UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Lucca Mariano

Solução do jogo de xadrez através de redes convolucionais profundas e aplicações da modelagem na indústria aeronáutica.

São Carlos

2021

Lucca Mariano

Solução do jogo de xadrez através de redes convolucionais profundas e aplicações da modelagem na indústria aeronáutica.

Monografia apresentada ao Curso de Engenharia Aeronáutica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Aeronáutico.

Orientador: Prof. Dr. Álvaro Martins Abdalla

São Carlos

2021

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

MM333s

Mariano, Lucca
Solução do jogo de xadrez através de redes convolucionais profundas e aplicações da modelagem na indústria aeronáutica / Lucca Mariano; orientador Àlvaro Martins Abdalla. São Carlos, 2022.

- Programa de e Área de Concentração em -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2022.

1. Redes Neurais Convolucionais. 2. Aprendizagem com Reforço. 3. Xadrez. 4. Árvore de Monte Carlo. 5. Engenharia Aeronáutica. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO
Approval sheet

Candidato / *Student*: Lucca Mariano
Título do TCC / *Title* : Implementação de redes neurais artificiais na avaliação de posições e variantes em partidas de xadrez.
Data de defesa / *Date*: 22/03/2022

Comissão Julgadora / *Examining committee*	**Resultado** / *result*
Professor Assistente Hernan Dario Ceron Muñoz	Aprovado.
Instituição / Affiliation: EESC - SAA	
Professor Doutor Ricardo Afonso Angélico	Aprovado
Instituição / Affiliation: EESC - SAA	Ricardo A. Angélico

Presidente da Banca / Chair of the Examining Committee:

Professor Assistente Hernan Dario Ceron Muñoz
(assinatura / *signature*)

AGRADECIMENTOS

Ao meu pai Paulo e minha mãe Roseli pela educação ímpar, incentivo e infindável amor que me proporcionaram. Agradeço, ainda, a inacreditável confiança que depositaram em mim e minha empreitada de estudar longe do conforto do lar. Foi fundamental para o meu crescimento.

Ao meu irmão Bruno por ser meu melhor amigo, em todos os momentos da minha vida. Agradeço o aprendizado diário que é desfrutar da sua companhia e companheirismo.

À minha namorada Caroline pelo imensurável amor, apoio e paciência nos últimos sete anos. Agradeço por sempre estar disposta à se aventurar pela penumbra com uma companhia extremamente duvidosa.

Ao professor Álvaro Martins Abdalla, orientador extremamente atencioso e dedicado, que aceitou o desafio de me orientar em tema tão heterodoxo. Foi imprescindível para a realização desse trabalho.

A todos os docentes que trabalharam exaustivamente para me fazer um ser humano melhor. Obrigado por exercerem com tanta maestria a mais nobre das missões.

Agradeço à escola de Engenharia de São Carlos por muito me capacitar, como aluno, profissional e ser humano. Os constantes desafios, aos quais fui exposto diariamente ao longo da minha graduação, forjaram em meu cerne uma anti-fragilidade.

Por fim, agradeço aos meus irmãos de graduação - e assumo a liberdade de citá-los - Lucas, Vinícius, Luís e Pedro. Esses anos dourados não teriam o mesmo brilho sem o bom humor, cumplicidade e alto astral de vocês. Sempre representaram fonte inesgotável de inspiração, motivação e energia para superar minhas limitações.

“"Mas, doutor... Eu sou o Pagliacci."
Boa piada. Todo mundo ri. Rufam os tambores. Desce o pano”
Alan Moore.

RESUMO

MARIANO, . **Solução computacional do jogo de xadrez por aprendizagem com reforço e redes convolucionais.**. 2021. 59p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2021.

O presente trabalho busca implementar uma solução computacional em *Python* para o milenar jogo de xadrez, utilizando-se de redes neurais convolucionais profundas e algoritmos de aprendizagem com reforço. Como referência de desenvolvimento, examinou-se arquitetura estado-da-arte em jogos determinísticos de dois jogadores - o revolucionário e generalista projeto *AlphaZero*, da *DeepMind*. Concluiu-se que para jogos de alto fator de *branch*, o custo computacional de uma solução exclusivamente baseada em aprendizagem com reforço é muito elevado, optando-se por soluções híbridas, dotadas de um pré treinamento supervisionado. Atingiu-se, dessa forma, um modelo capaz de aprender o jogo de xadrez com resultados satisfatórios. Por fim, o caráter generalista da implementação permite uso em projetos de otimização da indústria aeronáutica, com estudos contundentes de caso.

Palavras-chave: Redes Neurais Convolucionais. Xadrez. Aprendizagem com reforço. Árvore de Monte Carlo.

ABSTRACT

MARIANO, . **Solving the game of Chess through reinforcement learning and deep convolutive neural network.**. 2021. 59p. Monograph (Conclusion Course Paper) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2021.

This work aims to solve the timeless game of chess through reinforcement learning algorithm and deep convolutional neural networks. Investigations were carried out in state of art solutions for deterministic games - *DeepMind*'s top notch engine *AlphaZero* - and its approach. It's been concluded that high branching factor games require large amount of computational resources, making the solely reinforcement learning approach unfeasible. An hybrid solution of supervised and reinforcement learning has been used for this work purpose, leading to satisfactory results.

Keywords: Convolutional Neural Network. Chess. Reinforcement Learning. Monte-Carlo Tree Search.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ReLU	Unidade Linear Retificada
BN	Normalização de *batch*
MCTS	Árvore de Decisões de Monte Carlo
SL	Aprendizagem supervisionada
RL	Aprendizagem com reforço
MLP	Perceptron de múltiplas camadas

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto

O tradicionalíssimo jogo de xadrez é objeto de estudo, paixão e curiosidade desde sua concepção, há dois milênios. Entre os fatores que justificam seu sucesso e popularidade - atemporais e, na presente conjuntura, crescentes - destacam-se a acessibilidade e natureza desafiadora que o jogo - também classificado como esporte, arte e ciência - promete em partidas singulares.

O primeiro motivo é de extrema nobreza e detém beleza inata. O xadrez demanda raros recursos para ser adequadamente praticado: dois entusiastas, na areia de uma praia, com conchas representando peças, podem jogar sem quaisquer prejuízos. Constituído por regras simples, claras e quantitativamente diminutas, o xadrez tem a mágica capacidade de conectar pessoas complexamente distintas - em gênero, raça, classe e cultura - por meio de uma eletrizante batalha no tabuleiro. Comunicando-se através das peças, que deslizam elegantemente sobre casas bem definidas, o xadrez é linguagem universal.

Ademais a graciosidade supracitada, não se deve subestimar o poder de entretenimento que o jogo é capaz de fornecer. Como enunciou brilhantemente Seirawan (2003), uma partida habilita a possibilidade de se travar guerra fisicamente segura, ainda que psicologicamente extenuante. Análogo às grandes batalhas, o posicionamento, coordenação e trabalho de cada peça é submetido à forte abordagem analítica, com sofisticadas estratégias de ataque e defesa, avanços e recuos, sequências de arremate e fugas cinematográficas.

De maneira abreviada, mas convenientemente didática, podemos examinar cada cada posição no tabuleiro sob ótica *posicional* e *tática*. Na primeira, realiza-se a avaliação de uma determinada configuração de peças. Seja pela comparação material - quantidades em cada exército -, pelo controle da porção central do tabuleiro ou pela segurança de cada rei, o raciocínio para proclamar vantagem em uma posição conhecida pouco requer análises dos próximos lances. A abordagem tática, por sua vez, tem natureza numérica, sendo fortemente amparada pela capacidade de cálculo do jogador. Em termos enxadrísticos, *cálculo* corresponde à varredura de cada árvore de jogadas e respectivas respostas, numa exaustiva procura por pequenas vantagens.

É intuitivo concluir que a etapa de avaliação tem caráter intrinsecamente humano, enquanto o cálculo é exercido com máxima excelência por circuitos integrados. O desenfreado aperfeiçoamento dos microprocessadores foi suficiente para ultrapassar, com cruel facilidade, nossa inspeção de lances possíveis. O avanço das ciências da computação superou, por fim, nossa implacável perícia em abstrair padrões, comportamentos e respostas de sistemas complexos. Máquinas enxergam com maior profundidade que humanos. E já

aprendem, também, com mais qualidade.

1.2 Motivação

Em 1997, Garry Kasparov, então Campeão Mundial de Xadrez, protagonizou evento inédito ao ser derrotado pelo supercomputador da IBM *Deep Blue* em um *match* de seis partidas oficiais. A disputa em questão se tratava de uma revanche entre as partes: no ano anterior, Kasparov vencera a máquina com margem razoável de dois pontos. Ao ser perguntado sobre o nível médio do *Deep Blue*, após o primeiro confronto, Garry respondera que "em certas posições, 3100 (rating), em outras, 2300". Para fins comparativos, Kasparov detinha o maior *rating* de Xadrez em seu tempo, aquilatado em 2851. Um Grande-Mestre ordinário matinha-se próximo dos 2500. Kasparov compreendia que em posições essencialmente táticas, a máquina tinha poder incomparável, porém sob circunstâncias mais posicionais, fracassava em encontrar lances triviais para jogadores de menor expertise (KASPAROV, 2017). Para a segunda disputa, a IBM trabalhou com uma vasta equipe de renomados enxadristas, a fim de ensinar critérios humanos na avaliação de posições conhecidas. *Deep Blue* foi aluno excepcional e, ao final da sexta e última partida do lendário *match*, Kasparov sintetizou sua frustração em frase icônica: "Sou um ser humano. Quando vejo algo muito além do meu entendimento, sinto medo".

Findou-se, então, a longínqua disputa entre o mais habilidosos dos humanos e a mais municiada das máquinas. Havia claro vencedor. Conjecturou-se sobre o inevitável desinteresse que o fato novo poderia trazer ao xadrez. Jogadores realizariam preparações com potentes computadores - aniquilando o caráter democrático do xadrez competitivo. Além disso, máquinas apresentavam baixa tentação por posições caóticas, de sacrifícios misteriosos e cenários dramáticos de jogo. Partidas arrastadas, com empates prováveis desde o primeiro lance tornar-se-iam tônica padrão daquele que, um dia, foi o mais artístico dos esportes da mente.

A história escrita, no entanto, divergiu da narrativa pessimista. Se em 1997 Kasparov conseguiu dois empates e uma vitória no *match* de seis partidas, tais feitos já não podem mais ser replicados. Para atingir tamanha excelência, os melhores motores computacionais do jogo trouxeram para o tabuleiro sacrifícios românticos, posições abertas, de difícil avaliação e a adrenalina característica das melhores partidas da história. A mais bonita maneira de se jogar é, em grande parte dos casos, também a mais eficaz.

O estado da arte em *engines* de xadrez - e quaisquer jogos fechados de dois jogadores, tais quais *Go* e *Shogi* - é o *AlphaZero*, desenvolvido pela *DeepMind* - grupo de pesquisas subsidiário da Google. Dada a natureza generalista de seu algoritmo - qualquer jogo determinístico pode ser masterizado pela *engine* através de aprendizado reforçado - sua implementação contempla às mais diversas propostas de otimização.

1.3 Objetivos

Objetivo Geral

O presente trabalho objetiva desenvolver, amparado em modelos de redes neurais artificiais com aprendizado reforçado, motor computacional capaz de predizer lances vencedores em uma partida de xadrez. Avaliar-se-á, ao fim, a transposição desta modelagem para implementações na indústria aeronáutica.

Objetivos Específicos

- Desenvolver, implementar e treinar modelo em redes convolutivas de avaliação e seleção de lances para qualquer configuração de tabuleiro.
- Utilizar abordagens e estratégias similares às implementadas nas *engines* estado da arte.
- Validar sucesso da implementação através de posições com desequilíbrio - vencedoras ou perdedoras - e desempenho contra outros motores computacionais.
- Utilizar *Python* como ferramenta de prototipagem do modelo.
- Investigar cenários de implementação do modelo na indústria aeronáutica.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo, examinaremos a arquitetura de rede neural utilizada no motor computacional desenvolvido, discorrendo sobre seus principais elementos constitutivos. Adiante, será explicitado o algoritmo de aprendizagem, bem como suas particularidades e principais vantagens no presente projeto. Finalmente, soluções computacionais estado da arte para o jogo de xadrez serão dissecadas, da implementação à qualidade de resultados obtidos.

2.1 Arquitetura da Rede Neural

2.1.1 Perceptron

O neurônio é a unidade básica e fundamental para o adequado funcionamento de uma rede neural, biológica ou artificial. Uma modelagem ilustrativa de um neurônio genérico está disposta na Figura 1, da qual depreendem-se elementos chave para a compreensão dessa unidade.

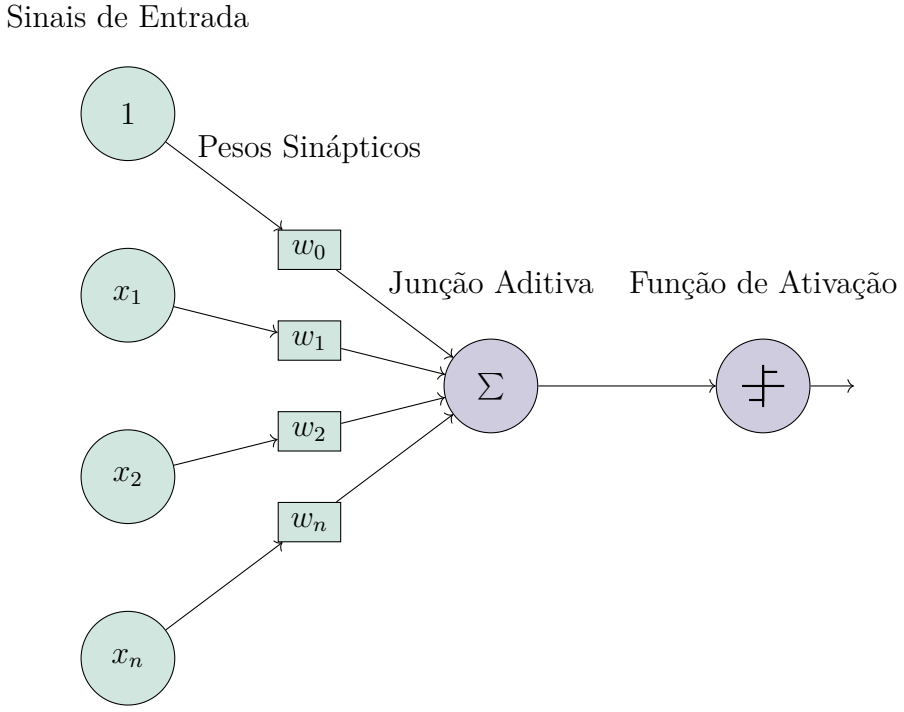


Figura 1 – Modelagem de um neurônio genérico

Do modelo acima, tem-se uma camada de sinais de entradas, que são multiplicados por pesos sinápticos, concatenados em um combinador linear - de natureza aditiva - e avaliados por uma função de ativação, com caráter típico não linear e restritivo. No topo dos sinais de entrada, reside um *bias*, cujo intuito é incrementar ou reduzir a entrada da

função de ativação (HAYKIN, 1999). Podemos sintetizar o neurônio pelo consecutivo par de equações

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

$$y_k = \phi(u_k + b_k) \tag{2.2}$$

Onde x_j corresponde aos m sinais de entrada do neurônio, w_{kj} descreve os pesos sinápticos atuantes nas entradas, u_k é a combinação linear das entradas ponderadas pelos pesos, y_k é a saída da função de ativação ϕ, com b_k representando o *bias* da unidade neural k inspecionada.

As funções de ativação, também conhecidas como *funções restritivas*, são responsáveis por normalizar a saída de um combinador linear em um intervalo finito de interesse. Intervalos típicos tendem a ser unitários - [0, 1] - ou alternativos - [-1, 1]. Em abordagens que demandam maior granularidade na resposta predita, a função de ativação mais recorrente é a sigmoide, de caráter monótono crescente, diferenciável e forma similar à letra *s*. Pode-se exemplificar uma função sigmóide através da função logística, definida por

$$\phi(u_k + b_k) = \frac{1}{1 + e^{a(u_k + b_k)}}$$

Onde a é o parâmetro de inclinação da função sigmóide. Sua natureza - com balanceamento razoável entre seções lineares e não lineares - torna a função sigmóide extremamente atrativa na formulação de uma rede neural, apresentando maior facilidade em generalizar dados de entrada e classificar respostas complexas (HAYKIN, 1999). Para modelos cuja saída esperada é a predição de uma probabilidade, sua utilização é muito eficaz, dado que qualquer evento provável tem chances compreendidas no intervalo contínuo [0, 1], imagem da função logística.

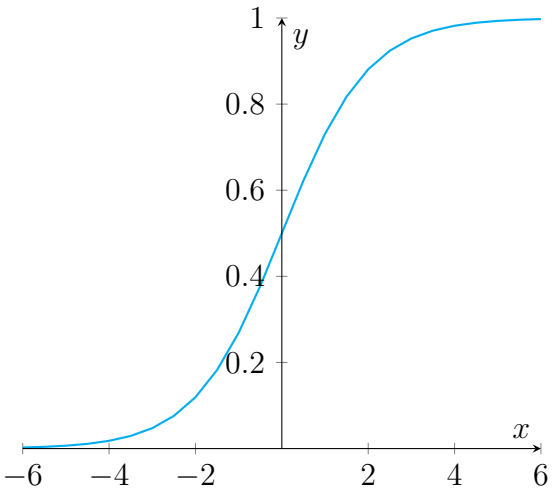


Figura 2 – Função logística.

Depreende-se da Figura 2 que, para entradas cada vez mais positivas, a resposta esperada se aproxima assintoticamente do máximo da função, indicando maior *confiança* na predição realizada. O raciocínio é análogo para entradas negativas.

Em certos cenários, é desejável que a saída da função de ativação esteja contida no intervalo contínuo [-1, 1]. Para modelos de avaliação de posições de xadrez, podemos interpretar as respostas como cenário vencedor para brancas (1) ou negras (-1). Empates e posições de vantagem diminuta devem circundar a origem. Neste escopo, tem-se a função tangente hiperbólica para a ativação, descrita pela equação:

$$\phi(u_k + b_k) = \tanh(u_k + b_k)$$

Permitir que a saída de uma função de ativação assuma valores negativos fornece vantagens analíticas ao sistema(HAYKIN, 1999). Entradas moderadamente negativas devem retornar resposta de mesmo sinal, com o mesma lógica para as entradas positivas. Dessa forma, torna-se simples orientar um sistema determinístico conhecido, através de recompensas ou punições para cada entrada fornecida. O comportamento da função tanh u_k está ilustrado na Figura 3

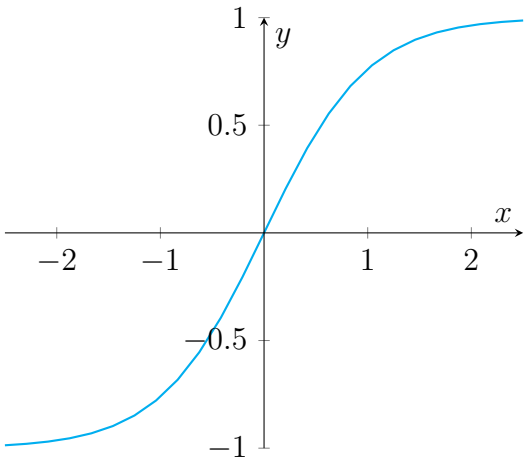


Figura 3 – Função tangente hiperbólica.

Tem-se, enfim, a representação adequada de um perceptron, unidade básica de uma rede neural artificial. Conforme discutido em Haykin (1999), a organização, estrutura e comunicação entre neurônios artificiais apresenta forte dependência do algoritmo de aprendizagem adotado. Explicitado em 2.2.4, o adotou-se a aprendizagem com reforço no presente trabalho, utilizando método heurístico para seleção dos melhores lances disponíveis.

Adiante, serão avaliadas diferentes camadas de neurônios artificiais, com suas especificidades e atuações particulares. O entendimento de cada camada será de importância fundamental para o desenvolvimento bem sucedido da rede neural artificial pretendida.

2.1.2 Perceptron de Múltiplas Camadas

Uma rede conexionista apresenta, geralmente, arquitetura de camadas múltiplas: uma camada de entrada, uma ou mais camadas ocultas de neurônios e uma camada de saída. As entradas são propagadas pela rede de forma sequencial, camada por camada. Tal estrutura típica se trata do perceptron de múltiplas camadas, disposto na Figura 4

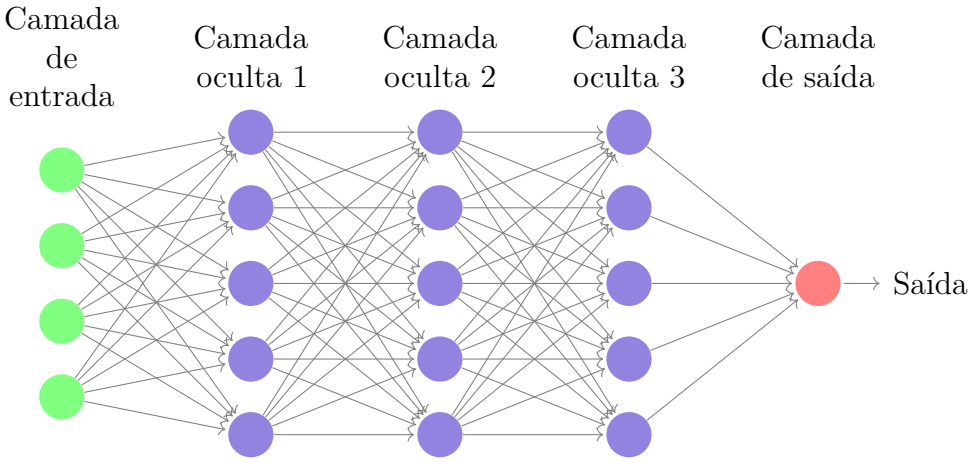


Figura 4 – Perceptron de Múltiplas Camadas com três camadas ocultas.

Por um período substancial da história de redes neurais profundas - redes dotadas de camadas ocultas - houve grande dificuldade para treinar satisfatoriamente os perceptrons de múltiplas camadas. Os estudos desenvolvidos em (RUMELHART; HINTON; WILLIAMS, 1985) apresentaram o algoritmo de retropropagação de erros, extremamente bem sucedido em sua proposta e ainda utilizado no treinamento de redes conexionistas. O algoritmo supracitado é constituído por duas etapas principais: inicialmente, as entradas são propagadas "adiante", através das camadas, com sentido ordinário à saída da rede. O erro da resposta é devidamente computado, através de uma função de perda. Aquilata-se, então, a contribuição individual de cada conexão no erro global, em sentido reverso - dirigindo-se à entrada da rede. Finalmente, os pesos sinápticos das camadas, inicialmente fixos, são atualizados por intermédio de gradiente descendente (HAYKIN, 1999). Desse modo, a resposta real do sistema é estatisticamente ajustada para a resposta esperada. Aliados ao algoritmo de retropropagação de erro, os perceptrons de múltiplas camadas têm desempenhado com excelência na solução de problemas complexos.

2.1.3 Perceptron de Múltiplas Camadas Convolutivas

A arquitetura de rede pretendida nesta subseção é uma categoria particular de perceptron de múltiplas camadas. Tratam-se das redes convolutivas - ou convolucionais - extremamente populares em projetos sofisticados, com entradas dimensionalmente grandes. A motivação para o desenvolvimento das redes convolutivas remonta estudos revolucionários

do córtex visual de gatos e, posteriormente, de macacos. De acordo com Hubel (1959), parte considerável dos neurônios no córtex visual apresentam *campo receptivo local*, isto é, reagem exclusivamente à estímulos de uma região diminuta do campo visual. Quando os campos receptivos são sobrepostos, o campo visual é remontado em completude.

Inicialmente, o projeto de redes neurais convolutivas objetivava identificar padrões em entradas bidimensionais com alto grau de invariância quanto à métodos de distorção - imagens são excelente exemplo desse tipo de entrada Haykin (1999). Para que a tarefa supracitada seja bem sucedida, a rede deve ser capaz de *extrair características locais* - em que a localização exata de uma característica é menos relevante do que sua posição relativa às demais -, *definir mapas de características* - cada mapa é representado por um plano, no qual ocorre o compartilhamento dos pesos sinápticos entre os neurônios que o acessam - e *realizar subamostragem da entrada* - reduzindo a dimensionalidade e sensibilidade dos mapas de características.

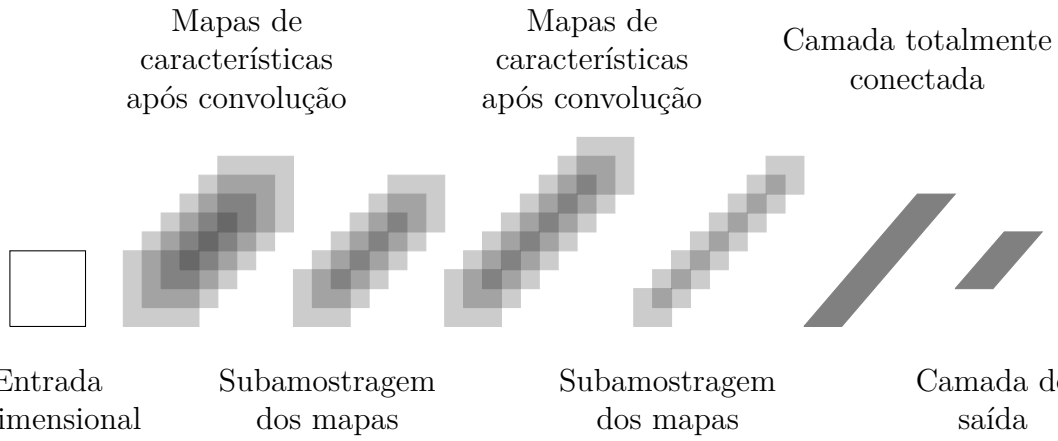


Figura 5 – Arquitetura de rede neural convolucional, como proposto por LeCun, Bengio *et al.* (1995), com alternância entre camadas convolucionais com retificadores lineares e camadas de subamostragem.

Da Figura 5, temos a estrutura de uma rede convolutiva, com uma entrada bidimensional sendo sucessivamente filtrada por camadas convolucionais e camadas de subamostragem. A cada convolução, o número de mapas de características aumenta, enquanto a resolução do mapa diminui, comparado ao mapa da camada imediatamente anterior. Tal conceito é análogo à especificação celular de tecidos: células com complexa são precedidas por células mais simples, generalistas (HUBEL; WIESEL, 1962).

O processo de aprendizagem em uma rede similar à Figura 5 deve contemplar mapeamento não linear de características - de alta complexidade e dimensionalidade - através de algum conhecimento prévio e/ou empírico sobre os dados de entrada. Segue-se, então, com o ajuste fino dos pesos sinápticos - compartilhados nos mapas característicos - e do valor de *bias*, através de um algoritmo de correção de erros - a retropropagação

apresenta boa compatibilidade com a rede - e um conjunto de dados para treinamento da rede.

2.1.4 Camadas Completamente Conectadas

Como a própria nomenclatura sugere, camadas completamente conectadas são camadas em que todos os neurônios estão conectados aos neurônios da próxima camada. Na arquitetura de redes convolucionais, costumam ser a última camada neural antes da saída. Dessa forma, garante-se que toda a informação, proveniente de cada unidade da rede, será computada na resposta final.

2.1.5 Gradientes de fuga e explosão de gradientes

Durante a modelagem de redes neurais artificiais profundas, não são raras as instâncias em que o número de parâmetros livres, treináveis configura alto custo computacional, demandando muitos recursos para treinamento adequado. Outros problemas são típicos dessas estruturas mais "inchadas", entre eles os gradientes de fuga ou a explosão de gradientes.

Remontando as explicações anteriores, o mecanismo de retropropagação retorna o gradiente de erro, aquilata as participações de cada conexão da rede no erro global e realiza ajustes de peso e *bias* em cada neurônio. Conforme o algoritmo alcança camadas mais profundas da rede, os gradientes de erro ficam progressivamente menores. Assim, a alteração nos pesos sinápticos das camadas anteriores é praticamente desprezível durante a etapa de retropropagação, impedindo boa convergência da resposta. Tal inconveniente é denominado *problema do gradiente de fuga*.

Em alguns cenários, ocorre a situação contrária: quanto mais profunda a camada apreciada, maior o gradiente da função de perda, culminando em atualizações exorbitantes dos pesos sinápticos e resposta com caráter divergente. Nesse caso, tem-se o fenômeno da explosão de gradientes. A natureza instável desses gradientes em redes neurais profundas foi cientificamente identificada no trabalho de Glorot e Bengio (2010), onde se observou que funções de ativação sigmóide, aliadas ao método padrão de inicialização dos pesos, contribuem para um aumento crescente da variância nas saídas. Assim, quanto mais profunda a camada, maior a chance de ocorrer saturação do gradiente de erro. Próximo das assíntotas [0, 1] - assumindo função de ativação ϕ logística - a derivada local é desprezível, fato que compromete a retropropagação para as camadas inferiores.

No intuito de contornar a adversidade exposta, serão abordadas duas estratégias elegantemente simples, porém de enorme eficácia.

2.1.5.0.1 Retificação Linear - ReLU

Uma maneira de evitar os problemas de instabilidade dos gradientes é selecionar uma função de ativação *melhor*. Avaliamos ao longo do presente capítulo as vantagens de se aplicar uma função sigmóide - a tangente hiperbólica parece sob medida para a solução enxadrística - e, definitivamente, é de grande interesse mantê-la na arquitetura final da rede. Uma saída atrativa, portanto, reside em acrescentar retificadores linear intermediários às camadas com $\phi = \tanh u_k$. Um ReLU pode ser descrito como uma função degrau com fronteira de ativação $\mathbb{N}$, de modo que

$$\alpha(x) = \begin{cases} x, & \text{se } x > 0 \\ 0, & \text{para demais casos} \end{cases} \tag{2.3}$$

Assim, na porção linear da função - valores positivos de x - as derivadas devem assumir valor sensível e, por consequência, a retropropagação não é comprometida. Aproveita-se, portanto, da não-linearidade de funções sigmóides - excelente para aprendizado de padrões finos - enquanto se evita o problema do gradiente de fuga.

2.1.5.0.2 Normalização de Batch

Ainda que o emprego correto de retificadores lineares reduza consideravelmente as chances de gradientes de fuga/explosão de gradientes, sua exclusiva utilização não é garantia de que em estágios avançados do treinamento da rede eles não ocorram.

Uma abordagem possível para minimizar tal risco é a normalização de *batch*, que consiste em normalizar o valor e centralizar a origem de cada entrada, buscando otimizar suas médias e escalas. Para tal, o algoritmo avalia parâmetros estatísticos - especificamente, média e desvio-padrão - de cada entrada individual (IOFFE; SZEGEDY, 2015).

Após aplicar a normalização de *batch*, porém, tem-se que a média estatística é equivalente a zero, enquanto a variância da saída vale um. Tais valores não retratam de forma fidedigna a realidade da camada: suas entradas podem ter diferente distribuição e esta também deve ser deduzida pelo modelo, uma vez que o aprendizado é conduzido por *batches*, inspecionando todo o conjunto de dados para treino. São aquilatados, portanto, parâmetros γ e β, correlatos respectivamente ao escalonamento e deslocamento da saída.

Atestou-se que a normalização de *batch* teve impacto significativo nos resultados das mais diversas redes neurais profundas, com destaque extremamente positivo às redes convolutivas (IOFFE; SZEGEDY, 2015). A penalidade de sua implementação, claro, diz respeito ao acréscimo de custo computacional e consequente aumento no tempo para predição.

2.2 Soluções estado da arte: *AlphaZero*.

Concluindo o capítulo, faz-se necessário examinar com afinco a melhor solução computacional desenvolvida para jogos determinísticos de dois jogadores. Passando por breve reconstituição histórica até sua factual implementação, as próximas seções buscam demonstrar o feito revolucionário no xadrez computacional que *AlphaZero*, *engine* estado da arte, concretizou através de uma arquitetura simples de redes neurais convolucionais e algoritmo de aprendizagem com reforço (SILVER *et al.*, 2018).

Explicitado em minúcias em 1.2, a disputa entre jogadores humanos e oponentes artificiais foi encerrada em 1997, após chocante vitória do supercomputador *Deep Blue* sobre o então Campeão Mundial de Xadrez, Garry Kasparov. As máquinas e programas de xadrez continuaram progredindo, aproveitando-se de microprocessadores cada vez mais potentes, funções de avaliação cuidadosamente ajustadas por Grandes-Mestres e emprego dos algoritmos de busca *alpha-beta* - responsáveis por varrer com excelência infinitas árvores de lances possíveis. Ainda em 2016, quando o *software* norueguês *Stockfish 8* venceu o Campeonato Mundial de Xadrez Computacional, as estratégias e arquiteturas das mais poderosas *engines* eram demasiadamente similares ao muito antiquado *Deep Blue*(SILVER *et al.*, 2018). O progresso consistia em aprimorar uma já impecável função de avaliação e tornar a investigação de lances mais rápida e eficiente.

AlphaZero foi desenvolvido com o propósito de substituir as sofisticadas funções de avaliação por algoritmos de aprendizagem com reforço. Por intermédio de métodos estatísticos de seleção de lances, amparado por uma modelagem robusta de redes convolucionais e *nenhum conhecimento do domínio* - *AlphaZero* tem ciência exclusiva da lógica do jogo: seu conjunto de regras e ações legais em cada posição(SILVER *et al.*, 2018). Através da experiência - adquirida após inúmeras partidas contra seu próprio algoritmo - a *engine* compreende padrões, lances candidatos, táticas refinadas e posições vantajosas.

Na subseção 2.2.1, apreciaremos a elegantemente singela arquitetura de sua rede convolutiva. Adiante, em 2.2.4, compreenderemos seu algoritmo de treinamento e seleção de movimentos.

2.2.1 Arquitetura de Rede

A arquitetura de rede desenvolvida é, em síntese, uma rede convolucional profunda, dotada de *conexões de salto*. Tais conexões atuam complementarmente às camadas de retificador linear a normalização de batch, com forte intuito de evitar problemas de gradiente e aprimorar a convergência das respostas.

A entrada do modelo corresponde à presente configuração do tabuleiro, encodada em planos bidimensionais 8×8. Esperam-se duas saídas da rede: um vetor, contendo as probabilidades individuais de todo lance possível e uma avaliação global da posição

averiguada.

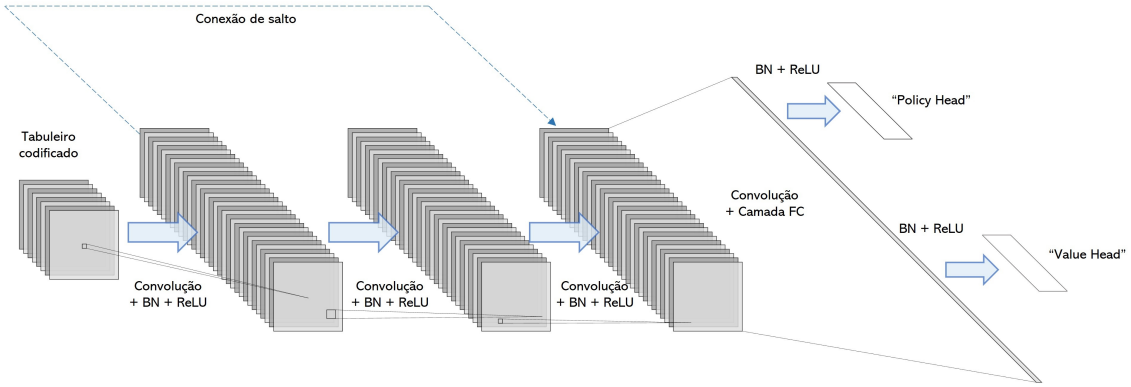


Figura 6 – Arquitetura da rede convolucional presente em *AlphaZero*.

Na Figura 6 encontra-se disposta a arquitetura da rede neural supracitada: a entrada é inicialmente tratada por uma camada convolutiva com 256 filtros e *kernel* 3×3. Prossegue-se, então com uma normalização de *batch* e retificação linear da resposta. O próximo passo consiste em uma cadeia de 19 blocos residuais, compostas por duas camadas convolutivas, intercaladas, novamente, por ReLU e BN. Há, ainda, uma conexão de salto ligando a entrada do bloco residual a sua saída.

Por fim, a resposta é tratada em duas frentes diferentes: a "*policy head*", correspondente às probabilidades individuais de cada movimento possível e a "*value head*", responsável por atribuir avaliação - vencedora, perdedora ou empatada - à posição jogada em cada turno. Em ambas as frentes, as saídas são precedidas por uma camada convolucional, uma camada completamente conectada e as típicas camadas de normalização de *batch* e retificação linear.

2.2.2 Entradas da rede

Assumindo que o modelo busca predizer lances para uma configuração de tabuleiro, é intuitivo concluir que a entrada da rede deve representar o estado atual do jogo. A dificuldade reside em "traduzir"para o modelo os tipos de peças presentes, em que casas estão localizadas, qual jogador deve fazer o próximo movimento e se existem direitos de *roque* - reorganização de duas peças em único lance, desde que regras mínimas sejam cumpridas.

Optou-se por codificar o estado de jogo em planos bidimensionais 8×8 - de mesma dimensão do tabuleiro. São doze planos correlatos aos posicionamentos das peças - seis tipos de peça para dois jogadores - quatro planos computando os direitos de roque, um

plano com informação de turno e três planos contadores, a fim de registrar empates por repetição ou limite de lances sem desenvolvimento.

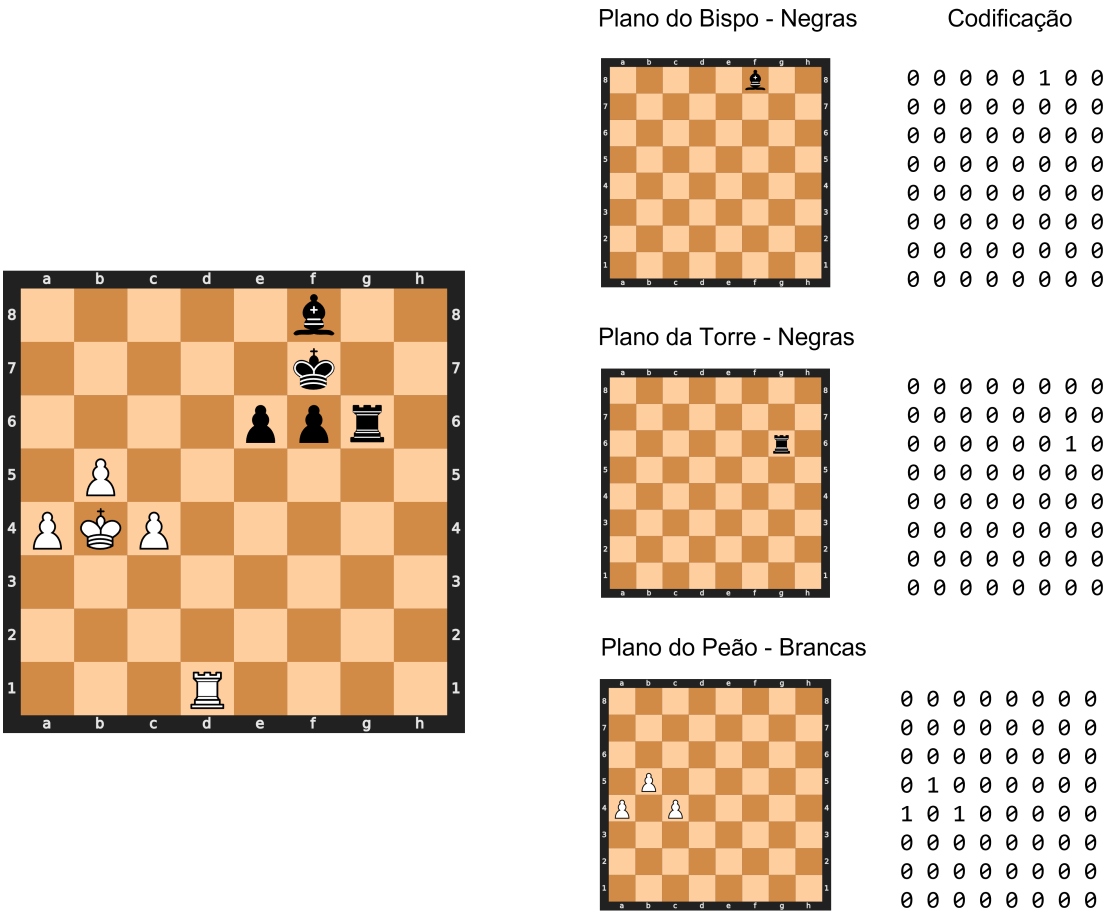


Figura 7 – Planos de peças para uma posição conhecida.

A Figura 8 ilustra os planos das peças: quando presentes no tabuleiro, sua posições espaciais assumem valor unitário. O plano de turnos é completamente preenchido com 1 ou 0 - primeiro caso para turno das brancas e segundo, naturalmente, das negras. Os quatro planos de roque - correlatos as quatro variações do lance - são integralmente unitários quando o movimento é legal e zerado em caso contrário. Os planos contadores seguem lógica parecida: se a posição se repetiu ao menos uma vez, torna-se um plano exclusivamente de valores unitários.

Por fim, são computados os últimos sete estados do jogo. Observou-se que o conhecimento do turnos anteriores contribuiu positivamente para um treinamento mais veloz, eficaz e com resultados de melhor qualidade.

2.2.3 Saídas do modelo

Conforme explicitado em 2.2.1, a rede neural deve retornar um vetor com as probabilidade dos lances possíveis - "*policy head*- e uma avaliação global do estado do jogo - "*value head*". A concepção de um vetor de probabilidades de lances é desafiadora: ele deve sempre ter mesma dimensão, ainda que a lista de movimentos legais seja dinâmica - diferentes tamanhos para posições distintas. Um exemplo didático desse problema é observado na própria abertura do jogo. Brancas somente podem mover peões e cavalos. Os bispos, torres, rainha e rei estão bloqueados por peças aliadas. Caso um peão de bloqueio seja movido na abertura, a lista de movimentos disponíveis aumenta sensivelmente.

A abordagem então adotada apresenta elegante simplicidade. Para computar todo e qualquer movimento de uma partida de xadrez, assumiu-se existência de uma peça *ligeiramente desbalanceada*, com capacidades motores de rainha e cavalo. A inacreditável peça é, então, posta sobre todas as 64 casas do tabuleiro e seus lances legais são concatenados. Desse modo, garante-se que toda jogada, desempenhada por qualquer peça em qualquer casa, já foi apreciada.

É evidente que o peão de *e2* não tem o direito de atacar o peão adversário em *e7* na abertura, apesar do movimento ser considerado evento provável na abordagem escolhida. Para contornar a situação, as probabilidades dos lances ilegais são arbitrariamente anuladas, enquanto as demais são recomputadas - garantindo que a soma das probabilidades de lances legais seja equivalente a 1.

Uma possibilidade não atacada pelo método da "super-peça"é a coroação de um peão. Quando a unidade consegue chegar à primeira fileira adversária, o peão recebe direito de se tornar rainha, torre, bispo ou cavalo. Existem circunstâncias em que a promoção não é trivial: transformar peão em rainha pode levar à um desperdício inacreditável de vitória. Essa discussão não é pertinente para a presente seção, mas a coroação de peças "menores"tem enorme valor e deve ser considerada. São apreciados quatro lances possíveis para cada peão na segunda fileira inimiga, um para cada unidade de promoção. O vetor de movimentos prováveis, por fim, terá dimensão fixa, contemplando 4672 jogadas em qualquer posição de entrada.

2.2.4 Aprendizagem com reforço: Árvore de Monte Carlo

Dispondo de uma sofisticada rede convolucional profunda, utiliza-se da Árvore de Decisão de Monte Carlo para acessar os melhores lances disponíveis em uma posição conhecida. Na buscas de jogadas pela árvore, os *nós* correspondem às configurações de tabuleiro acessadas e as conexões entre dois nós são representadas pelos movimentos realizados. Cada conexão armazena quatro parâmetros de desempenho atualizáveis: Q, correlato à avaliação do lance, N, computando o número de visitas que a conexão recebeu, W, variável que soma a avaliação de todos os lances disponíves e P, representando a

probabilidade inicial do lance jogado. Todos os parâmetros são inicializados nulos, com exceção de P - que assume valor da posição jogada. Serão apresentadas, a seguir, as quatro etapas fundamentais do algoritmo MCTS: *seleção*, expansão, *simulação* e *retropropagação*.

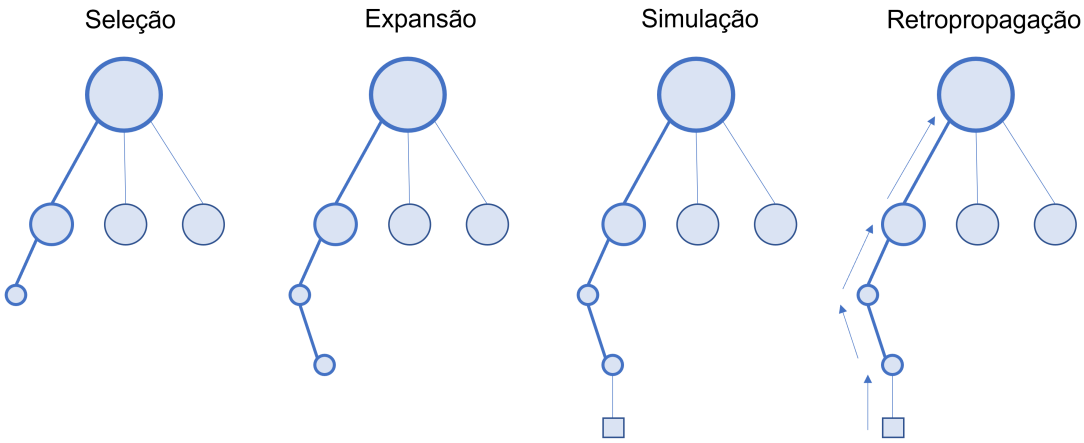


Figura 8 – Esquema genérico do algoritmo MCTS.

Durante a *seleção*, inicia-se a busca na "raiz"da árvore de decisões e o nó que maximiza a soma $Q + u$ é selecionado, com u descrito por

$$u = c \cdot P \cdot \frac{\sqrt{\sum_m N_m}}{1 + N_m} \tag{2.4}$$

Onde c é uma constante de recompensa, o m diz respeito ao lance jogado e o termo $\sum_m N_m$ corresponde ao número de conexões que o "pai"do presente nó contém. Vale ressaltar que, a medida que mais nós são inspecionado, o termo u é perde relevância e Q torna-se dominante na expressão. A importância de u reside nas primeiras iterações, incentivando a seleção de jogadas com maior probabilidade inicial P.

Na etapa de expansão, a avaliação de um nó ainda não explorado é estimada pela rede convolutiva, bem como as probabilidades individuais de cada movimento possível. O nó é, então, expandido em seus lances legais.

Existe uma fundamental diferença entre a simulação do MCTS em *AlphaZero* e a simulação em Árvores de Monte Carlo típicas, respaldadas em literatura de métodos estatísticos. Na presente abordagem, utilizam-se dados da etapa de expansão e predições da rede para recomputar v.

Finalmente, a retropropagação atualiza os parâmetros Q, N e W dos nós - partindo de um nó terminal à raiz da árvore. Tem-se, então

$$N' = N + 1 \tag{2.5}$$

$$W' = W + v \tag{2.6}$$

$$Q' = \frac{W'}{N'} \tag{2.7}$$

A busca é concluída selecionando o lance de maior π_m, descrito por

$$\pi_m = \frac{N_m^{\frac{1}{\tau}}}{\sum_n N_n^{\frac{1}{\tau}}} \tag{2.8}$$

Onde τ é um parâmetro de ajuste da exploração de nós - impedindo que se opte sempre pela jogada mais visitada.

Integrando uma poderosa rede neural convolucional ao algoritmo de Monte Carlo, tem-se modelo extremamente efetivo na aprendizagem de jogos determinísticos. Para seu adequadamente treinado, *AlphaZero* joga partidas completas contra si mesmo, registrando os tabuleiros em todos os turnos, as probabilidades dos lances possíveis e o resultado final da partida - vitorioso ou perdedor, a depender do jogador que comanda o turno apreciado. Obtém-se, dessa forma, lote de partidas suficiente para treinar a rede que, imediatamente após o treinamento, é atualizada no algoritmo MCTS. Um novo lote de partidas é gerado e a rede, novamente, aprimorada. O processo é repetido à exaustão.

De acordo com o exposto em Silver *et al.* (2018), a etapa de treinamento do *AlphaZero* teve 700.000 iterações, com *mini-batches* de 4096 posições distintas. Cinco mil TPU's - unidades de processamento hiper-especializadas em operações com tensores - foram utilizados para gerar jogos a partir de *self-play*, enquanto 16 TPU's de segunda geração conduziram o treinamento da rede convolucional. O tempo total de aprendizagem foi estimado em nove horas. Os resultados estão dispostos na Figura 10.

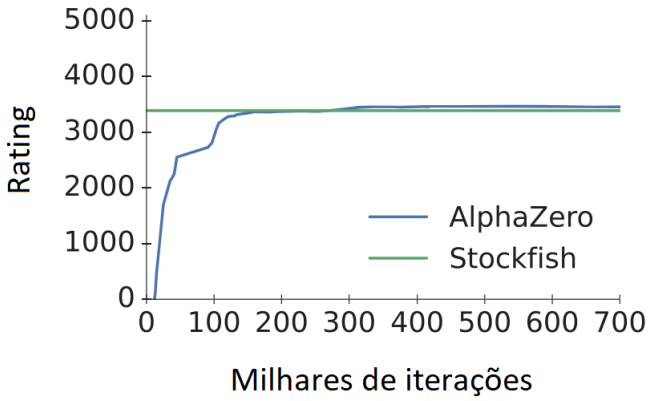


Figura 9 – Rating do *AlphaZero* parametrizado pelo número de iterações de treinamento. Extraído integralmente de *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*, 2018 .

A implementação de *AlphaZero* foi marco revolucionário no xadrez computacional. Por intermédio de graciosa estratégia *self-play*, seu motor computacional compreende conceitos táticos, posicionais e estratégicos sem qualquer exposição à catálogos de aberturas, repertórios de livros ou manuais de fim de jogo. Dois milênios de conhecimento acumalado são superados em quatro horas de treinamento do modelo (SILVER *et al.*, 2018). Ao confrontar as melhores *engines* de aprendizado supervisionado - com insumos refinadíssimos, partidas magnânimas dos jogadores de elite - *AlphaZero* é implacável. Seu nível é super humano, não pela profundidade de lances que inspeciona, mas por enxergar padrões de jogo que nunca fomos capazes de compreender.

3 METODOLOGIA

Objetivou-se no presente trabalho reproduzir modelagem similar à discorrida em 2.2, utilizando o *Python* como linguagem para o desenvolvimento do algoritmo de aprendizagem com reforço. A escolha dessa ferramenta é guiada pela grande facilidade de escrita, leitura e interpretação de seus *scripts* - tornando a implementação mais acessível - além da natureza open-source e, portanto, gratuita para uso e distribuição. A linguagem ainda apresenta grande integração com os módulos TensorFlow e Keras, de extrema utilidade em projetos de Machine Learning e estruturação de redes neurais profundas.

Apesar de seguir filosofia muito próxima da proposta em Silver *et al.* (2018), simplificações foram adotadas no intuito de minimizar o custo computacional da rede. O vetor de saída que retorna as probabilidades dos lances foi dimensionalmente reduzido, sem prejuízo de informação. Na metodologia do *AlphaZero*, alguns movimentos previstos não ocorrem em circunstância alguma - como, por exemplo, coroação de peões em casas centrais. Tal alteração reduz o vetor de saída de 4672 lances possíveis para 1968. Além disso, o Tensor de Entrada não contém informações de estados prévios do tabuleiro. Com essa alteração, o número de canais do tensor passa de 119 para 17, valor razoável para a etapa de treinamento, ainda que culmine em pequeno prejuízo de generalização. Os demais métodos foram preservados e satisfatoriamente replicados no presente projeto.

3.1 Arquitetura de Rede

No projeto desenvolvido, reproduziu-se de maneira fidedigna o desenvolvimento de Silver *et al.* (2018), com mesma estrutura, sequência de camadas, codificação das entradas e políticas de saída. A Figura 6 retrata adequadamente a implementação. As alterações no Tensor de Entradas e na dimensão de uma das saídas foram discutidas na abertura do capítulo.

3.2 Aprendizagem com Reforço

Realizou-se implementação da Árvore de Decisões de Monte Carlo em acordo com os estudos de Silver *et al.* (2018), expostos na seção 2.2.4. Na lógica adotada, os nós devem preservar informações de estado do tabuleiro, enquanto as conexões correspondem a movimentos particulares de uma posição conhecida. Depois, durante a fase de expansão, cada lance legal deve gerar novas conexões e subsequentes nós - contendo o estado de um tabuleiro gerado pelo lance dessa conexão particular. Intermediariamente, é solicitado à rede neural as probabilidades dos lances possíveis e a avaliação global da posição. Esses dados são armazenados nas conexões, juntamente aos parâmetros de desempenho daquela

ramificação da árvore. Em cada nó, verifica-se condição terminal. Em caso positivo, a rede neural não é solicitada e o resultado da partida compõe a etapada de retropropagação.

Os jogos, depois de finalizados, são então agrupados em lotes de centenas. Esses lotes apresentam informação completa para o treinamento da rede - etapa imediatamente posterior.

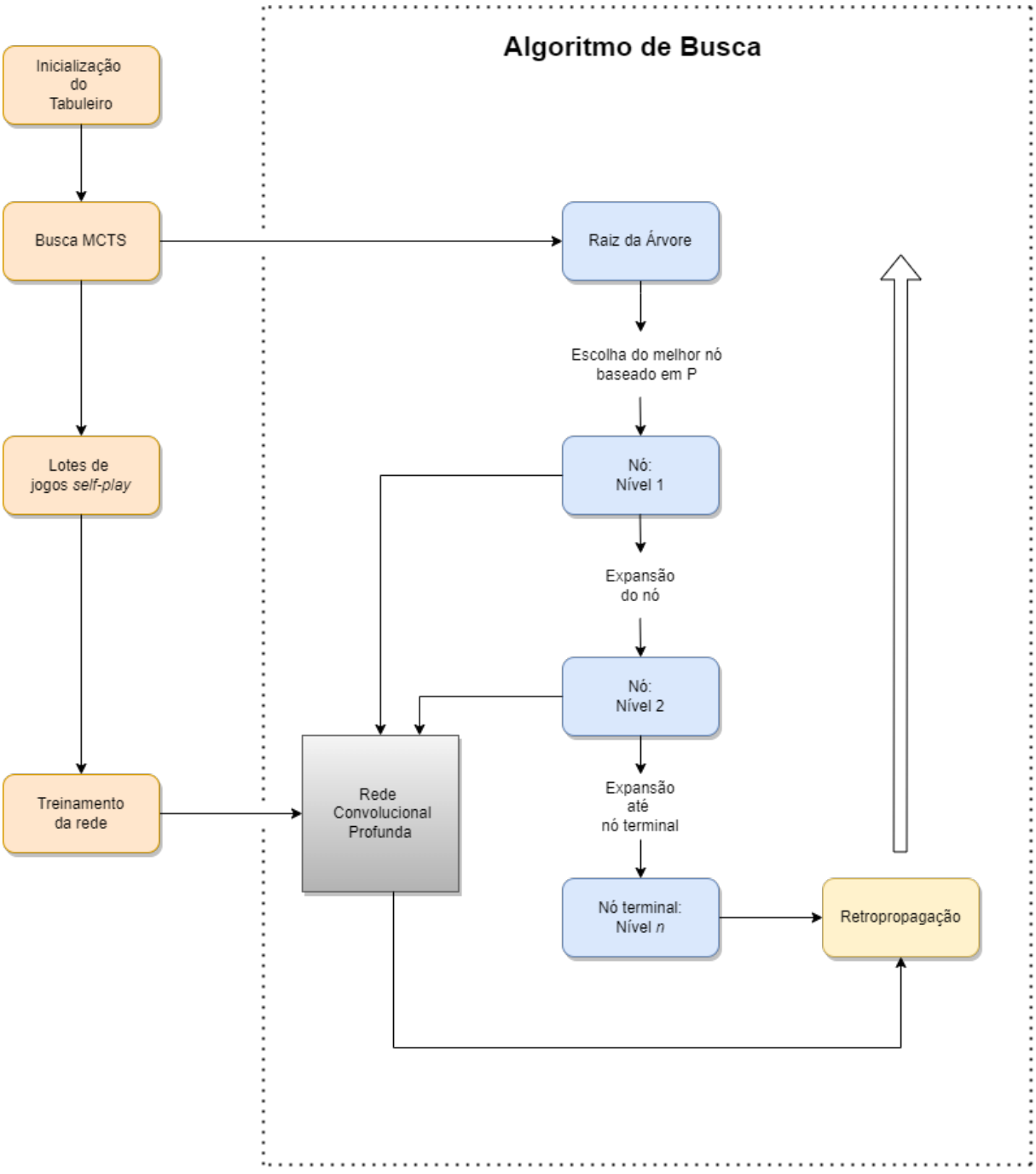


Figura 10 – Fluxograma do modelo integrado MCTS e rede convolucional profunda.

3.3 Aprendizagem supervisionada

Aliado ao processo de aprendizagem com reforço, desenvolveu-se paralelamente uma rotina de aprendizagem supervisionada, visando comparar a qualidade da solução obtida por *self-play* com MCTS e a saída de uma rede exposta a um seleto conjunto de partidas, praticadas entre Grandes-Mestres de elite.

A arquitetura de rede é idêntica à aplicada na aprendizagem com reforço. O diferencial entre os métodos reside no conjunto de treino: são fornecidas 3.000.000 de posições de tabuleiro - presentes em partidas do mais alto nível de xadrez - com probabilidade máxima para o lance executado e nula par os demais e avaliação posicional coerente com resultado final da disputa.

O conjunto de partidas exemplares foi extraído gratuitamente do portal *Lichess Elite Database*. Por meio de filtragens de rating e titulação dos jogadores, é possível selecionar de forma fina a qualidade das partidas de treino. O tamanho dos *batches*, a taxa de aprendizado e a dimensão do Tensor de Entradas é ajustado empiricamente para evitar *overfitting*.

3.4 Otimização do código

É fundamental ressaltar que o treinamento de uma rede competitiva de xadrez demanda exorbitante capacidade computacional - *AlphaZero* foi desenvolvido em linguagem de alta performance e estava municiado com mais de 5000 TPU's(SILVER *et al.*, 2018). Para aprimorar o desempenho do *script* em uma máquina caseira, duas soluções foram adotadas e explicitadas adiante.

3.4.1 Vetorização das operações

Operações matriciais de grande dimensão são preferíveis à *loops* executando, repetidamente, as mesmas operações lineares. A retropropagação é um exemplo de operação que deve ser vetorizada, no intuito de melhorar o desempenho do código. *GPU's*, unidades de processamento gráfico, são especialistas na manipulação e solução desses problemas. No presente projeto, foi utilizada uma placa gráfica dedicada *NVIDIA GTX 1660 Ti*, com desempenho mais do que satisfatório para problemas tipicamente vetorizados.

3.4.2 Processos em *multithreading*

Uma operação é paralelizável quando não faz parte de uma cadeia ordenada de execuções, independe de quaisquer entradas e sua saída não interfere nos processos de execução simultânea. Para aumentar significativamente a performance do algoritmo MCTS, uma atraente possibilidade consiste em paralelizar as raízes das árvores de decisão, gerando partidas *self-play* simultâneas em diferentes *cores* da unidade de processamento selecionada.

4 RESULTADOS E ANÁLISES

4.1 Aprendizagem com reforço

A implementação do algoritmo de Monte-Carlo, integrada às redes convolucionais profundas, foi bem sucedida. As etapas de seleção, expansão, simulação e retropropagação tiveram o comportamento esperado, em claro acordo com o proposto na seção 2.2.4. A mesma consideração é valida para a arquitetura da rede neural: apesar de uma natureza sofisticada nos tensores de entrada e a dupla política de saída, o funcionamento da rede atendeu às expectativa estabelecidas.

Ainda que o modelo tenha alcançado resultados satisfatórios de implementação, sua performance não possibilitou treinamento exclusivo por *self-play*. Nas subseções que seguem, são explorados os principais pontos de dor do projeto e seus impactos na qualidade dos resultados.

4.1.1 Performance da linguagem

Para projetos de aprendizagem de máquinas - parte crucial do presente desenvolvimento - análise de dados e prototipagem de soluções computacionais, *Python* é uma opção muito sedutora, por razões já elencadas ao longo deste trabalho. É de se notar, no entanto, que a linguagem é consideravelmente mais lenta do que alternativas de mais "baixo nível". Tal inconveniente ocorre porque *Python* é linguagem interpretada, demandando mais instruções para que suas linhas sejam executadas. Ademais, o processo de alocação de memória é dinâmico, realizado por um gerenciador de memória que, com incômoda frequência, sobrestima a quantidade necessária para o armazenamento de uma variável.

Através do módulo *cProfile*, realizou-se o mapeamento de um jogo completo praticado pela *engine*. O tempo de conclusão de uma partida foi aquilatado em 542 segundos, demasiadamente lento para se alcançar resultados qualificados no treinamento. Demandam-se lotes de jogos completos para cada passo de treino e, na velocidade apresentada, o processo se torna inviável.

As quatro funções mais onerosas do código foram, então, identificadas e estão dispostas na Figura 11

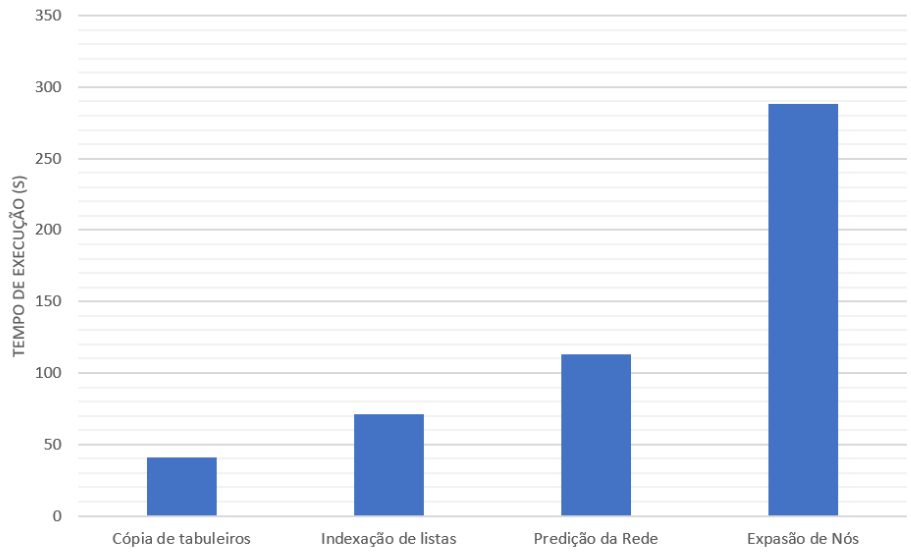


Figura 11 – Tempo de execução das quatro principais funções ofensoras, de acordo com *cProfile*

Dos escopos das funções listadas, temos:

- **Expansão de Nós** - Principal etapa do algoritmo de busca em MCTS. É fundamentalmente uma função de exploração de lances e, em um cenário de muitas ramificações - *high branching factor* elevado - é esperado caráter computacionalmente onerosa. Sua participação no tempo total de execução é de aproximadamente 56%.
- **Predição da Rede** - Autoexplicativo, corresponde às consultas e solicitações de probabilidade/avaliação posicional durante as fases de expansão e simulação. Pouco pode ser aprimorada sem mudança intensiva na arquitetura da rede ou melhoria do *hardware* em que é executada.
- **Indexação de Listas** - Referente aos processos de listagens e comparações de índices ao longo do algoritmo. É muito acionada para comparar o vetor de probabilidades de lances com a lista de lances legais para uma dada posição no tabuleiro.
- **Cópia de Tabuleiros** - Para cada expansão de nó, um novo *objeto* de tabuleiro deve ser gerado, idêntico ao tabuleiro do nó imediatamente superior, a fim de realizar lances sem comprometer o estado atual de jogo.

As principais otimizações realizadas após o mapeamento supracitado atacaram a *Indexação de Listas* e a *Cópia dos tabuleiros*, cujo aprimoramento é realizável. A lista de lances possíveis foi transformada em estrutura de dicionário. Cada jogada recebeu um index inteiro próprio, em pareamento *chave/valor*.

Os tabuleiros tornaram-se característica de cada classe de nós, ao invés de uma lista pouco prática de posições jogadas. Dessa forma, cada nó tem informação exclusiva de seu respectivo tabuleiro, sem necessidade de consultar uma variável global de grande dimensão.

A função de *Expansão de Nós* se aproveitou de algumas melhorias gerais - em especial, da abordagem de dicionários descrita anteriormente - e recebeu, em algumas partes específicas da função, método de compilação *Just-In-Time*, mais próximo das linguagens de baixo nível.

Após a implementação destas melhorias, o modelo completou uma partida em 353 segundos. Uma vez que jogos distintos apresentam diferente número de lances até sua conclusão - bem como árvores de decisão fundamentalmente distintas - pouco se pode extrair de um único mapeamento. Idealmente, dispondo de um bom espaço amostral de partidas, pode-se realizar tratamento estatístico desses aprimoramentos. No entanto, o ainda elevado tempo de execução torna a alternativa pouco atrativa. Teríamos um diagnóstico para um problema que, na origem, não podemos solucionar. A linguagem apresenta limitações intrínsecas à sua natureza interpretada.

Feitas essas considerações, o novo perfil de execução ainda é de grande utilidade., Pode-se avaliar a nova contribuição percentual de cada função no algoritmo. É perceptível que a etapa de predição da rede tornou-se parte mais onerosa da execução, indicando que as alterações resultaram em algum ganho de performance.

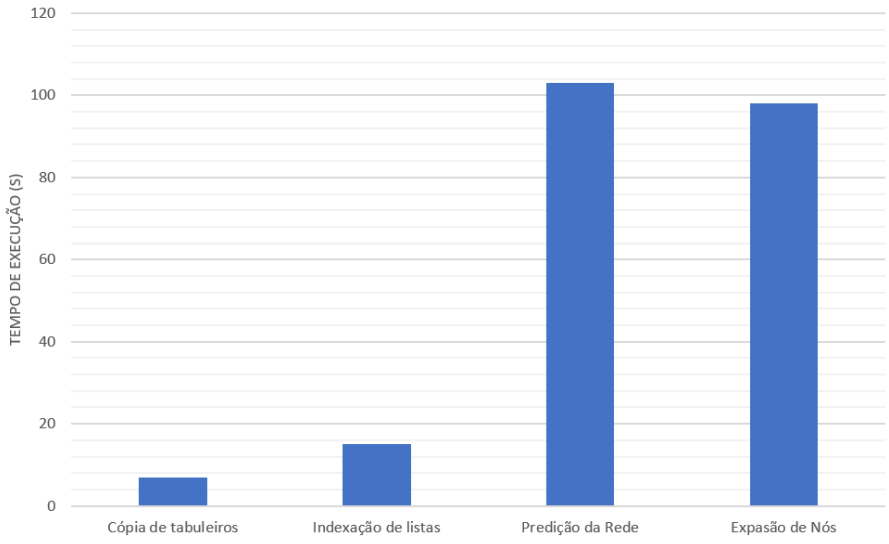


Figura 12 – Tempo de execução das quatro principais funções ofensoras, em novo mapeamento com *cProfile*

4.1.2 Jogos excessivamente longos

Partidas de xadrez entre seres humanos apresentam duração média de 40 lances. Em novembro de 2021, durante o Torneio Mundial de Xadrez - principal competição do circuito oficial FIDE - a sexta partida entre o campeão Magnus Carlsen e o postulante ao título Ian Nepomniatchi tornou-se a partida mais longa da história do torneio. Foram 136 lances, jogados em cerca de oito horas.

Dado o início vagaroso do algoritmo MCTS - a *engine* ainda não faz ideia de como concluir um jogo - raras são as partidas que não chegam ao lance de três dígitos. Esse cenário se reflete em árvores de decisão excepcionalmente custosas, atrasando a aprendizagem do modelo e seu sua performance geral.

4.1.3 Limitações de *hardware* e observações gerais sobre *AlphaZero*

Além do óbvio gargalo de *software*, o modelo é executado em um computador *high-end*, porém caseiro. *AlphaZero* foi treinado em um *cluster* de TPU's, otimizado para operações com tensores, por um período moderadamente prolongado (SILVER *et al.*, 2018).

Além disso, é importante salientar que até cerca de 10.000 iterações de treinamento, *AlphaZero* apresenta nível extremamente precário de jogo (SILVER *et al.*, 2018). Nas primeiras centenas de treinamentos, o modelo parece fazer esforço ativo para perder, obtendo resultados significativamente piores do que um jogador que escolha aleatoriamente seus próximos movimentos. Entre os padrões aprendidos pela *engine*, a captura de peças é inicialmente seu favorito, ainda que resulte em sacrifícios sem compensação.

Após 150 passos de treino - em aproximadas 36 horas - o modelo do presente trabalho apresentava a mesmíssima *natureza suicida* apontada pelo time do *DeepMind*. A fim de aproveitar o algoritmo de MCTS implementado, fez-se necessária uma etapa de pré-treinamento por aprendizagem supervisionada.

4.2 Aprendizagem Supervisionada

Dispondo de um vasto catálogo de partidas brilhantes, realizou-se uma etapa de aprendizagem supervisionada na *engine*. Com as posições de tabuleiro devidamente encodadas, lances exemplares recompensados e avaliações coerentes do estado do jogo, prosseguiu-se com o treinamento da rede convolucional. Após cerca de 100.000 iterações de treino, obteve-se para a acurácia do modelo em um *set* de validação, resultado ilustrado em 13

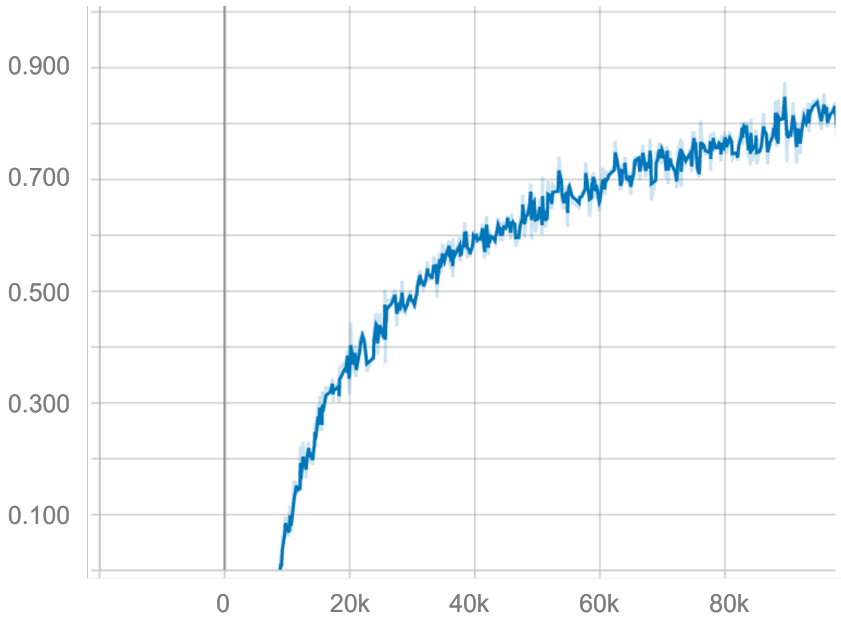


Figura 13 – Acurácia do modelo frente a um conjunto de validação

Submeteu-se o algoritmo, então, à etapa de aprendizagem com reforço. Após 10 horas de treinamento - e duzentas e trinta alterações na rede - apreciaremos sua capacidade de predizer lances em três testes de desempenho: primeiramente, contra adversários de saber enxadrístico inferior à *engine*, a fim de determinar se houve aprendizagem do jogo. Depois, é avaliada a capacidade do modelo em identificar posições de avaliação ambígua - armadilhas sensíveis. Por fim, o projeto é posto à prova contra adversário humano.

4.2.1 Oponentes de baixo conhecimento

Nesta brevíssima subseção, será avaliado o desempenho do motor computacional desenvolvido frente a adversários de menor - ou nenhum - conhecimento sobre o jogo. A expectativa é de absoluta dominância da *engine* sobre os oponentes: caso contrário, torna-se difícil atestar que houve sucesso no aprendizado de xadrez.

Em primeira avaliação, o modelo é desafiado por um jogador que *arbitrariamente* seleciona lances aleatórios. Em todos os jogos realizados, o oponente jogou de brancas. Nas 100 partidas realizadas, negras sagraram-se vencedoras em todas: leve indício de sucesso na etapa de aprendizagem.

Posteriormente, a *engine* confrontou a primeira geração de seu algoritmo, após mil iterações de treinamento. Mais uma vez, o modelo foi implacável: venceu todas as 100 partidas contra sua versão desatualizada. Podemos assumir, com certa confiança, que houve factual aprendizado.

4.3 Armadilhas comuns

4.3.1 Scholar's Mate

Também conhecido por "mate do pastor", *Scholar's Mate* consiste em uma maneira veloz de se vencer contra oponentes desavisados. É uma armadilha de baixíssimo nível e que, caso detectada, coloca o jogador de brancas em leve desvantagem.

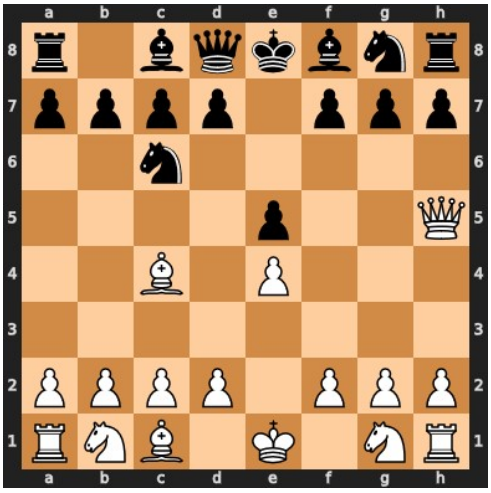


Figura 14 – Disposição das peças para *Scholar's Mate*. Negras jogam.

A Figura 14 corresponde à configuração de um capcioso tabuleiro. O turno é das negras e, caso o jogador desavisado não defenda imediatamente a casa de f7, a dama deve proclamá-la não apenas com cheque, mas sim, com arremate da partida.

Durante reprodução do presente tabuleiro na *engine*, percebemos que, na posição das negras, o lance escolhido é *adequado*: o modelo desliza sua dama para f6, protegendo a ameaça de mate, porém, expondo à própria dama à ataques futuros. Quanto à avaliação global do tabuleiro, a *engine* prediz que brancas possuem ligeira vantagem - quando em verdade a posição é empatada.

Na posição de um jogador de brancas - e assumindo que o adversário faça um lance terrível, como colocar o cavalo em f6 - a *engine* reconhece mate no lance seguinte e o executa.

4.3.2 Légal's Trap

Légal's Trap se trata de uma armadilha substancialmente mais refinada que o "mate do pastor". Nela, assume-se desenvolvimento inicial típico de uma abertura italiana: avanço do peão rei, desenvolvimento do bispo de casas claras e manobras de cavalos. Caso o jogador de negras tente explorar o tema da "peça cravada- melhor visualizado na Figura 15 - um espetacular recurso tático é habilitado.

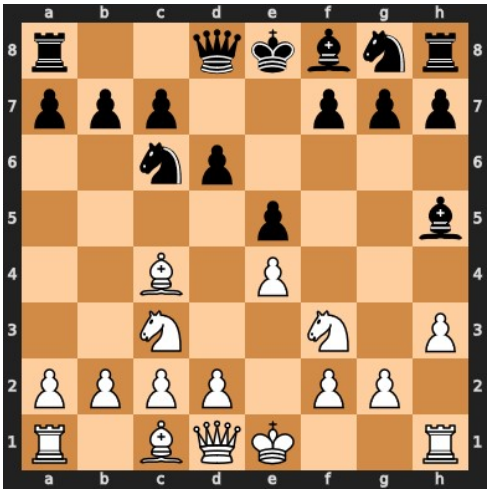


Figura 15 – Disposição das peças para *Légal's Trap*. Brancas jogam.

No tabuleiro fornecido, o cavalo de f3 é, em primeira avaliação, "peça cravada". Caso ele se movimente, a dama das brancas está sob grande vulnerabilidade, dada a presença de um bispo em h5. Contra-intuitivamente, a melhor jogada da posição consiste em capturar o peão de e5, condenando a própria dama ao extermínio. Nesse momento, a *engine* interpreta que a partida está perdida: o bispo de h5 deve executar sua melhor peça e vencer, eventualmente.

Caso a preocupação da *engine* se concretize, a posição, de fato, é terminal: brancas possuem cheque-mate em dois lances. A sequência é graciosa: bispo de c4 toma o peão de f7, colocando o rei das negras em cheque. Após fuga para e7, única rota de escape para o rei, brancas arrematam a partida com cavalo d5.

A modelo não foi capaz de enxergar o belo sacrifício de dama. Em posição invertida, jogando como negras, a *engine* aceita o sacrifício de damas, sacrificando, em contraparte, a partida.

4.3.3 Movses Movsisyan vs Thomas Patton, 2004

A disposição do tabuleiro 16 é obtida após naturais movimentos de abertura, com habitual disputa pelo centro, desenvolvimento dos cavalos e melhorias nas posições dos bispos. Eventualmente, brancas tem a oportunidade de ameaçar a dama inimiga com bispo em g5, protegido pelo cavalo de f3. Uma possibilidade - perdedora - para as negras é bloquear o ataque à dama com o cavalo de g8.

Figura 16 – Disposição das peças para *Movses Movsisyan vs Thomas Patton*, jogada em 2004. Brancas jogam.

Nesse ponto, brancas podem realizar um brilhante lance de cavalo, tomando o peão de d4. Caso negras recapturem a peça, a partida está perdida. Recapturando a peça com o cavalo de c6, a dama está condenada depois de bispo e7.

A recaptura com o bispo de g7 proporciona um belo sacrifício de dama - que, se aceito, resulta em mate imparável. Dama captura o bispo em d4 e, após ser executada pelo cavalo de c6, brancas arrematam o jogo com cavalo f6 - colocando o rei em cheque - e bispo h6, conretizando mate.

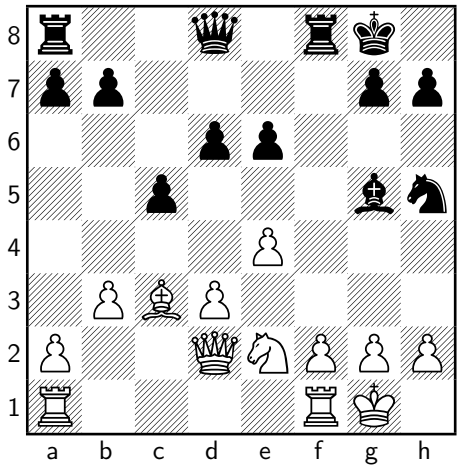
A *engine*, mais uma vez, não foi capaz de encontrar os lances vencedores. Após o belo sacrifício de dama em d4, o modelo desperdiça o arremate artístico, avaliando que o jogo está completamente perdido. Na condição de negras, por sua vez, o *engine* capturaria a dama e imediatamente perderia a partida.

4.4 Partida contra oponente qualificado

O último teste do modelo consiste em uma partida completa contra um adversário humano. O autor, dotado de nível de jogo mediano, tomou a liberdade de desafiar a *engine* desenvolvida no presente trabalho. Deu-se ao participante de menor conhecimento enxadrístico o direito de escolher o lado. O humano optou por jogar de brancas.

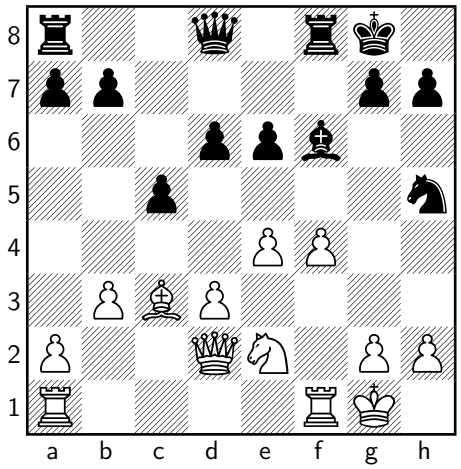
Mariano vs. Engine

1 e4 e5 2 ♘f3 ♘f6 3 ♘c3 ♘c6 4 ♗c4 ♗e7 5 d3 d6 6 O-O ♗e6 7 ♗×e6 f×e6 8 ♕d2 O-O 9 b3 ♘d4 10 ♘×d4 e×d4 11 ♘e2 c5 12 ♗b2 ♘h5 13 c3 d×c3 14 ♗×c3 ♗g5



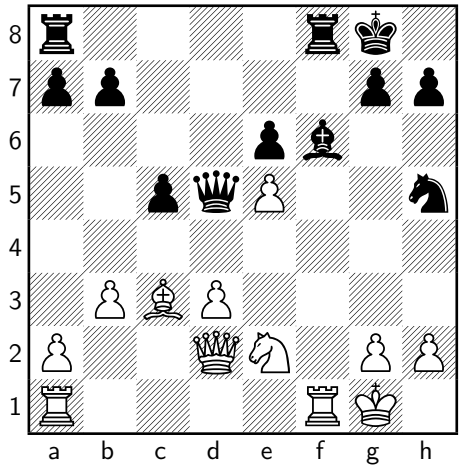
O início da partida foi caracterizado por absoluto equilíbrio. Não foram detectadas imprecisões, erros ou gafes - nomenclatura enxadrística para lances equivocados, em magnitude crescente - e podem-se extrair informações muito positivas sobre o desempenho da *engine*. Os três primeiros lances das negras são classificados "lances de livro", presentes nos repertórios tipicamente humanos de abertura e dotados de bom amparo teórico e literário. Ao mesmo tempo que o fato é forte indicativo de bom aprendizado, a etapa supervisionada pode ter maior influência na escolha da abertura do que, essencialmente, dedução por *self-play*. Um comportamento muito positivo da *engine* desenvolvida diz respeito à defesa ativa de peças, exemplificado no 14... ♗g5, que efetivamente protege a torre de e4 enquanto ataca **♕d2**.

15 f4 ♗f6



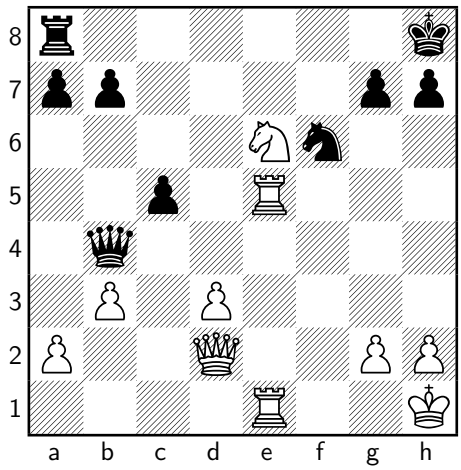
A ideia de **♙f4** reside em aumentar a tensão sobre essa mesma casa. Negras, a priori, não podem propor trocas: f4 é uma casa resguardada por três defensores. A resposta da *engine* não impressiona: 15... ♗f6 possibilita o avanço **♙e5**, ameaçando, ainda, o bispo das negras.

16 e5 d×e5 17 f×e5 ♕d5??



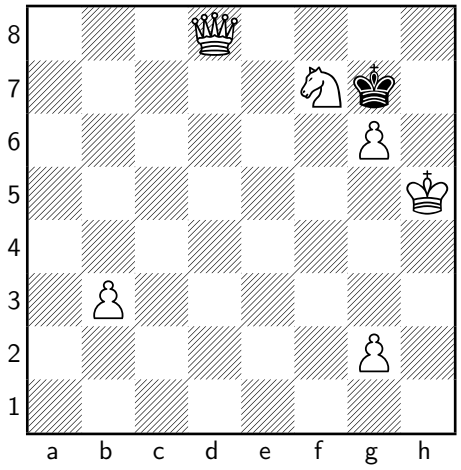
Conforme discutido, brancas avançam o peão para e5 e a pressão sobre o bispo de casas escuras é mantida. A *engine* encontra o único lance que não perde a partida - apesar de trivial - dxe5. Ocorre a recaptura do peão por parte das brancas e, então, negras realizam a gafe fatal da partida: 17... ♕d5??. Além de perder o bispo imediatamente, é fundamental observar que brancas têm diversos atacantes no flanco-rei: **♗c3** se encontra na melhor diagonal do tabuleiro, **♕d2** está desenvolvida e com grandes possibilidades de ataque, **♘e2** tem casas interessantes para manobras e **♖f1** coloca imensa pressão na coluna semi-aberta de f.

18 e×f6 ♖×f6 19 ♗×f6 ♘×f6 20 ♘f4 ♕c6 21 ♖ae1 e5 22 ♖×e5 ♕d6 23 ♖fe1 ♕d4+ 24 ♔h1 ♔h8 25 ♘e6 ♕b4



A partida segue sem grandes imprecisões. Entretanto, brancas conseguem trocar suas peças maiores, dogmático em situações de vantagem material. O lance 25. . . ♕b4 encerra toda e qualquer esperança da *engine* nessa posição.

26 ♕×b4 c×b4 27 ♘c7 ♖d8 28 ♖e8+? ♘×e8 29 ♖×e8+ ♖×e8 30 ♘×e8 a6 31 ♘d6 h6 32 ♘×b7 a5 33 d4 ♔g8 34 d5 ♔f7 35 d6 ♔e8 36 ♘×a5 ♔d8 37 ♘c4 g5 38 a4 b×a3 39 ♘×a3 ♔d7 40 ♘c4 ♔e6 41 ♔g1 g4 42 ♔f2 h5 43 ♔e3 g3 44 h×g3 ♔d7 45 ♔f4 ♔e6 46 ♔g5 ♔d7 47 ♔×h5 ♔e6 48 g4 ♔d7 49 g5 ♔e8 50 g6 ♔f8 51 ♘e5 ♔g7 52 d7 ♔f6 53 ♘f7 ♔g7 54 d8♕??



O final foi protocolar: brancas aproveitaram a vantagem material para continuar simplificando a posição, com claro plano de coroar peões e vencer a partida. No último lance da partida, a única jogada perdedora se tratava da coroação de peão em rainha. Conforme discutido na seção 2.2.2, coroação para peças menores é de extrema importância e, por vezes, ganhar uma rainha corresponde ao desperdício inacreditável de vitória. Dada a tragicomicidade - e caráter didático - de 54 d8♕??, a coroação máxima foi realizada e partida se encerrou em empate por *stalemate*: o turno é das negras, porém, não dispõem de nenhum lance legal na posição. O rei tampouco se encontra em cheque. Como não há possibilidade de passar o turno no xadrez, a partida é encerrada em empate imediato, independente de desbalanço material.

5 APLICAÇÕES DO ALGORITMO NA INDÚSTRIA AERONÁUTICA

A batalha travada entre *engine* e jogador humano legitimou, com razoável segurança, o poder de aprendizado do motor computacional. Houve clara compreensão de preceitos fundamentais do jogo de xadrez - valor intrínseco das peças, táticas simples de captura e defesa, sequências coerentes de abertura de jogo - e ofereceu resistência às investidas do jogador humano, ademais proposição de *contra-jogo* e planos de vitória.

Em primeira análise, conjecturar-se-ia que o presente trabalho tem escopo e utilização bem delimitados, restritos à jogos determinísticos de dois jogadores - em especial, partidas de xadrez. A dita conclusão se mostra, porém, equivocada. A validação da *engine* corresponde, também, à validação do algoritmo de varredura e seleção de cenários finais - o MCTS - e da rede convolucional profunda. Com alguns ajustes às entradas esperadas e saídas de interesse, é possível aplicar mesma metodologia e aprendizagem com reforço para situações e problemas além do tabuleiro de sessenta e quatro casas.

Descrevendo, com clareza, as regras e condição final de um problema particular, é possível consumir o poderoso algoritmo estatístico - acoplado à sofisticada rede convolucional de aprendizagem - para suportar a tomada de decisão em situações complexas. Investigaremos, na presente seção, duas situações exemplares - no escopo da engenharia aeronáutica - em que a aplicação do modelo desenvolvido é bem sucedida e de enorme valor.

5.1 Utilização de MCTS e aprendizagem reforçada no controle de VANT de asa fixa.

Uma vez que a modelagem validada é, sobretudo, generalista, basta um ajuste das dimensões do problema - especificamente, dos tensores de entrada e saída correlatos à rede neural - e uma redefinição dos cenários terminais para transportar sua operação aos mais distintos cenários de interesse. Dentre utilizações bem sucedidas desta arquitetura, inspecionaremos solução, em tempo real, para a trajetória de manobras de VANT's de asa fixa (WANG; WANG, 2020).

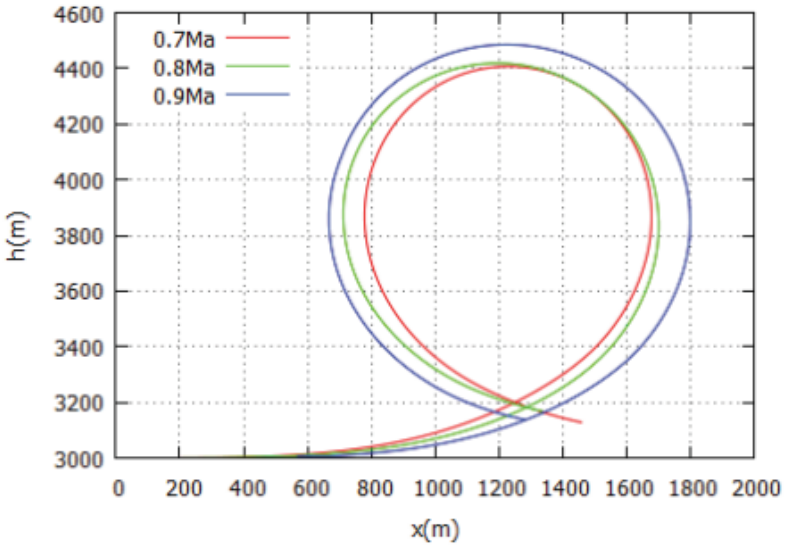
O projeto clássico - e há muito respaldado - para controle autônomo de manobras aéreas é fortemente embasado em trajetórias já conhecidas de manobras especificas, resgatando métodos fundamentalmente empíricos, aliado às robustas leis de controle. Em aplicações factuais, a combinação de diferentes missões de voo e circunstâncias ambientais requer generalização do método supracitado. Assim, para toda manobra - frente a quaisquer condições iniciais - nosso sistema de controle de trajetória deve prover resposta funcional, eficaz e de maneira ágil. Sob tais premissas, a modelagem tradicional é computacionalmente

custosa e inevitavelmente impraticável.

Descreve-se, inicialmente, o comportamento dinâmico de uma aeronave em manobras aéreas através de suas equações de movimento - representadas num Espaço de Estados. Conforme metodologia exposta em Wang e Wang (2020), realizam-se as quatro etapas inerentes ao MCTS - seleção, expansão, simulação e retropropagação - para, enfim, efetuar a ação mais próxima de ocasionar o estado terminal de interesse. A simulação em Wang e Wang (2020) segue fundamento clássico de MCTS: realizam-se ações intrinsecamente arbitrárias até que um cenário final seja atingido. O parâmetro de recompensa é condicionado, portanto, pela inspeção primária de árvores aleatórias e suas circunstâncias terminais. Podemos tangenciar a aleatoriedade desta abordagem - e potencializar a geração e varredura de árvores - através de redes neurais convolutivas. Quando devidamente treinadas, devem predizer de forma satisfatória o valor de cada nó inspecionado e aprimorar significativamente a resposta do algoritmo.

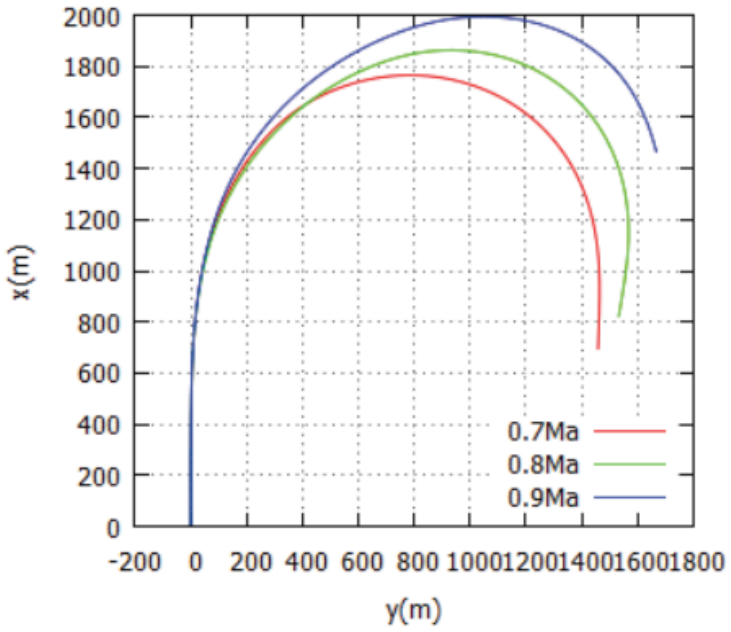
Por fim, legitima-se a solução através de ensaios experimentais com um VANT de asa fixa típico (WANG; WANG, 2020). São aferidas duas manobras - *loop* e *break* - em condições iniciais distintas para velocidade e altitude. Ambas as manobras foram executadas noventa vezes, com sucesso absoluto em todas as tentativas. Ilustrado nas Figuras em 17 e 18, têm-se as trajetórias típica para *loop* e *break*, respectivamente.

Figura 17 – Trajetória típica da manobra *loop*, em diferentes velocidades iniciais.



Fonte: (WANG; WANG, 2020)

Figura 18 – Trajetória típica da manobra *break*, em diferentes velocidades iniciais.



Fonte: (WANG; WANG, 2020)

5.2 Utilização do MCTS e aprendizagem reforçada na otimização de geometria de hélices de VANT.

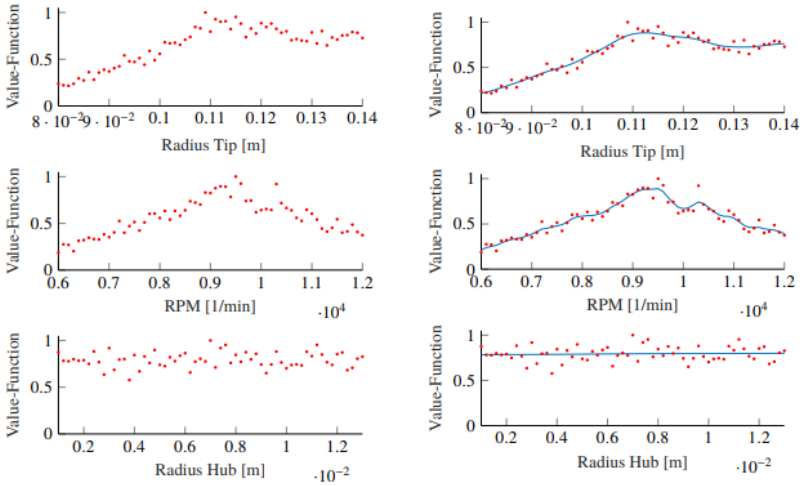
O projeto de hélices eficientes, visando soluções de altíssima performance, impõe imenso desafio aos métodos tradicionais de cálculo de desempenho e algoritmos de otimização. Os parâmetros do sistema são vastos e, para adicionar complexidade à tarefa, geralmente interdependentes (THIELE *et al.*, 2020). Por intermédio da aprendizagem reforçada, em conjunto com uma poderosa rede neural convolutiva, decisões de projeto - tais quais o raio da hélice, suas distribuições de corda e espessura, o *pitch* de operação e número de lâminas - podem ser simplificadas, ao passo que a eficácia da solução é potencializada.

Uma vez que as condições do escoamento são intercambiáveis em cada etapa do voo, é irrealizável propor geometria ideal para todo e qualquer momento da missão. Factível, no entanto, é obter solução ótima para todos os instantes mais relevantes da operação. Serão aferidos, no presente estudo, decolagem e aterrissagem vertical, *hover*, transição para voo horizontal e cruzeiro (THIELE *et al.*, 2020).

De maneira análoga ao presente trabalho e ao estudo de caso exposto em Wang e Wang (2020), são definidas as regras de otimização - os parâmetros mutáveis estão elencados em tabela de máximo e mínimo - e uma função de avaliação - que pode ser substituída pela classificação de uma rede neural convolutiva. Posteriormente, são realizadas as quatro etapas do MCTS, inspecionando, para cada nível da árvore, um parâmetro

distinto (THIELE *et al.*, 2020). Na Figura 19, dispõe-se a evolução da função-valor, frente à alterações nas variáveis aferidas.

Figura 19 – Função-valor de três parâmetros ajustáveis do projeto de hélices.



Fonte: (THIELE *et al.*, 2020)

Por fim, Thiele *et al.* (2020) realiza a comparação entre diferentes algoritmos de otimização multi-objetiva e tomada de decisão. O MCTS alcança resultados extraordinários: apresenta soluções ótimas, convergindo em tempo diminuto. É de grande utilidade frente à problemas com muitas *constraints*, uma vez que aproveita funções de avaliação flexíveis, atualizadas após ações positivas no *design* da hélice.

6 CONCLUSÃO

6.1 Considerações finais

O trabalho objetivou implementar satisfatoriamente os métodos de aprendizagem com reforço utilizados em *AlphaZero* e demais motores computacionais estado da arte. Para tal, apropriou-se dos métodos expostos em Silver *et al.* (2018), utilização a fundamentação teórica de redes neurais proposta em Haykin (1999), desenvolvimento em *Python* com amparo técnico de Géron (2019) e integrando ao modelo preditivo Árvore de Decisão de Monte Carlo, com duas políticas de saída. O trabalho de implementação foi extremamente fiel ao proposto pelo *DeepMind* e o modelo se mostrou funcional, integrado e capaz de predizer lances.

Ressalta-se, no entanto, que a geração de jogos através de *self-play* requer ilimitado recurso computacional, inacessível ao autor. Outro forte motivo para a diminuta geração de dados para treino foi a linguagem de programação utilizada no algoritmo MCTS. Linguagens de mais baixo nível - por exemplo, C - poderiam adicionar maior vazão e velocidade à produção de lotes de jogos, fundamental para adequado treinamento da rede.

Frente à adversidade enfrentada, realizou-se pré treinamento da rede com partidas de alto nível, das quais se conhecia resultado final e a ordem dos lances jogados. Obteve-se êxito com a abordagem supracitada, atingindo boa acurácia nos lotes de treino e validação, além de resultados satisfatórios nos testes da *engine*.

Definitivamente, a característica mais elegante da modelagem desenvolvida é a capacidade de generalizar problemas sem conhecimento prévio de domínio. Ainda que o *subtema* - motivação e paixão pessoal - tenha sido o milenar jogo de xadrez, bastam limitadas alterações na "lógica do jogo"para que o motor desenvolvido seja capaz de aprender e solucionar outros, cenários determinísticos. Conhecendo regras mínimas de um sistema, o algoritmo é capaz de masterizá-lo, de maneira autodidata.

Por fim, foram investigados dois cenários, no escopo da indústria aeronáutica, em que o algoritmo MCTS de aprendizagem reforçada é de enorme valor e apresenta resultados extremamente promissores. Reforça-se, portanto, o caráter generalista da implementação realizada ao longo destas páginas.

6.2 Sugestões para trabalhos futuros

A lista de sugestões para trabalhos futuros é encabeçada pela mudança na linguagem de desenvolvimento. *Python*, apesar facilitar a implementação do algoritmo, foi grande obstáculo no treinamento exclusivo por reforço. Recomenda-se, também, desenvolvimento

e treinamento do projeto através da computação na nuvem, garantindo que *hardware* também não configure uma limitação.

Entre os testes e validações da *engine*, existe grande valor científico em confrontar modelos exclusivamente treinados por aprendizagem com reforço, modelos exclusivamente treinados por aprendizagem supervisionada e modelos de abordagem híbrida.

A última sugestão diz respeito à lógica de jogo: que seja alterada drasticamente. A metodologia aqui desenvolvida tem a conveniente vantagem de ser genérica e independente de domínio, podendo ser implementada para demais problemas determinísticos. Com um pouco de imaginação - e esforço descritivo - também teria enorme sucesso em solucionar problemas cotidianos, desde que bem representados.

REFERÊNCIAS

GÉRON, A. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems**. [*S.l.: s.n.*]: O'Reilly Media, 2019.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. *In*: JMLR WORKSHOP AND CONFERENCE PROCEEDINGS. **Proceedings of the thirteenth international conference on artificial intelligence and statistics**. [*S.l.: s.n.*], 2010. p. 249–256.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. [*S.l.: s.n.*]: Prentice Hall, 1999.

HUBEL, D. H. Single unit activity in striate cortex of unrestrained cats. **The Journal of physiology**, Wiley Online Library, v. 147, n. 2, p. 226–238, 1959.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. **The Journal of physiology**, Wiley Online Library, v. 160, n. 1, p. 106–154, 1962.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *In*: PMLR. **International conference on machine learning**. [*S.l.: s.n.*], 2015. p. 448–456.

KASPAROV, G. **Deep Thinking: Where Machine Intelligence Ends and Human Creativity Begins**. [*S.l.: s.n.*], 2017.

LECUN, Y.; BENGIO, Y. *et al.* Convolutional networks for images, speech, and time series. **The handbook of brain theory and neural networks**, v. 3361, n. 10, p. 1995, 1995.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. **Learning internal representations by error propagation**. [*S.l.*], 1985.

SEIRAWAN, Y. **Play Winning Chess**. [*S.l.: s.n.*]: Everyman Chess, 2003. ISBN 1857443314.

SILVER, D. *et al.* A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. **Science**, American Association for the Advancement of Science, v. 362, n. 6419, p. 1140–1144, 2018.

THIELE, M. *et al.* Development of a reinforcement learning inspired monte carlo tree search design optimization algorithm for fixed-wing vtol uav propellers. 2020.

WANG, H.; WANG, W. Autonomous control of fixed-wing unmanned aerial system by reinforcement learning. 2020.