Python e interagir com as ferramentas computacionais capazes de jogarem xadrez desenvolvidas. Essas serão apresentadas em sequência, a primeira consistindo de uma abordagem clássica e a segunda utilizando redes

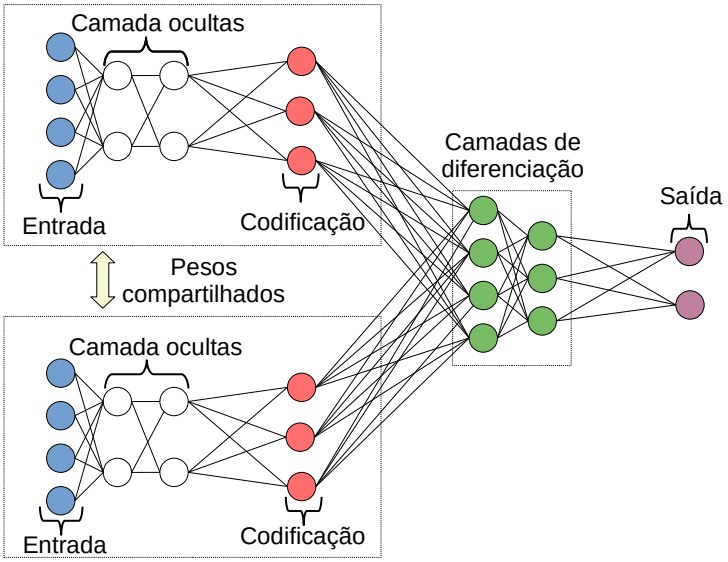


Figura 9: Arquitetura de uma Rede Neural Siamesa. Inspirado na Figura 3.1 do artigo [42].

neurais, baseadas no artigo [13].

3.1 Interface Gráfica

Para simular e controlar um tabuleiro de xadrez na linguagem de programação Python, foi utilizada a biblioteca *python-chess* 1.9.0 [44]. Com isto, podemos realizar qualquer tarefa que existe em tabuleiro real de xadrez, pode-se gerar um novo tabuleiro, mover as peças e validar os movimentos, verificar empate ou cheque-mate. Além de possuir métodos auxiliares para ler ou gerar notações comuns no xadrez, como o PGN, formato de arquivo para salvar registros de uma partida, e a notação FEN, utilizada para descrever o estado e distribuição das peças em um tabuleiro.

Em complemento a *python-chess*, foi utilizada a biblioteca *pygame* 2.1.2 [45] para desenvolver uma interface gráfica interativa (Figura 10) com a funcionalidade de uma pessoa jogar xadrez contra as *engines* desenvolvidas. Essa biblioteca consiste em um conjunto de outros módulos em Python com o objetivo de desenvolver ferramentas multimídia, principalmente jogos digitais, ao adicionar diversas funcionalidades na biblioteca SDL [46], a qual é escrita na linguagem de programação C para controle de gráficos, sons e dispositivos de entrada em diversas plataformas. Sendo assim, é possível desenvolver ferramentas interativas com uma boa responsividade gráfica, como a utilizada neste projeto.

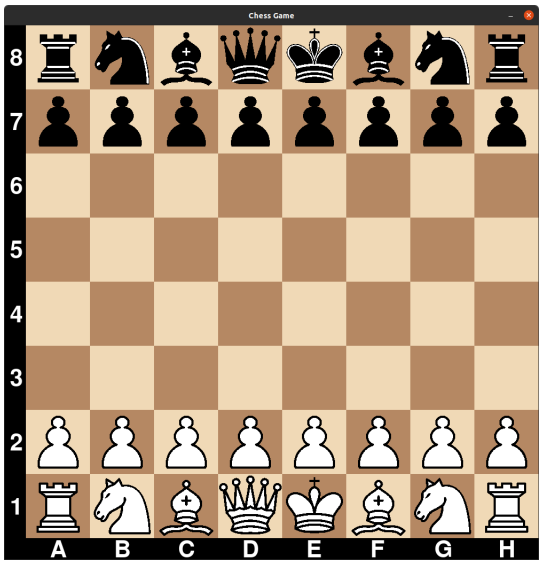


Figura 10: Interface gráfica interativa desenvolvida com as bibliotecas *pygame* e *python-chess*.

3.2 Abordagem Utilizando Métodos Clássicos

3.2.1 Implementação

Primeiramente, foi desenvolvida uma *engine* utilizando métodos amplamente estudados e sem uso de qualquer método de aprendizado de máquina [47]. Foi desenvolvida uma classe em Python com um método capaz de receber uma instância de um tabuleiro definido pela *python-chess* e procurar o melhor lance. A busca utiliza o algoritmo Minimax com poda alfa-beta (Algoritmo 2), recebendo como parâmetro a profundidade da busca e de qual jogador irá escolher o lance no turno atual, assim, realiza até chegar na profundidade definida, onde é chamada a função de avaliação baseada em heurística. Para este projeto, foi adotado o mesmo padrão que a literatura, ou seja, o jogador maximizador é aquele que controla as peças brancas e o minimizador as pretas.

A função de avaliação utilizada para esta abordagem é definida por uma combinação linear do material dos jogadores (quantidade de peças associadas aos valores das mesmas), segurança do rei, mobilidade das peças e controle do centro.

$$\begin{aligned} f(t) = & 200(R-r) + 9(D-d) + 5(T-t) + 3(B-b+C-c) + (P-p) \\ & + 0.01(M-m) + 0.05(CC-cc) + 0.25(RC-rc) + 0.15(RL-rl) \end{aligned} \tag{1}$$

em que (as letras maiúsculas são em relação às peças brancas e as minúsculas às pretas):
R: Rei em cheque-mate.
$DTBCP$: Quantidade de damas, torres, bispos, cavalos e peões no tabuleiro.
M: Quantidade de movimentos disponíveis.
CC: Quantidade de casas centrais atacadas.
RC: Roque curto.

RL: Roque longo.

Um exemplo de avaliação da função baseada em heurísticas, dado um tabuleiro t é ilustrado na Figura 11, com $f(t) = 6.1$.

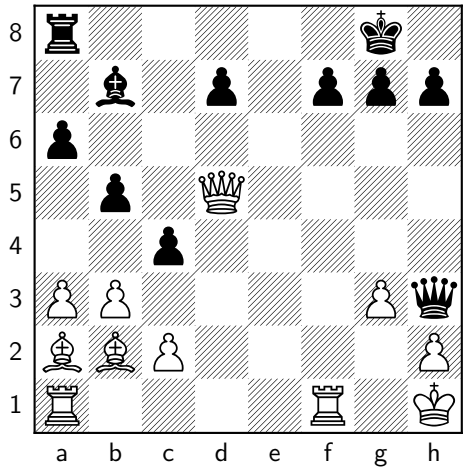


Figura 11: Estado de um tabuleiro de xadrez usado para o exemplo da função de avaliação.

Então, foi implementado o Algoritmo 3 com o intuito de aplicar a busca Minimax com poda alfa-beta aplicados ao xadrez e utilizando a função de avaliação proposta baseada em heurísticas.

3.3 Abordagem Utilizando Redes Neurais

3.3.1 Objetivo Principal do Método

Este método foi baseado na abordagem proposta no artigo [13], no qual observa que os principais avanços e melhorias na qualidade de jogo estão relacionados a refinamentos da função de avaliação. Assim, ainda será usado o algoritmo Minimax com poda alfa-beta, entretanto, a função de avaliação será desenvolvida através de redes neurais, estas sendo treinadas a comparar duas posições e predizer qual delas tem maior chance de encaminhar para um resultado final vitorioso. Para isso, foi utilizada uma Rede de Crenças Profundas para extrair características de uma posição e uma variação de Rede Siamesa que, ao invés de predizer similaridade, como normalmente utilizada, foi treinada para predizer qual das duas entradas tem maior chance de vitória.

3.3.2 Coleção de Dados

Para o treinamento de ambas as redes, é necessário um grande número de posições de um tabuleiro de xadrez, com o rótulo de se a posição faz parte de um jogo ganho ou perdido em relação a um dos jogadores, no caso, o jogador de peças brancas. A ideia inicial foi utilizar as partidas realizadas na plataforma *chess.com*, por ser a maior plataforma de jogos online

Algoritmo 3: Minimax com poda alfa-beta.

```
Minimax(profundidade, maximizador, alfa, beta, tabuleiro)
    if profundidade = 0 then
        valor = f(tabuleiro)
        return valor
    end
    else if maximizador = True then
        melhor_valor = −∞
        for tabuleiro filho do tabuleiro atual do
            valor = Minimax(profundidade - 1, False, alfa, beta, tabuleiro)
            melhor_valor = max(melhor_valor, valor)
            alfa = max(alfa, melhor_valor)
            if beta ≤ alfa then
                break
            end
        end
        return melhor_valor e lance que levou ao tabuleiro filho de melhor valor
    end
    else
        melhor_valor = +∞
        for tabuleiro filho do tabuleiro atual do
            valor = Minimax(profundidade - 1, True, alfa, beta, tabuleiro)
            melhor_valor = min(melhor_valor, valor)
            beta = min(beta, melhor_valor)
            if beta ≤ alfa then
                break
            end
        end
        return melhor_valor e lance que levou ao tabuleiro filho de melhor valor
    end
```

de xadrez, mas foram encontradas algumas dificuldades em acessar uma grande quantidade de jogos. As partidas são apenas acessáveis através de uma API REST, a qual é ideal para acessar algumas partidas de um determinado jogador, contudo, para acessar o número necessário de partidas desejadas, divididas entre diversos jogadores, a API do *chess.com* não se demonstrou prática e eficiente.

Outra grande plataforma de xadrez online, o *Lichess*, também possui uma base de dados [48] aberta, entretanto, nesta estão disponíveis todos os jogos realizados na plataforma para *download* direto, divididos em períodos de um mês desde 2013. Atualmente, possui cerca de 3,8 bilhões de partidas acessáveis por arquivos compactados no formato PGN. Devido a essa praticidade, esta foi a opção selecionada para coletar os jogos de xadrez, em especial, os realizados em janeiro deste ano, 2022, no qual possui 102 milhões 110 mil e 423 partidas.

Além dos jogos realizados por pessoas coletados do *Lichess*, também foi usada a base de dados do grupo *Computer Chess Rating Lists* (CCRL) [49]. Grupo este, que consiste em realizar partidas entre as diversas *engines* disponíveis para uso público e classificá-las por ordem de vitórias. As partidas realizadas estão disponíveis para *download* e, atualmente, possui 1 milhão 492 mil e 283 jogos. Esta opção foi considerada para efeitos comparativos entre o treinamento do método realizado em partidas entre pessoas e entre *engines*, pois esta última espera-se que tenha menos erros cometidos e não tenha abandonos, podendo, assim, ter mais posições de cheque-mate.

Em seguida, partidas foram filtradas dos dados obtidos que poderiam apresentar resultados melhores para o treinamento. De ambas bases de dados, foram removidas partidas que terminaram empatadas ou terminaram porque o tempo de alguns dos jogadores se encerrou. Removeu-se também jogos em que a média de *rating* dos jogadores fosse menor que 2400 e partidas do formato *bullet* (1 minuto para cada jogador). No final, resultaram 1 milhão 160 mil e 552 partidas da plataforma *Lichess* e 607 mil e 825 do grupo CCRL.

Por fim, para utilização nos treinamentos das redes neurais, foi utilizado apenas um pequeno número de posições de cada partida e não a partida inteira. De preferência, posições que não fossem seguidas de uma captura, pois poderia resultar em uma partida falsamente desequilibrada, pois num instante um jogador tem uma vantagem de peças e existe uma grande probabilidade de ser recuperada logo em seguida numa troca de peças. Assim, a vantagem material naquele instante não representa verdadeiramente uma vantagem ao longo do jogo e, como descrito no artigo [13], apresenta prejuízos ao treinamento. Além disso, também não foram selecionadas posições que estão logo no início do jogo, por se repetirem sem um resultado claro associado à ela. Por exemplo, a abertura italiana aparecerá diversas vezes na base de dados e cerca de metade delas estará rotulada como vitória e a outra metade como derrota. Como a abertura depende, principalmente, do estilo pessoal do jogador e possui baixa influência no resultado final, sua inclusão apresenta prejuízos à análise aqui pretendida.

Selecionando 3 posições de cada partida do *Lichess* e 5 de cada partida do grupo CCRL, foram obtidas 3 milhões 481 mil 656 posições da primeira plataforma e 3 milhões 39 mil 126 posições da segunda (Tabela 1).

Tabela 1: Dados coletados.

Fonte	Partidas Obtidas	Partidas Filtradas	Posições Extraídas
Lichess	102.110.423	1.160.552	3.481.656
CCRL	1.429.283	607.825	3.039.125

3.3.3 Vetorização de uma Posição de Xadrez

Antes da implementação das redes neurais, faz-se necessário estabelecer um método para vetorizar cada posição obtida para ser possível usá-las como entradas das redes. Existem alguns métodos para vetorizar um determinado estado do jogo de xadrez [50]. Como a arquitetura das redes neurais escolhidas para implementação foi a mesma utilizada para

o desenvolvimento da *engine* nomeada *DeepChess* no artigo [13], escolheu-se também o método descrito no mesmo artigo para vetorizar uma posição, este sendo uma representação do tabuleiro e seu estado em uma sequência de dígitos binários, amplamente conhecida como *bitboard*.

Esta representação consiste em descrever o estado atual do jogo, semelhante à notação FEN, entretanto, utilizando matrizes binárias para cada conjunto composto de peça e cor. Desta forma, cada peça será associada a uma matriz binária 8×8, na qual cada bit representa uma casa do tabuleiro e possui valor 1, caso esteja ocupado pela peça e valor 0, caso esteja vazio. Considerando que existem 6 peças para cada cor, assim, a representação *bitboard* consiste em uma matriz binária de 8×8×12, acrescidos de outros 5 bits, 4 para manter o registro se o roque curto ou longo de cada cor está disponível e outro para retratar qual jogador realizará o lance no turno. Assim, todas as posições obtidas foram vetorizadas em um vetor *NumPy* [51] composto de 773 bits para a representação *bitboard* e mais um para rotular se a posição faz parte de um jogo em que a cor branca venceu ou perdeu, totalizando, por fim, um vetor binário de tamanho 774.

3.3.4 Implementação da Rede de Crenças Profundas

Para a implementação das redes neurais deste projeto, foi utilizada a biblioteca *PyTorch* [52], além do artigo [53], no qual foi desenvolvida sua própria versão da *engine DeepChess* utilizando o *framework TensorFlow* [54] e, assim, auxiliou nas decisões de hiperparâmetros.

A primeira parte da arquitetura da *engine*, a Rede de Crenças Profundas (DBN), foi implementada e realizado um pré-treinamento separadamente. Seguindo o conceito da arquitetura descrita na Subseção 2.3.1, foi implementada uma DBN de 5 camadas (Figura 12) consistindo de uma camada de entrada para um tensor de tamanho 773 (representação *bitboard*), 3 camadas ocultas de tamanhos 600, 400 e 200, por fim, a camada de saída de tamanho 100. Assim, obteve-se como entrada o estado do jogo, em um determinado instante, no formato *bitboard*, o que irá reduzir a dimensionalidade de 773 elementos para 100, descrevendo, assim, a mesma posição, entretanto, com uma codificação de apenas 100 elementos. Desta forma, a Rede de Crenças Profundas está sendo utilizada para extrair as principais características de uma dada posição de um jogo de xadrez.

Os melhores resultados obtidos foram utilizando a função de ativação Unidade Linear Retificada (ReLU) entre as camadas intermediárias, a função Sigmoid para normalização da saída e, principalmente, normalização em *batch* [55] entre as ativações das camadas intermediárias.

3.3.5 Treinamento da Rede de Crenças Profundas

Esta primeira etapa de treinamento consiste num treinamento não supervisionado apenas com o modelo da DBN, antes de utilizá-la no modelo de Rede Siamesa. Como consiste em um pré treinamento não supervisionado, para aprender a extrair características a partir de um determinado *bitboard*, neste momento apenas foram utilizados os dados obtidos da plataforma *Lichess*, por conter maior número de dados com grande variabilidade entre eles, já que foram extraídos menos posições de mais partidas.

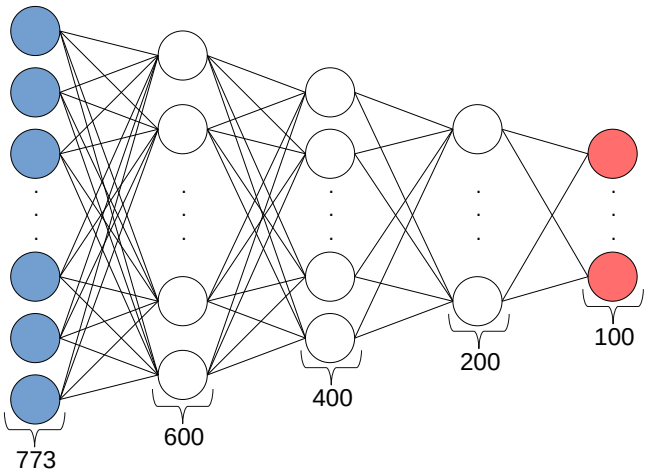


Figura 12: Arquitetura da Rede de Crenças Profundas da *DeepChess*. Inspirado na Figura 1 do artigo [13].

Os dados foram divididos entre 98% de treinamento e 2% de teste, sendo assim, obteve-se 3 milhões 412 mil e 22 dados de treinamento e 69 mil e 634 de teste. Para a função de erro, foi escolhida a função de Erro Quadrado Médio e a taxa de aprendizado 0,005, sendo esta multiplicada por 0,98 a cada época. Por fim, o treinamento foi executado durante 200 épocas, com 64 *batches*, em GPU, a placa de vídeo sendo uma do modelo RTX 3070.

O treinamento consistiu em, primeiramente, codificar as posições de xadrez recebidas em um tensor de tamanho 100 e, após isso, realizar o caminho inverso para reconstruir a representação *bitboard* a partir da codificação. O cálculo do erro é realizado entre a posição original e a reconstruída. Na Figura 13, pode-se notar o decaimento da função de cálculo do erro durante o treinamento. Ao fim das 200 épocas, predições foram realizadas sobre as 69 mil e 634 posições de teste, as quais 65,15% reconstruíram todos os 773 bits corretamente e a média de bits incorretos entre o dado original e o reconstruído foi de 1,69.

3.3.6 Implementação da Rede Neural Siamesa

Esta segunda parte da implementação contém a arquitetura completa da *engine Deepchess*. A Rede de Crenças Profundas, treinada anteriormente, constituiu parte da arquitetura da Rede Neural Siamesa desenvolvida, a saída codificada da mesma foi totalmente conectada a uma nova etapa de treinamento. Como descrito na Subseção 2.3.2, a nova etapa mencionada, normalmente, tem como objetivo calcular a similaridade entre ambas as entradas, contudo, nesse modelo para aprender a jogar xadrez, essa etapa foi modificada para predizer qual das duas entradas tem maior probabilidade de encaminhar para uma vitória.

A arquitetura dessa nova inclusão ao modelo consiste em 4 novas camadas e seus tamanhos são, respectivamente, 400, 200, 100 e 2 (Figura 14). A primeira camada recebe as saídas de ambas as DBNs, que possuem pesos compartilhados, e junto das próximas camadas, calcula qual delas possui a maior probabilidade de vencer a partida. Por fim, a

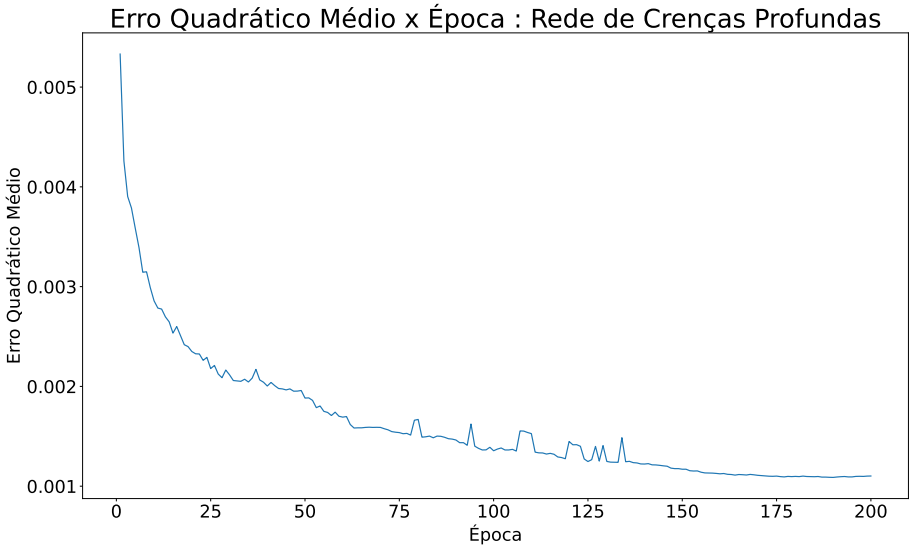


Figura 13: Progresso do erro quadrático médio a cada época do treinamento da DBN.

última camada resulta em um tensor de tamanho 2, com o primeiro elemento contendo a probabilidade de vitória da primeira posição e o segundo elemento com a probabilidade da segunda posição.

Demais parâmetros se mantiveram da implementação da Rede de Crenças Profundas, mantendo-se a normalização em *batches* entre as camadas, pois se mostrou essencial na primeira etapa do treinamento, a função de ativação de camadas intermediárias segue sendo a ReLU e a função de normalização ao fim, a Sigmoid.

3.3.7 Treinamento da Rede Neural Siamesa

Esta segunda etapa de treinamento é a principal, pois nesse momento, os pesos de toda arquitetura foram modificados, inclusive da rede pré-treinada, entretanto, mantêm-se os pesos compartilhados. O treinamento consiste em ser supervisionado, ou seja, foram utilizados os rótulos de vitória ou derrota para calcular o erro da predição. Para teste, foram separados 100 mil jogos dos dados, sendo 50 mil de jogos ganhos e 50 mil de jogos perdidos, contudo, os dados de treinamento não foram totalmente utilizados a cada época.

Os dados de entrada para o treinamento consistem em uma dupla de uma posição de uma partida ganha e uma posição de uma partida perdida, na qual a ordem é relevante, ou seja, dadas duas posições A e B, a dupla (A, B) se difere da (B, A). Desse modo, ambas as bases de dados coletadas possuem a quantidade de duplas possíveis na ordem de 10^{12}, dado que os jogos obtidos da plataforma *Lichess* possuem 1.847.382 vitórias e 1.634.274 derrotas e os jogos do grupo CCRL possuem 1.820.982 vitórias e 1.218.143 derrotas. Por esse conjunto de possibilidades possuir um tamanho grande, é selecionado apenas 1 milhão de duplas a cada época.

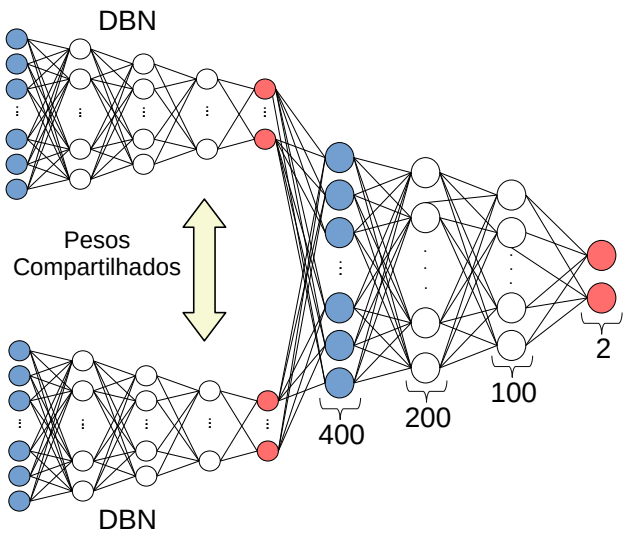


Figura 14: Arquitetura da Rede Neural Siamesa da *DeepChess*. Inspirado na Figura 1 do artigo [13].

Ademais, o treinamento seguiu um método semelhante ao anterior, utilizando 64 *batches* durante 200 épocas e na mesma GPU, modificando a taxa de aprendizado para iniciar em 0,01 e ser multiplicada por 0,99 a cada época e também foi trocada a função do cálculo de erro para Entropia Cruzada Binária.

Primeiramente, foi realizado o treinamento com os dados obtidos do *Lichess*. Na Figura 15, pode-se observar que, apesar da curva serrilhada ser esperada, já que cada época possui um conjunto de dados novos, o valor do erro se manteve alto, iniciando próximo a 0,5 e estabilizando em torno de 0,2. Ao realizar as predições sobre o conjunto de teste, obteve-se uma acurácia de 79,2%.

Esperando que o erro alto e a acurácia relativamente baixa sejam devido ao fato de a base de dados obtida ser composta por jogos online de pessoas, que podem ter erros e uma variabilidade grande, principalmente em formatos que possuem pouco tempo para cada jogador, foi realizado um novo treinamento na base de dados do grupo CCRL, composto apenas por jogos de *engines*. Foram mantidos os mesmos parâmetros e, na Figura 16, pode-se notar que o erro realmente diminuiu, iniciando próximo a 0,3 e estabilizando próximo 0,1. Ao realizar as predições no conjunto de teste, a acurácia aumentou para 92.8%.

3.3.8 Modificação do Algoritmo Minimax com Poda Alfa-Beta

Para esse método de comparação de posições ser utilizado como função de avaliação do algoritmo Minimax com poda alfa-beta, o algoritmo precisa ser alterado. Como descrito na Seção 2.1, a função de avaliação baseada em heurísticas, normalmente, avalia apenas o estado atual do tabuleiro, entretanto, o método desenvolvido necessita de duas posições para serem comparadas e, por fim, predizer qual é melhor. Para isso, optou-se por também

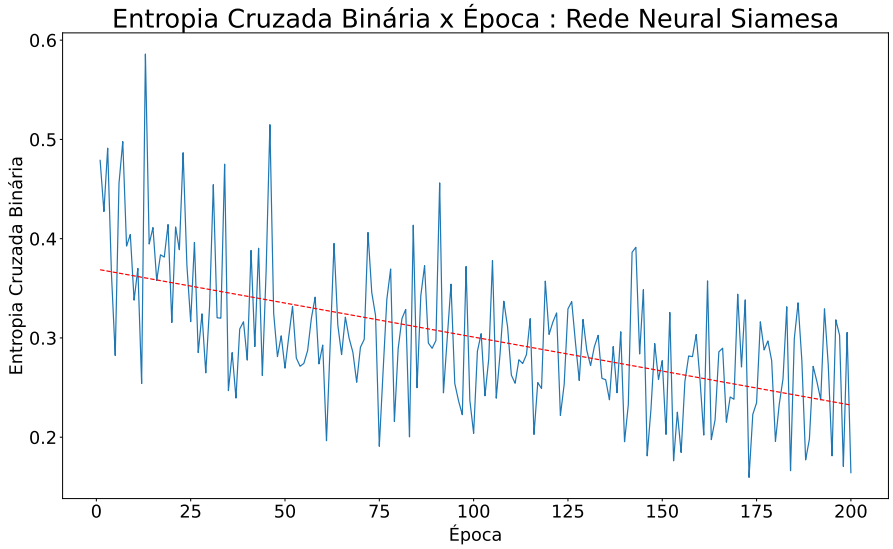


Figura 15: Progresso da Entropia Cruzada Binária a cada época do treinamento da Rede Neural Siamesa.

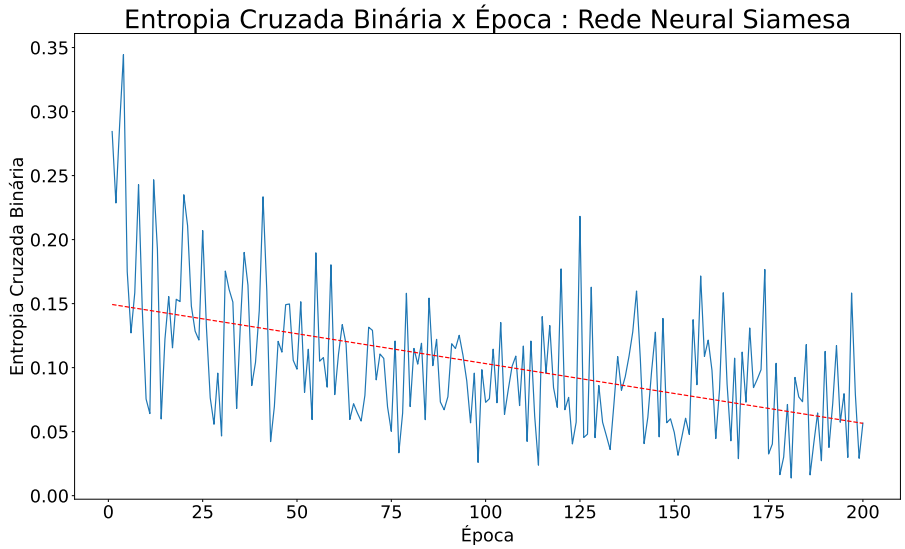


Figura 16: Progresso da Entropia Cruzada Binária a cada época do treinamento da Rede Neural Siamesa.

passar como parâmetro a posição pai, posição do tabuleiro antes do lance ser realizado, assim, no nó folha a função de avaliação realizará a comparação entre a posição pai e a posição filha e retornará a probabilidade da posição filha ser melhor. Por fim, a posição que

possuir o melhor valor em relação ao pai será a escolhida para ser jogada (Algoritmo 4).

Algoritmo 4: Minimax com poda alfa-beta comparando posições no nó folha.

```
Minimax(profundidade, maximizador, alfa, beta, posição_filha, posição_pai)
    if profundidade = 0 then
        valor = compara_posições(posição_filha, posição_pai)
        return valor
    end
    else if maximizador = True then
        melhor_valor = −∞
        for posição_filha da posição atual do
            valor = Minimax(profundidade - 1, False, alfa, beta, posição_filha, posição)
            melhor_valor = max(melhor_valor, valor)
            alfa = max(alfa, melhor_valor)
            if beta ≤ alfa then
                break
            end
        end
        return melhor_valor e lance que levou a posição filha de melhor valor
    end
    else
        melhor_valor = +∞
        for posição_filha da posição atual do
            valor = Minimax(profundidade - 1, True, alfa, beta, posição_filha, posição)
            melhor_valor = min(melhor_valor, valor)
            beta = min(beta, melhor_valor)
            if beta ≤ alfa then
                break
            end
        end
        return melhor_valor e lance que levou a posição filha de melhor valor
    end
```

4 Resultados

Através da interface gráfica, foi possível verificar um avanço da qualidade do jogo durante o refinamento da função de avaliação até a demonstrada tanto na baseada em heurísticas quanto na melhoria do treinamento das redes neurais, entretanto, para uma análise qualitativa e quantitativa profunda do desempenho de uma *engine* de xadrez, seria necessário realizar diversas partidas contra adversários de diversas classificações de *rating* para, por fim, estimar o *rating* da *engine* desenvolvida. Dado ao nível de complexidade desse método de avaliação, para este projeto, o método escolhido para análise dos resultados se baseia em problemas de xadrez que a ferramenta foi capaz de solucionar.

Os problemas utilizados estão disponíveis na base de dados da plataforma de jogos online *Lichess* [56]. Contendo mais de três milhões de problemas propostos, foram filtrados cinco mil deles, dividindo entre nove temas que consideram as três fases do jogo (abertura, meio-jogo e final), cheque-mate e quantidade de lances necessários para a solução, sendo, médio para cinco lances, abaixo desse valor é curto e acima longo. Além disso, realizou-se também, uma divisão em 5 intervalos de *rating*, com cada uma contendo mil problemas, os intervalos são: [0,1200], (1200,1800], (1800, 2200], (2200, 2500], (2500, ∞).

A Tabela 2 contém o total de problemas divididos entre os noves temas e a Figura 17 apresenta a proporção de problemas resolvidos tanto no total quanto em cada intervalo de *rating* para a abordagem clássica.

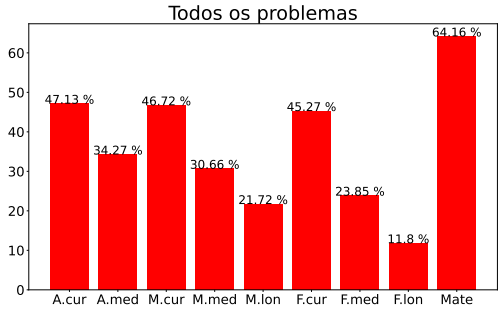
Tabela 2: Tabela de resultados totais da abordagem Minimax clássica.

Temas	Problemas	Solucionados
Abertura curta (A.cur)	522	246
Abertura média (A.med)	531	182
Meio-jogo curto (M.cur)	640	299
Meio-jogo médio (M.med)	649	199
Meio-jogo longo (M.lon)	617	134
Final curto (F.cur)	592	268
Final médio (F.med)	566	135
Final longo (F.lon)	551	65
Cheque-mate (Mate)	332	213
Total	5000	1741

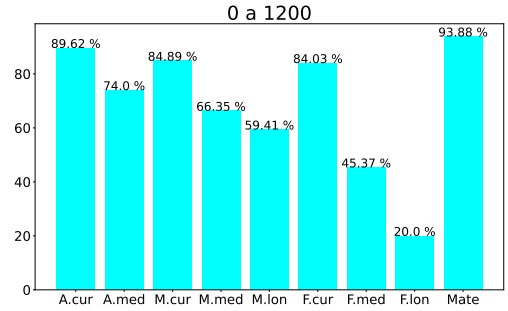
Utilizando os mesmos problemas coletados do *Lichess* para avaliar o desempenho dos modelos desenvolvidos. Na Tabela 3, pode-se observar todos os problemas resolvidos através do treinamento realizado nos dados obtidos do *Lichess* e, na Figura 18, a proporção de solução para cada tema dividido por *rating*. Na Tabela 4 e Figura 19, temos os mesmos resultados para o treinamento dos dados do CCRL.

Na Figura 20, pode-se observar a quantidade de problemas resolvidos em três profundidades diferentes. Todas melhoram ao aprofundarem suas buscas, entretanto, a abordagem clássica mantém elevada vantagem. Assim é válido dizer que uma função de avaliação baseada em heurísticas simples, a partir de conhecimentos básicos de conceitos gerais do xadrez, sendo eles, vantagem material, maior mobilidade das peças, controle das casas centrais e proteção do rei, já foram suficientes para alcançar bons resultados e esses conhecimentos aparentemente simples não são facilmente aprendidos pela máquina. Contudo, é possível realizar observações interessantes a partir desses resultados.

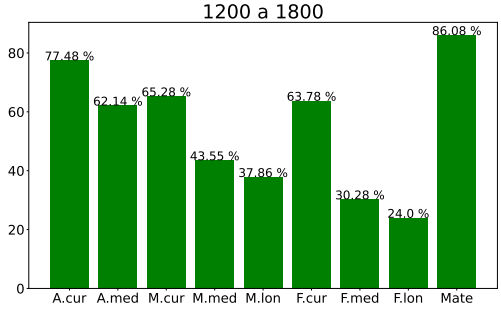
Durante o treinamento, o modelo treinado com a base de dados do *Lichess* pareceu apresentar resultados inferiores ao modelo treinado com dados do CCRL, entretanto, esse modelo somente se destacou utilizando profundidade 1, enquanto que, em outros, o modelo do *Lichess* superou e, inclusive nos jogos um contra o outro, obteve uma vitória (Figura 21b). A maior acurácia nos treinamentos, provavelmente, está relacionada ao fato dos jogos do grupo CCRL possuírem muitas partidas das mesmas *engines* e por serem programadas,



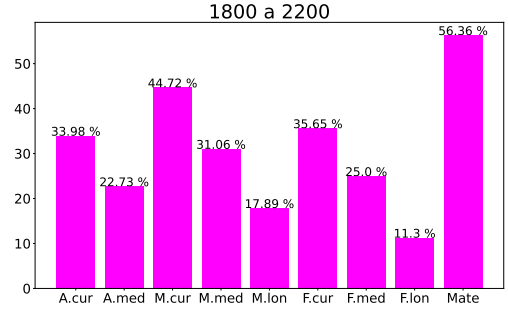
(a) Resultado total dos 5 mil problemas.



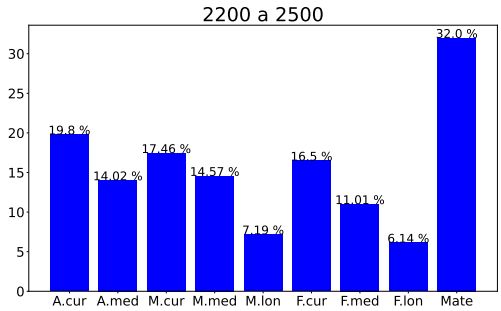
(b) Resultado nos problemas de até 1200 de *rating*.



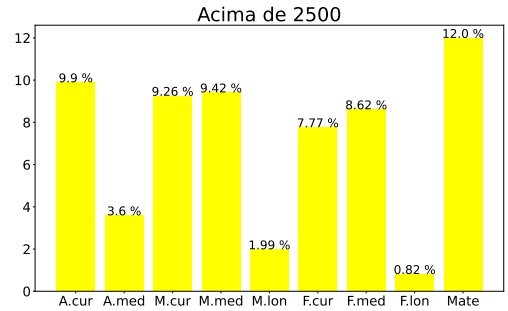
(c) Resultado nos problemas com intervalo de *rating* de 1200 até 1800.



(d) Resultado nos problemas com intervalo de *rating* de 1800 até 2200.



(e) Resultado nos problemas com intervalo de *rating* de 2200 até 2500.



(f) Resultado nos problemas acima de 2500 de *rating*.

Figura 17: Problemas solucionados utilizando a abordagem clássica.

tenderem a jogar com o mesmo estilo e repetir muitas posições. Dessa forma, a predição durante o treinamento possui melhores resultados, entretanto, mesmo com menor acurácia, o modelo treinado com *Lichess* aproveitou da maior variabilidade para apresentar resultados melhores em problemas diversificados em temas e níveis de dificuldade.

As principais dificuldades dos modelos de aprendizado de máquina em relação à abor-

Tabela 3: Tabela de resultados totais do modelo treinado com dados do *Lichess*.

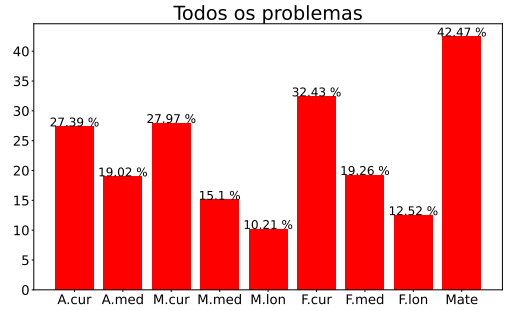
Temas	Problemas	Solucionados
Abertura curta (A.cur)	522	143
Abertura média (A.med)	531	101
Meio-jogo curto (M.cur)	640	179
Meio-jogo médio (M.med)	649	98
Meio-jogo longo (M.lon)	617	63
Final curto (F.cur)	592	192
Final médio (F.med)	566	109
Final longo (F.lon)	551	69
Cheque-mate (Mate)	332	141
Total	5000	1095

Tabela 4: Tabela de resultados totais do modelo treinado com dados do grupo CCRL.

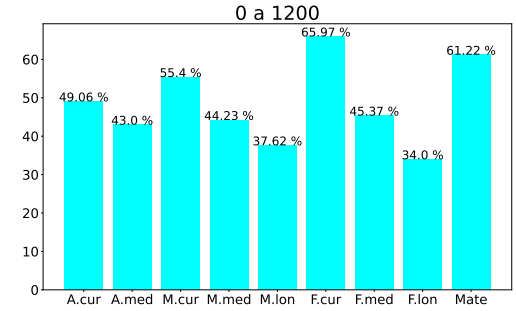
Temas	Problemas	Solucionados
Abertura curta (A.cur)	522	108
Abertura média (A.med)	531	71
Meio-jogo curto (M.cur)	640	164
Meio-jogo médio (M.med)	649	103
Meio-jogo longo (M.lon)	617	59
Final curto (F.cur)	592	198
Final médio (F.med)	566	98
Final longo (F.lon)	551	61
Cheque-mate (Mate)	332	108
Total	5000	970

dagem clássica foram a abertura, em especial, o CCRL, e cheque-mate. Enquanto o modelo clássico obteve um equilíbrio entre abertura, meio-jogo e final, os outros se destacaram em meio-jogo e final, especialmente final, sendo o modelo treinado com o *Lichess* que mais resolveu problemas com tema de finais longos, no geral, e o modelo treinado com CCRL se destacou em finais de *rating* elevado. Assim, pode-se perceber os contextos nos quais os modelos de aprendizado de máquina se inserem melhor e como a base de dados afetou os resultados finais.

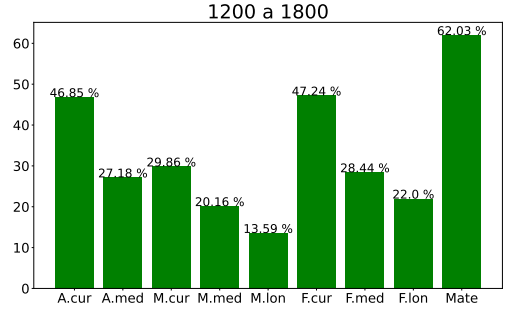
A Figura 21 apresenta os resultados finais do jogos entre os modelos desenvolvidos indicando os jogadores e qual cor venceu a partida, os dois empates obtidos (Figura 21a e Figura 21d) foram por repetição. Apesar de estarem numa posição vantajosa, o modelo treinado com dados CCRL no primeiro caso e o modelo clássico no segundo, não conseguem aplicar o cheque-mate. Cada modelo realizou 4 jogos, considerando 1 ponto para vitória, 0,5 para empate e 0 para derrota, o saldo final foi o seguinte: modelo clássico com 3,5 pontos, modelo treinado com a plataforma *Lichess* 2 pontos e modelo treinado com dados do grupo CCRL 0,5 ponto.



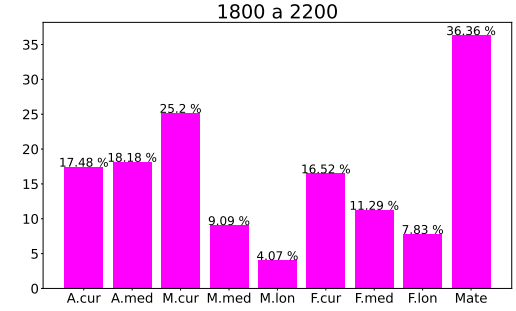
(a) Resultado total dos 5 mil problemas.



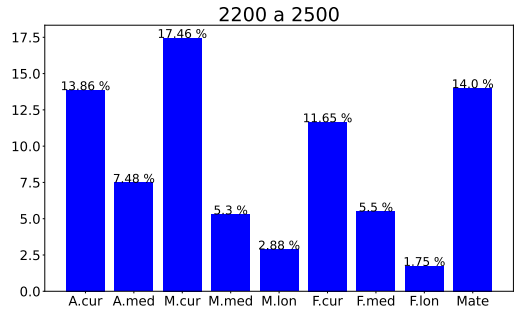
(b) Resultado nos problemas de até 1200 de *rating*.



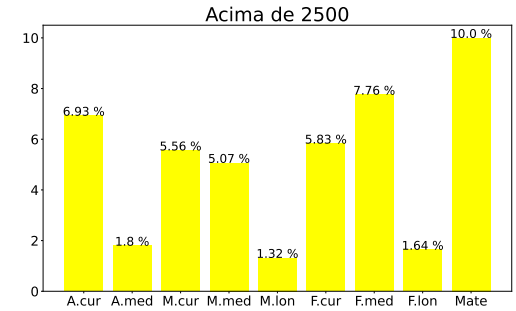
(c) Resultado nos problemas com intervalo de *rating* de 1200 até 1800.



(d) Resultado nos problemas com intervalo de *rating* de 1800 até 2200.



(e) Resultado nos problemas com intervalo de *rating* de 2200 até 2500.

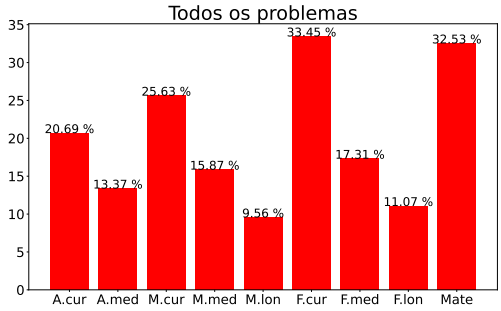


(f) Resultado nos problemas acima de 2500 de *rating*.

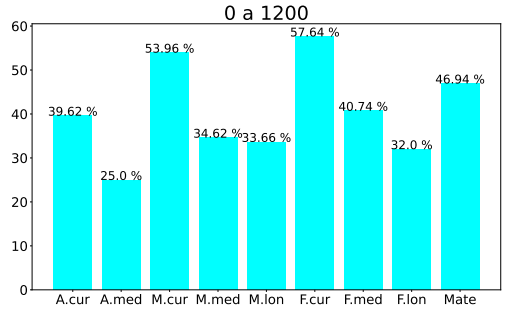
Figura 18: Problemas solucionados utilizando o modelo treinado com dados do *Lichess*.

5 Conclusões e Trabalhos Futuros

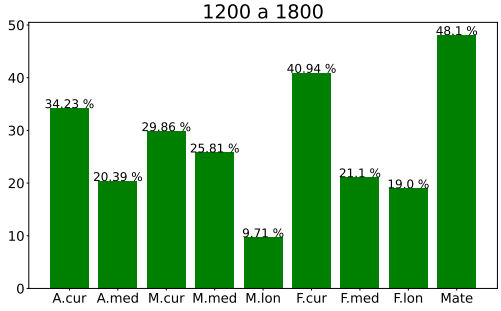
Conclui-se que métodos de aprendizado de máquina no xadrez possuem diversas dificuldades. Estudos que apresentam resultados superiores são recentes, entretanto, é possível observar cenários que possuem vantagens, por exemplo, em final de jogo em que os conceitos básicos da maioria das heurísticas, por vezes, se perdem, métodos com redes neurais podem



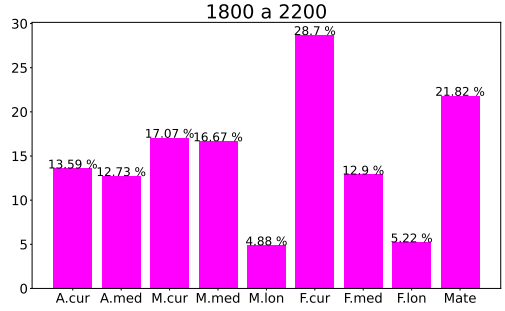
(a) Resultado total dos 5 mil problemas.



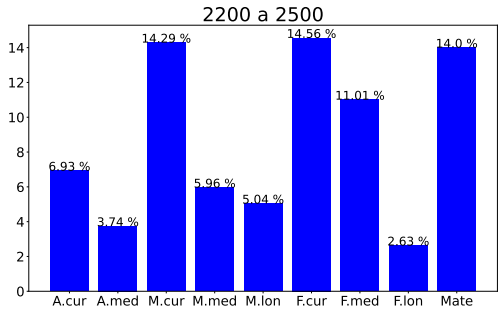
(b) Resultado nos problemas de até 1200 de *rating*.



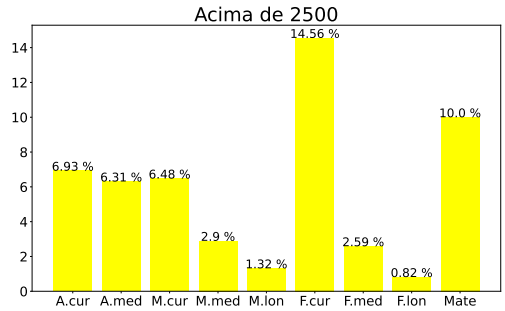
(c) Resultado nos problemas com intervalo de *rating* de 1200 até 1800.



(d) Resultado nos problemas com intervalo de *rating* de 1800 até 2200.



(e) Resultado nos problemas com intervalo de *rating* de 2200 até 2500.



(f) Resultado nos problemas acima de 2500 de *rating*.

Figura 19: Problemas solucionados utilizando o modelo treinado com dados do grupo CCRL.

ajudar. Assim, pode-se compor ambos os métodos para maior vantagem, por exemplo, utilizar modelos baseados em aprendizado de máquina apenas em finais de jogo e em aberturas utilizar outros métodos, já que esse se mostrou ineficaz.

Além de notar que a qualidade dos jogos em bancos de dados diferentes influenciou o resultado final, sendo assim, poderia ser melhorado refinando os filtros de partidas utilizado e

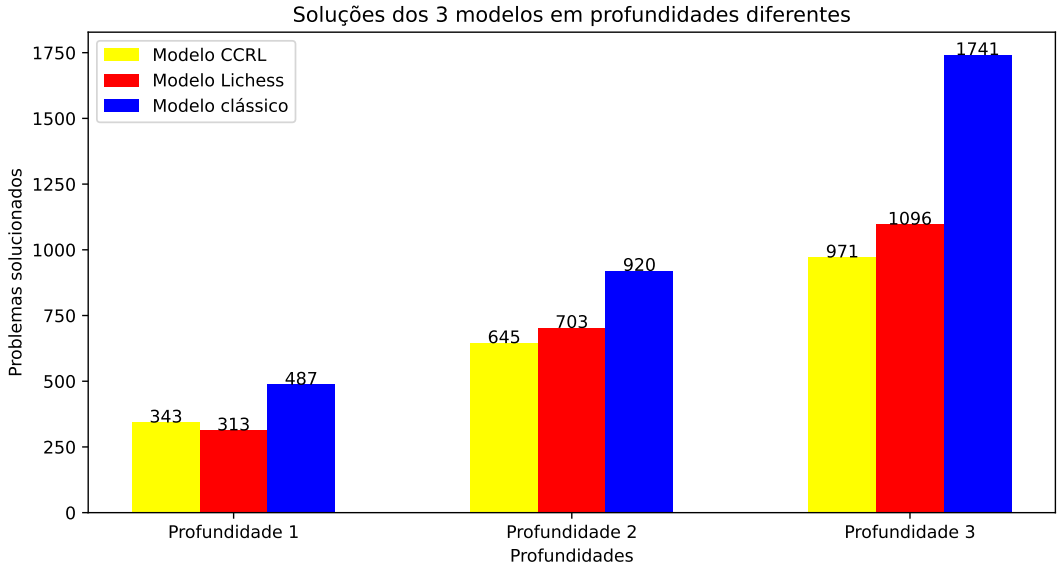


Figura 20: Desempenho das *engines* em cada profundidade.

investigando em mais dados, ou então, excluído o uso de base de dados e utilizar aprendizado por reforço, assim, a qualidade não se limitaria à qualidade dos jogos coletados para o treinamento.

Com este projeto, foi possível introduzir conceitos de aprendizado de máquina no xadrez, entender suas vantagens e desvantagens, para assim, outros trabalhos conseguirem partir de uma base concreta que auxilie a avançar os estudos.

Referências

[1] S. Russel and P. Norvig, *Inteligência Artificial, Tradução da Segunda Edição*, p. 16. Rio de Janeiro, RJ: Elsevier: Campus, 2004.

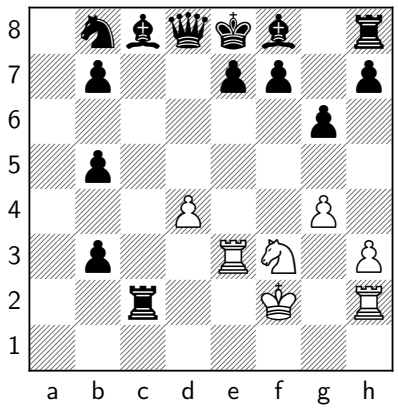
[2] C. E. Shannon, "Xxii. programming a computer for playing chess," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 41, no. 314, pp. 256–275, 1950.

[3] S. H. Fuller, J. G. Gaschnig, and J. Gillogly, *Analysis of the alpha-beta pruning algorithm*. Department of Computer Science, Carnegie-Mellon University, 1973.

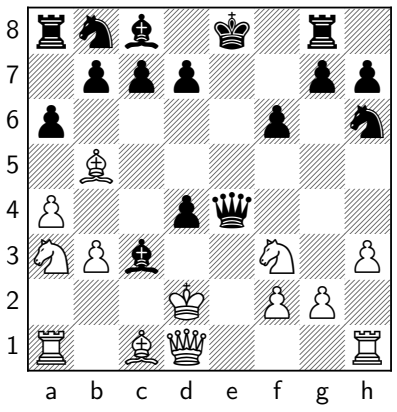
[4] B. Keen, "A history of computer chess," tech. rep., University of Bordeaux, 2009.

[5] M. Newborn, *Kasparov versus Deep Blue: Computer chess comes of age*. Springer Science & Business Media, 2012.

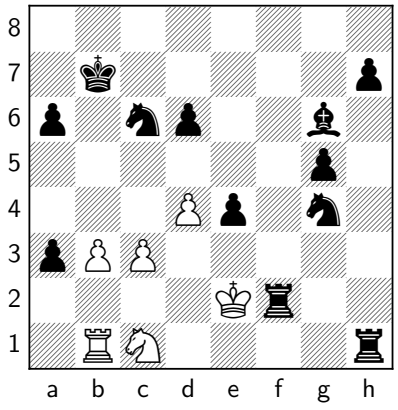
[6] P. W. Frey, "Second appendix chess 4.5 and chess 4.6: Competition in 1977 and 1978," in *Chess Skill in Man and Machine*, pp. 248–256, Springer, 1983.



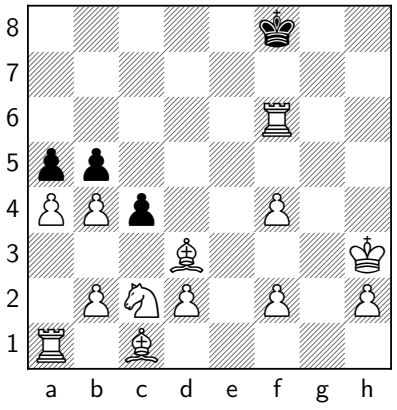
(a) Branca: *Lichess*; Preta: CCRL; Empate.



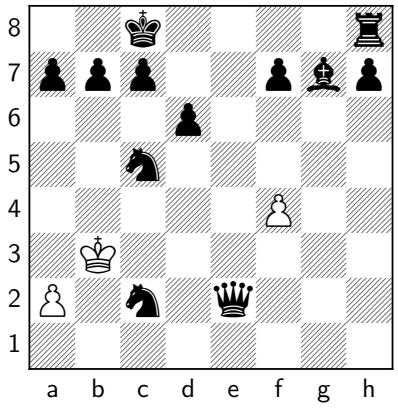
(b) Branca: CCRL; Preta: Lichess; Pretas.



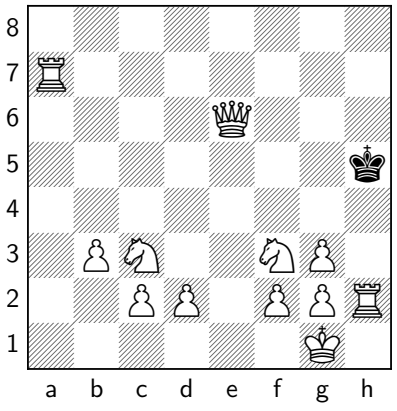
(c) Branca: Lichess; Preta: Classico; Pretas.



(d) Branca: Classico; Preta: Lichess; Empate.



(e) Branca: CCRL; Preta: Clássico; Pretas.



(f) Branca: Clássico; Preta: CCRL; Brancas.

Figura 21: Resultados finais dos jogos entre os modelos desenvolvidos.

[7] F.-H. Hsu, *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2002.

[8] G. Kendall and G. Whitwell, "An evolutionary approach for the tuning of a chess evaluation function using population dynamics," in *Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, vol. 2, pp. 995–1002, IEEE, 2001.

[9] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

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

[11] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[12] M. C. Fu, "AlphaGo and Monte Carlo tree search: the simulation optimization perspective," in *Winter Simulation Conference (WSC)*, pp. 659–670, IEEE, 2016.

[13] O. E. David, N. S. Netanyahu, and L. Wolf, "Deepchess: End-to-end deep neural network for automatic learning in chess," in *International Conference on Artificial Neural Networks*, pp. 88–96, Springer, 2016.

[14] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.

[15] S. B. J, "A friendly introduction to siamese networks," 2020. Towards Data Science. Disponível em: `https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942`. Acesso em: 25 de set. de 2022.

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

[17] "Stockfish." Disponível em: `https://stockfishchess.org/`. Acesso em: 25 de set. de 2022.

[18] A. Champion, "Dissecting Stockfish Part 2: In-Depth Look at a chess engine," 2021. Towards Data Science. Disponível em: `https://towardsdatascience.com/dissecting-stockfish-part-2-in-depth-look-at-a-chess-engine-2643cdc35c9a`. Acesso em: 24 de set. de 2022.

[19] A. E. Elo, *The Rating of Chessplayers, Past and Present*. Arco Pub., 1978.

[20] P. Pinto, "Introducing the min-max algorithm," *Submited to the AI Depot article contest*, pp. 1–10, 2002.

[21] D. W. North, "A tutorial introduction to decision theory," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 3, pp. 200–210, 1968.

[22] M. J. Osborne, *An introduction to game theory*, vol. 3. Oxford University Press, 2004.

[23] D. Levy and M. Newborn, "How computers play chess," in *All About Chess and Computers*, pp. 24–39, Springer, 1982.

[24] J. Clune, "Heuristic evaluation functions for general game playing," in *Association for the Advancement of Artificial Intelligence*, vol. 7, pp. 1134–1139, 2007.

[25] R. Nasa, R. Didwania, S. Maji, and V. Kumar, "Alpha-Beta Pruning in Mini-Max Algorithm: An Optimized Approach for a Connect-4 Game," *Int. Res. J. Eng. Technol*, pp. 1637–1641, 2018.

[26] M. S. Campbell and T. A. Marsland, "A comparison of minimax tree search algorithms," *Artificial Intelligence*, vol. 20, no. 4, pp. 347–367, 1983.

[27] C. Felstiner, "Alpha-beta pruning," 2019.

[28] S. Mandadi, B. Tejashwini, and S. Vijayakumar, "Implementation of sequential and parallel alpha-beta pruning algorithm," *International Journal of Innovations in Engineering Research and Technology*, vol. 7, no. 08, pp. 98–104, 2020.

[29] Z.-H. Zhou, *Machine Learning*. Springer Nature, 2021.

[30] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[31] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[32] R. Rojas, "The backpropagation algorithm," in *Neural Networks*, pp. 149–182, Springer, 1996.

[33] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Annals of Data Science*, vol. 9, no. 2, pp. 187–212, 2022.

[34] S. Dara and P. Tumma, "Feature extraction by using deep learning: A survey," in *Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 1795–1801, IEEE, 2018.

[35] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991*, 2020.

[36] Y. Hua, J. Guo, and H. Zhao, "Deep belief networks and deep learning," in *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, pp. 1–4, IEEE, 2015.

[37] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[38] P. Hamel and D. Eck, "Learning features from music audio with deep belief networks.," in *ISMIR*, vol. 10, pp. 339–344, Utrecht, The Netherlands, 2010.

[39] A. Fischer and C. Igel, "An introduction to restricted boltzmann machines," in *Iberoamerican Congress on Pattern Recognition*, pp. 14–36, Springer, 2012.

[40] D. Chicco, "Siamese neural networks: An overview," *Artificial Neural Networks*, pp. 73–94, 2021.

[41] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Annual Meeting of the Cognitive Science Society*, vol. 33, 2011.

[42] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML Deep Learning Workshop*, vol. 2, p. 0, Lille, 2015.

[43] M. Arora, "Deep learning for computer chess (part 1)," tech. rep., Nanyang Technological University, Singapore, 2022.

[44] "python-chess." Disponível em: `https://python-chess.readthedocs.io/en/latest/`. Acesso em: 28 de nov. de 2022.

[45] "pygame." Disponível em: `https://www.pygame.org/docs/`. Acesso em: 15 de nov. de 2022.

[46] "Simple directmedia layer 2.0." Disponível em: `https://www.pygame.org/docs/`. Acesso em: 28 de nov. de 2022.

[47] W. B. Putra and L. Heryawan, "Applying alpha-beta algorithm in a chess engine," *Jurnal Teknosains*, vol. 6, no. 1, pp. 37–43, 2017.

[48] "Open Database Lichess Standard Chess." Disponível em: `https://database.lichess.org/#standard_games`. Acesso em: 28 de nov. de 2022.

[49] "Computer Chess Rating Lists Games." Disponível em: `http://www.computerchess.org.uk/ccrl/4040/games.html`. Acesso em: 30 de nov. de 2022.

[50] B. Kapicioglu, R. Iqbal, T. Koc, L. N. Andre, and K. S. Volz, "Chess2vec: Learning Vector Representations for Chess," *arXiv preprint arXiv:2011.01014*, 2020.

[51] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[52] "Pytorch." Disponível em: `https://pytorch.org/docs/stable/index.html`. Acesso em: 06 de dez. de 2022.

[53] N. Van, L. Lee, J. Lee, P. Bath, M. Halbutogullari, M. Shafi, E. Morton, A. Recalde, V. Chiang, N. Thota, and Z. Munad, "Developing a Human-Level Chess Player with Artificial Neural Networks," 2019. `https://github.com/ucdchessai/chess-ai/blob/master/report/report.pdf`.

[54] "Tensorflow." Disponível em: `https://www.tensorflow.org/api_docs/python/tf`. Acesso em: 06 de dez. de 2022.

[55] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding Batch Normalization," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[56] "Open Database Lichess Puzzles." Disponível em: `https://database.lichess.org/#puzzles`. Acesso em: 28 de nov. de 2022.