Universidade Federal do Amazonas - UFAM
Instituto de Computação - ICOMP
Programa de Pós-Graduação em Informática - PPGI

Impactos de curto, médio e longo prazo de funções de inicialização de pesos em NeuroEvolução Profunda

Manaus
2023

Lucas Gabriel Coimbra Evangelista

Impactos de curto, médio e longo prazo de funções de inicialização de pesos em NeuroEvolução Profunda

Exame de Qualificação submetido à avaliação como requisito parcial para a obtenção do título de Mestre em Informática no Programa de Pós-Graduação em Informática do Instituto de Computação.

Orientador: Prof. Dr. Rafael Giusti

Manaus
2023

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Evangelista, Lucas Gabriel Coimbra
E92i Impactos de curto, médio e longo prazo de funções de inicialização de pesos em NeuroEvolução Profunda / Lucas Gabriel Coimbra Evangelista . 2023
91 f.: il. color; 31 cm.

Orientador: Rafael Giusti
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Aprendizagem de Máquina. 2. Neuroevolução Profunda. 3. Funções de Inicialização. 4. Algoritmo Genético. I. Giusti, Rafael. II. Universidade Federal do Amazonas III. Título

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Informática

FOLHA DE APROVAÇÃO

"IMPACTOS DE CURTO, MÉDIO E LONGO PRAZO DE FUNÇÕES DE INICIALIZAÇÃO DE PESOS EM NEUROEVOLUÇÃO PROFUNDA"

LUCAS GABRIEL COIMBRA EVANGELISTA

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Dr. Rafael Giusti - PRESIDENTE

Profa. Dra. Eulanda Miranda dos Santos - MEMBRO INTERNO

Profa. Dra. Elloá Barreto Guedes da Costa - MEMBRO EXTERNO

Manaus, 10 de Abril de 2023

Documento assinado eletronicamente por **Rafael Giusti**, **Professor do Magistério Superior**, em 05/05/2023, às 11:36, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Elloá Barreto Guedes da Costa**, **Usuário Externo**, em 06/05/2023, às 09:13, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Eulanda Miranda dos Santos**, **Professor do Magistério Superior**, em 08/05/2023, às 10:48, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufam.edu.br/sei/controlador_externo.php?

Avenida General Rodrigo Octávio, 6200 - Bairro Coroado I Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181 / Ramal 1193
CEP 69080-900, Manaus/AM, coordenadorppgi@icomp.ufam.edu.br

Referência: Processo nº 23105.015303/2023-59 SEI nº 1467462

Resumo

A Computação NeuroEvoutiva surgiu como uma abordagem promissora para propor arquiteturas de redes neurais sem interferência humana. No entanto, o custo computacional muitas vezes alto dessas abordagens é um sério desafio para sua aplicação e pesquisa. Neste trabalho, analisamos empiricamente práticas padrão com o algoritmo *Coevolution of Deep NeuroEvolution of Augmenting Topologies* (CoDeepNEAT) e o efeito que diferentes funções de inicialização e ativação têm quando os experimentos são ajustados para redes de rápida evolução em números variados de gerações e populações. Comparamos redes inicializadas com as funções He, Glorot e Random em diferentes configurações de tamanho da população, número de gerações, épocas de treinamento etc. Nossos resultados sugerem que a configuração adequada de hiperparâmetros para treinamentos de poucas épocas em cada geração pode ser suficiente para produzir redes neurais competitivas. Observamos também que a inicialização He, quando associada à NeuroEvolução, tende a criar arquiteturas com múltiplas conexões residuais, enquanto o inicializador Glorot tem o efeito oposto.

Palavras Chave: Aprendizagem de máquina, NeuroEvolução Profunda, Funções de inicialização, Algoritmo Genético

Abstract

Neural evolutionary computation has risen as a promising approach to propose neural network architectures without human interference. However, the often high computational cost of these approaches is a serious challenge for their application and research. In this work, we empirically analyse standard practices with Coevolution of Deep NeuroEvolution of Augmenting Topologies (CoDeepNEAT) and the effect that different initialization functions have when experiments are tuned for quick evolving networks on a small number of generations and small populations. We compare networks initialized with the He, Glorot, and Random initializations on different settings of population size, number of generations, training epochs, etc. Our results suggest that properly setting hyperparameters for short training sessions in each generation may be sufficient to produce competitive neural networks. We also observed that the He initialization, when associated with neural evolution, has a tendency to create architectures with multiple residual connections, while the Glorot initializer has the opposite effect.

Keywords: Machine Learning, Deep NeuroEvolution, Initialization functions

Agradecimentos

A Deus, Criador da Vida.

A Jesus, Mestre Divino.

A Chico e Clarêncio, sempre comigo, e demais amigos da Vida Maior que, através do anonimato santificante, sustentaram-me até aqui.

A minha mãe, Lucinéia, mulher de fé.

A meu pai, Valberto, homem de serenidade.

A meus irmãos, Luís, Erick, Gohan e Simba.

A meu orientador, Rafael Giusti, que juntoss compartilhamos muitas horas de aprendizado.

A todos os demais amigos e amigas, professores e professoras, inclusos, que, de muitas formas, inclusive indiretamente, colaboraram para o cumprimento dessa etapa.

À dor e ao sofrimento, amigos que constroem e reformam o caráter.

A Kenneth Stanley, pela sua brilhante contribuição ao campo da Computação Evolutiva.

Epígrafe

"E ainda que tivesse o dom de profecia, e conhecesse todos os mistérios e toda a ciência, e ainda que tivesse toda a fé, de maneira tal que transportasse os montes, e não tivesse amor, nada seria." - 1 Coríntios 13:2

Paulo de Tarso

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Redes Neurais Profundas (RNPs) estão entre os métodos de Aprendizado de Máquina (AM) mais usados atualmente Miikkulainen et al. (2019). Elas podem ser empregadas em diversos contextos, demonstrando habilidade em aproximar funções de alta complexidade, as quais frequentemente desafiam os modelos tradicionais, como Máquinas de Vetores de Suporte (SVM—*Support Vector Machines*) e Redes Neurais rasas (também chamadas Redes Perceptron de Várias Camadas ou *Multi-Layer Perceptron*—MLP). Esses cenários de aplicação incluem reconhecimento de imagem, processamento de linguagem natural, diagnóstico médico, previsão financeira, sistemas de recomendação, análise de sentimentos e jogos eletrônicos.são capazes de aproximar funções que muitas vezes são consideradas muito complexas para modelos "clássicos". Contudo, as Redes Neurais Profundas (RNPs) apresentam uma complexidade considerável, caracterizada por múltiplas camadas de neurônios, conexões ponderadas, funções de ativação e ajustes de parâmetros. Essa intricada arquitetura possibilita o aprendizado de padrões e características não lineares, mas também pode resultar em maior consumo de recursos computacionais e dificuldades na interpretação dos modelos. Portanto seu treinamento geralmente requer conjuntos de dados muito grandes e são computacionalmente caras. Isso é particularmente desafiador até mesmo para a tarefa de ajuste (*fine tunning*) dos hiperparâmetros, pois seu treino e validação pode levar um tempo considerável.

Ao projetar uma RNP, vários fatores devem ser considerados, principalmente a quantidade e disposição das camadas. Uma vez escolhida a arquitetura, treinar uma RNP para uma tarefa específica requer a definição de vários hiperparâmetros, como número de épocas, taxa de aprendizado, função de otimização, *batch size* etc. Normalmente isso é feito treinando modelos com várias configurações diferentes de hiperparâmetros em amostras de treino e validação, o que exige tempo considerável devido ao custo envolvido no treinamento dessas redes.

Considerando-se os avanços na última década no desenvolvimento de modelos de Aprendizado de Máquina Profundo (DL—*Deep Learning*) para lidar com tarefas desafiadoras e o notável esforço envolvido em projetar esses modelos "manualmente", métodos

capazes de encontrar automaticamente arquiteturas de RNPs sem intervenção humana têm-se tornado cada vez mais relevantes. Muitos desses avanços foram possíveis com o relacionamento do campo de DL e Algoritmos Bio-Inspirados, que criou uma nova área de estudo: a NeuroEvolução, também conhecida como NeuroEvolução Profunda (NEP).

A NeuroEvolução Profunda (NEP) trata do uso de algoritmos evolutivos com o propósito específico de otimizar a arquitetura de uma RNP para resolver problemas difíceis (Ma e Xie, 2022). Assim como os algoritmos evolutivos clássicos, a abordagem NEP emprega uma população de soluções subótimas, chamadas indivíduos, na qual o objetivo é identificar e combinar os melhores elementos dos indivíduos para encontrar soluções o mais próximo possível da solução ideal. Isso geralmente acontece em ciclos chamados gerações. No contexto da NEP, isso significa combinar os elementos mais promissores das redes neurais (por exemplo, camadas, neurônios ou blocos de camadas) ao fim de cada geração, criando arquiteturas cada vez mais adequadas para alguma tarefa específica, como classificação, detecção de anomalias ou previsão de séries temporais, dentre outras.

Uma importante questão que tem recebido relativamente pouca atenção em NeuroEvolução é a inicialização dos pesos das redes neurais durante o processo neuroevolutivo. A cada geração, a fim de verificar quais indivíduos são mais promissores, o algoritmo evolutivo produz redes neurais cujos pesos podem precisar ser otimizados a partir de inicializações aleatórias. Existem diferentes funções de inicialização para definir esses pesos iniciais, porém nem todas são igualmente adequadas. Uma má escolha dos parâmetros inciais dos neurônios pode levar a alguns comportamentos indesejáveis, como o problema do *vanishing gradient* e o problema do *gradient explosion* (Kumar, 2017; Katanforoosh e Kunin, 2019). Geralmente, esses fenômenos ocorrem quando os parâmetros tendem a zero ou ao infinito, respectivamente, impossibilitando a convergência do modelo durante o treino (Katanforoosh e Kunin, 2019; Goodfellow et al., 2016).

Na última década, temos observado muitas otimizações alternativas e estudos sobre inicializadores de parâmetros Glorot e Bengio (2010); He et al. (2015). Esses estudos baseiam-se principalmente em duas pesquisas inovadoras que apresentaram as funções mais usadas atualmente: a inicialização de Glorot (Glorot e Bengio, 2010) e a inicialização de He (He et al., 2015). Entretanto, esses estudos não consideram o impacto de funções de inicialização de pesos para redes dentro do processo neuroevolutivo. Muitas funções são criadas considerando determinadas arquiteturas fixas, que é justamente o oposto do que ocorre em NeuroEvolução e em NEP, nas quais as arquiteturas estão em constante mudança de indivíduo para indivíduo e, principalmente, de geração para geração. Em NeuroEvolução, ademais, pouco se tem explorado sobre o benefício de qualquer função de inicialização de peso em detrimento de outras, se diferentes funções de inicialização podem levar a diferentes arquiteturas ou se, até mesmo, as funções tidas como canônicas (He e Glorot) são úteis considerando o contexto da NEP.

1.1 Objetivo do Trabalho

O objetivo geral deste trabalho constitiu avaliar o impacto de diferentes funções de inicialização de pesos no desempenho de redes neurais profundas obtidas a partir de NeuroEvolução. Para tanto, alguns objetivos específicos foram contemplados, a citar:

1. Caracterizar as funções mais utilizadas para inicialização de pesos em Redes Neurais Profundas.

2. Realizar experimentos de curto, médio e longo prazo utilizando diferentes funções de inicialização de pesos em redes neurais provindas de NeuroEvolução.

3. Comparar as práticas de inicialização a partir de métricas de desempenho computacional e de qualidade de predição dos modelos treinados.

4. Avaliar quantitativa e qualitativamente os resultados experimentais.

1.2 Relevância

O uso de RNPs tem se tornado cada vez mais comum em diversas áreas, desde reconhecimento de imagens até processamento de linguagem natural. Entretanto, o sucesso do treinamento de RNPs depende de diversos fatores, incluindo o uso de funções de inicialização de pesos apropriadas. Por essa razão, o estudo sobre o impacto dessas funções no desempenho de RNPs é um tema de grande relevância na área de aprendizado de máquina. Este trabalho, ao avaliar as diferenças de desempenho em RNPs evoluídas pelo algoritmo CoDeepNEAT a partir de diferentes funções de inicialização de pesos, contribui para o avanço do conhecimento sobre a utilização de algoritmos evolutivos na otimização de RNPs.

É importante destacar que a combinação de NeuroEvolução e funções de inicialização de pesos ainda é uma área pouco explorada na literatura. A maioria das pesquisas envolvendo NeuroEvolução tem como foco a utilização de outras técnicas, como redes neurais convolucionais e redes neurais recorrentes. Portanto, é essencial investigar como diferentes funções de inicialização de pesos afetam o desempenho de redes neurais profundas evoluídas pelo algoritmo CoDeepNEAT, uma vez que isso pode trazer novas contribuições e inspirações para a comunidade científica, além de possibilitar o desenvolvimento de novas técnicas em NeuroEvolução.

Outro ponto relevante a ser considerado é que as funções de inicialização de pesos têm um grande impacto na estabilidade e na velocidade de convergência do treinamento de redes neurais profundas. A escolha de uma função de inicialização de pesos adequada pode reduzir o tempo necessário para treinar uma rede neural profunda, além de ajudar a evitar problemas como o desaparecimento ou explosão do gradiente. Por essa razão, a

pesquisa desenvolvida na atual dissertação tem o potencial de contribuir para o avanço da área de Deep Learning, colocaborando para o desenvolvimento de modelos mais eficientes e precisos em diferentes áreas.

Por fim, é importante ressaltar que o estudo sobre funções de inicialização de pesos em redes neurais profundas tem implicações teóricas e práticas significativas. Cabe ressaltar que os resultados obtidos na atual pesquisa podem ter um grande impacto prático. Como essas áreas estão em constante evolução e desenvolvimento, os resultados obtidos com a atual pesquisa podem ser aplicados para melhorar a precisão e a eficiência de modelos já existentes, bem como para o desenvolvimento de novas soluções em Deep Learning. Em resumo, a pesquisa realizada pode trazer importantes contribuições para a comunidade científica e para o desenvolvimento de aplicações em diferentes áreas do conhecimento.

1.3 Organização do texto

Este trabalho está organizado da seguinte maneira: O Capítulo 1, Introdução, apresenta o objetivo e a relevância do trabalho. O Capítulo 2, Trabalhos relacionados, discute pesquisas focadas na atualização dos pesos e trabalhos que abordam a inicialização de pesos em redes neurais. No Capítulo 3, Computação Evolutiva, é oferecida uma visão geral dos algoritmos evolutivos e neuroevolutivos, incluindo NEAT, DeepNEAT e CoDeepNEAT. O Capítulo 4 descreve os blocos fundamentais de Redes Neurais Convolucionais, como camadas convolucionais, funções de ativação, camadas de pooling, batch normalization e conexões residuais. O Capítulo 5, Funções de Inicialização e Ativação, detalha as principais abordagens para a inicialização de pesos e funções de ativação comuns. O Capítulo 6, Proposta de trabalho, apresenta as baselines, parâmetros para experimentos de curto, médio e longo prazo e a implementação do CoDeepNEAT. O Capítulo 7, Resultados, discute os resultados obtidos nos experimentos de curto, médio e longo prazo, assim como a convergência. Por fim, o Capítulo 8, Considerações finais, aborda as conclusões, contribuições e trabalhos futuros. Além disso, o texto conta com três apêndices que apresentam detalhes sobre as melhores redes nos experimentos de longo prazo, melhores indivíduos ao término de cada processo evolutivo para os experimentos de médio prazo e um exemplo de cálculo da função de aptidão (fitness) no CoDeepNEAT.

Capítulo 2

Trabalhos relacionados

Muitos trabalhos, ao aplicarem os conceitos de Computação Evolutiva, focam na busca por estruturas topológicas ideais para o problema-alvo, em detrimento de melhores maneiras de considerar a atualização dos pesos dos indivíduos (redes neurais) durante o processo evolutivo. Neste caso, elencamos aqui uma série de pesquisas focadas em investigar novas maneiras de codificar, inicializar e atualizar os pesos em algoritmos NeuroEvolutivos.

2.1 Trabalhos com foco na atualização dos pesos, mas não suas inicializações

Focando na atualização de pesos durante o processo evolutivo com NeuroEvolução, Koutnik et al. (2010) criaram um novo método de codificar as matrizes de pesos utilizando coeficientes de Fourier. Essa abordagem explora as regularidades espaciais na matriz para reduzir a dimensionalidade da representação, ignorando os coeficientes de alta frequência, como é feito na compressão de imagem com perdas. Essa representação não apenas produz continuidade, mas também permite que a complexidade da matriz de pesos seja controlada pelo número de coeficientes. A codificação no domínio da frequência também significa que o tamanho do genoma[1] é independente do tamanho da rede que ele gera. Portanto, as redes podem ser dimensionadas para problemas de alta dimensão, como os de visão, uma vez que relativamente poucos coeficientes podem codificar matrizes de peso complexas de tamanho arbitrário.

Todas as redes neurais utilizadas em (Koutnik et al., 2010) foram Redes Neurais Convolucionais totalmente conectadas (FRNN–*Fully connected Recurrent Neural Network*). Uma FRNN consiste em três matrizes de peso: (i) uma matriz de entradas $n \times n$, (ii) uma matriz recorrente $n \times n$, e (iii) um vetor $\mathbf{t}$ de *bias* de tamanho n. Essas três matrizes são combinadas em uma matriz $n \times (n+i+1)$, que é e codificada indiretamente usando $c \leq N$

[1]Em computação evolutiva, o genoma é a codificação de um indivíduo. No caso específico de NEP, um genoma codifica as camadas e as conexões de uma RNP.

coeficientes, onde N é o número total de pesos da rede. As figuras 2.1 e 2.2 ilustram a relação entre os coeficientes e os pesos para uma hipotética matriz 4×6–uma rede neural com 4 neurônios, sendo que cada um possui 6 pesos. O lado esquerdo de ambas figuras mostra duas codificações de matriz de peso que usam diferentes números de coeficientes $C_1, C_2, ..., C_c$. De um modo geral, o coeficiente C_i é considerado mais significativo (associado a uma frequência na uql baixa) do que C_j , se $i < j$. O lado direito das figuras mostra as matrizes de peso geradas pela aplicação da transformada discreta do cosseno inversa aos coeficientes. No primeiro caso (Figura 2.1), todos os 24 coeficientes são usados, de modo que qualquer matriz de peso 4×6 possível possa ser representada. A matriz de peso particular mostrada foi gerada a partir de coeficientes aleatórios. No segundo caso (Figura 2.2), cada C_i tem o mesmo valor da Figura 2.1, mas o conjunto completo foi truncado para apenas os quatro coeficientes mais significativos. Quanto mais coeficientes, mais informações de alta frequência são potencialmente expressas na matriz de peso.

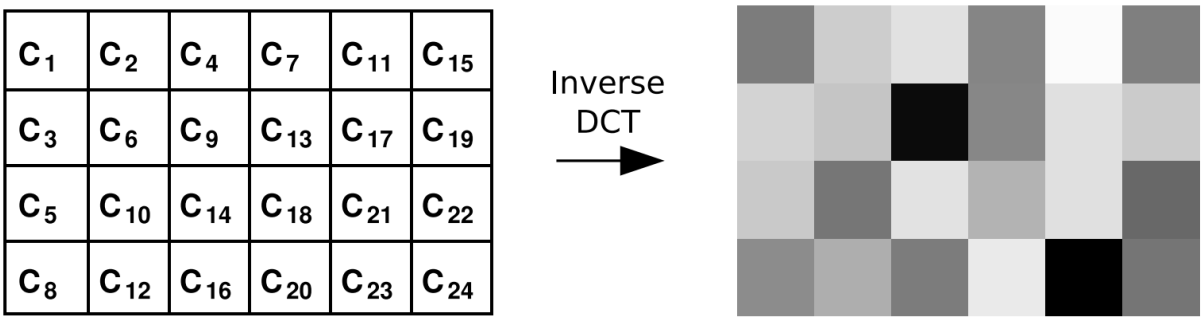

Figura 2.1: Representação da codificação dos pesos proposta por Koutnik et al. (2010). Neste caso, todos os coeficientes são usados. Fonte: (Koutnik et al., 2010)

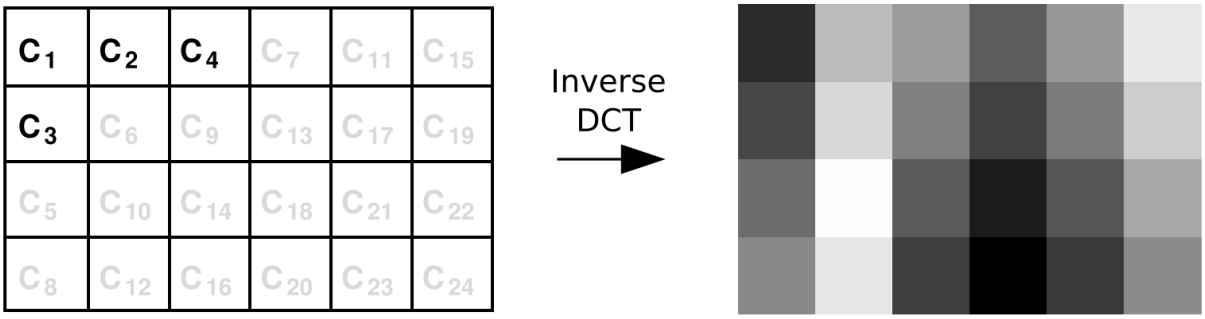

Figura 2.2: Representação da codificação dos pesos proposta por Koutnik et al. (2010). Neste caso, o conjunto completo foi truncado para apenas os quatro coeficientes mais significativos. Fonte: (Koutnik et al., 2010).

Para comprovar a eficácia de sua abordagem, Koutnik et al. (2010) utilizaram três problemas de *benchmark*: *pole balancing*, *ball throwing* e *octopus-arm control*.

Pole balancing (Figura 2.1) é uma referência padrão para sistemas de aprendizagem. A versão mais básica consiste em simular uma única haste articulada a um carrinho que se

desloca ao longo de um trecho finito de trilhos. Uma força deve ser aplicada à haste para que ela seja mantida em pé enquanto o carrinho é deslocado. Ao adicionar uma segunda haste ao lado da primeira, a tarefa se torna mais desafiadora. Uma segundo extensão consiste em realizar a tarefa apenas com informação da posição do carrinho e ângulo das hastes, sem a informação sobre a velocidade do carrinho, o que torna o problema não markoviano. A tarefa é considerada resolvida se as hastes permanecerem equilibradas ao longo de 100.000 iterações.

Na tarefa de *ball throwing* (Figura 2.1), o objetivo é balançar um braço artificial com uma articulação, aplicando um torque à essa articulação e, em seguida, soltar uma bola precisamente no momento certo, de modo que ela seja arremessada o mais longe possível. O sistema dinâmico braço-bola é descrito por

$$(\dot{\theta}, \dot{\omega}) = \left(\omega, -c \cdot \omega - \frac{g \cdot sin(\theta)}{l} + \frac{T}{m \cdot l^2}\right), \tag{2.1}$$

no qual θ é o ângulo do braço, ω é a velocidade angular, $c = 2{,}5s^{-1}$ é a constante de fricção, $l = 2$ m o tamanho do braço, $g = 9{,}81$ ms^{-2}, a massa é dada por $m = 0{,}1$ kg, T é o torque aplicado ($T_{max} = [-5\text{Nm}, 5\text{Nm}]$) e $\dot{\theta}$ e $\dot{\omega}$ representam derivadas parciais. O controlador recebe como entrada (θ, ω) em cada etapa e responde com uma quantidade de torque a ser aplicado.

Por fim, *octopus-arm control* (Figura 2.1) consiste em manipular um objeto composto por n compartimentos flutuando em um ambiente aquático 2D. Cada compartimento tem um volume constante e contém três músculos controláveis (dorsal, transversal e ventral). O estado de um compartimento é descrito pelas coordenadas de dois de seus cantos ((x_1, y_1) e (x_2, y_2)), mais suas velocidades correspondentes para x e y. Juntamente com a rotação da base do braço, o braço possui $8n+2$ variáveis de estado e $3n+2$ variáveis de controle. O objetivo da tarefa é atingir a posição final com a ponta do braço, partindo de três posições iniciais diferentes, contraindo os músculos apropriados a cada passo de 1 segundo do tempo simulado. Embora as posições iniciais 2 e 3, na Figura 2.1, pareçam simétricas, elas são bem diferentes devido à gravidade.

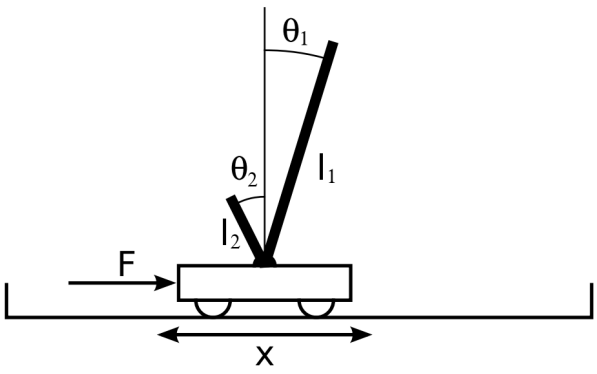

Figura 2.3: Ilustração do problema de *Pole balancing*. Fonte: (Koutnik et al., 2010)

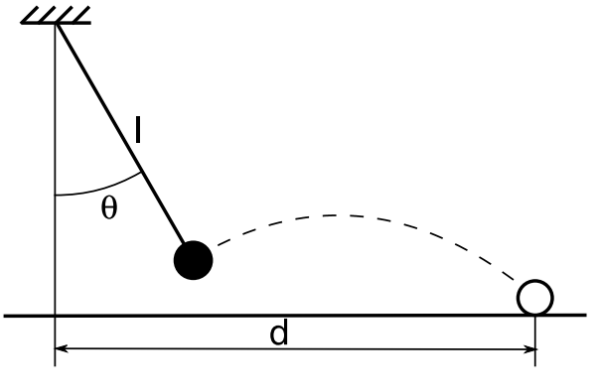

Figura 2.4: Ilustração do problema de *Ball throwing*. Fonte: (Koutnik et al., 2010)

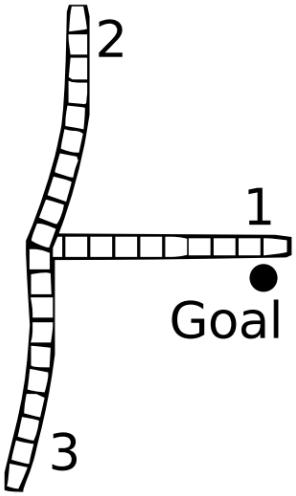

Figura 2.5: Ilustração do problema de *Octopus-arm control*. Fonte: (Koutnik et al., 2010)

Em outro trabalho, Togelius et al. (2008) apresentam um algoritmo chamado *memetic climber*, um algoritmo de busca simples que aprende a topologia e os pesos das redes neurais em diferentes escalas de tempo. Quando aplicado ao problema de controle de aprendizagem para uma tarefa de corrida simulada com entradas cuidadosamente selecionadas para a rede neural, o *memetic climber* supera um algoritmo padrão de *hill-climber*.

Em termos simples, Togelius et al. (2008) decidiram evoluir a estrutura e os pesos de uma rede neural, onde a busca global no espaço de topologia é intercalada com a busca local no espaço de peso. Em outras palavras, depois de alterar uma topologia, almeja-se tentar encontrar uma boa combinação de pesos antes de decidir se o algoritmo deve manter a nova topologia ou reverter para a antiga. O objetivo pretendido com essa abordagem era produzir uma família de algoritmos meméticos (Moscato et al., 1989) que pudessem inicialmente aprender mais lentamente do que os métodos que buscam apenas pesos, mas, em última instância, alcançar maior acurácia evitando ótimos locais induzidos pela topologia.

A Computação Memética é um ramo de pesquisa computacional que considera estruturas complexas como a combinação de agentes simples e uma solução local (*meme*) (Dutta e Mahanand, 2022), cujas interações evolutivas levam a complexos inteligentes capazes de resolver problemas. A pedra fundamental deste assunto tem sido o conceito de algoritmos meméticos, ou seja, uma classe de algoritmos de otimização cuja estrutura é caracterizada por uma estrutura evolutiva e uma lista de componentes de busca local (Neri e Cotta, 2012).

Em todos os experimentos em (Dutta e Mahanand, 2022), foi usado como modelo principal uma MLP com uma camada oculta, na qual cada conexão neural possui uma variável booleana que determina se está ligada ou desligada. A rede como um todo é assim definida por n números reais denotando os pesos das conexões e n booleanos denotando quais conexões estão ativas. Os algoritmos de busca operam nas redes por meio de dois operadores de mutação: mutação de peso e mutação de topologia. A mutação de peso adiciona valores extraídos de uma distribuição gaussiana com média 0 e desvio padrão 0,1 para todos os pesos sorteados; pesos que estão atualmente marcados como desativados também podem ser sorteados. A mutação da topologia consiste em ligar/desligar conexões que estejam atualmente desligadas/ligadas com probabilidade $(0, 0.05)$.

O algoritmo original de *hill-climber* (Algoritmo 1) é equivalente a uma estratégia de evolução $(1 + 1)$, que é tido como um dos algoritmos evolutivos mais básicos, com uma população de apenas um pai e um filho, e as mutação consistem apenas em uma pequena alteração nos valores dados por um parâmetro amostrado de uma distribuição normal (Droste et al., 2002; Glasmachers, 2020; Agudo, 2021). A diferença é o que o *hill-climber* conta com uma população de um indivíduo (o campeão) e, a cada atualização (geração), avalia a aptidão (*fitness*) do campeão, gera um novo indivíduo (o competidor) copiando o campeão, muta este novo indivíduo, e avalia a aptidão do mesmo. Se a aptidão do competidor for maior ou igual ao do campeão, o campeão é substituído pelo competidor, caso contrário o competidor é descartado e o campeão permanece.

Algorithm 1: $Hill - Climber(n)$

1. campeão ← Inicializar()
2. $fitness_{camp.}$ ← Avaliar(campeão)

for $i = l$ *até* n *faça* **do**

3. competidor ← campeão
4. MutarPeso(competidor)
5. $fitness_{conmpetidor}$ ← Avaliar(competidor)

if $fitness_{competidor} \geq fitness_{camp.}$ *então* **then**

6. campeão ← competidor

end if

end for

Já no novo algoritmo proposto, *memetic-climber* (Algoritmo 2), a cada geração, um competidor é gerado, conforme no algoritmo original, e então é aplicada a mutação de

Algorithm 2: $Memetic - Climber(n, m)$

```
1. campeão ← Inicializar()
2. fitness_camp. ← Avaliar(campeão)
for i = l até n faça do
    3. competidor ← campeão
    4. MutarTopologia(competidor)
    for j = l até m faça do
        5. fitness_conmpetidor ← Avaliar(competidor)
        6. subcompetidor ← competidor
        7. MutarPeso(subcompetidor)
        8. fitness_subcompetidor ← Avaliar(subcompetidor)
        if fitness_subcompetidor ≥ fitness_competidor then
            9. competidor ← subcompetidor
        end if
    end for
    10. fitness_conmpetidor ← Avaliar(competidor)
    if fitness_competidor ≥ fitness_camp. then
        11. campeão ← competidor
    end if
end for
```

topologia, o que normalmente causa uma grande queda na aptidão. O Algoritmo 1 é então aplicado para as iterações internas (passos 5 a 9) a fim de encontrar pesos melhores para a topologia modificada.

Ambos Algoritmos, 1 e 2, foram aplicados por Togelius et al. (2008) na competição de *car racing*, organizada pela IEEE CEC, onde o agente inteligente recebe como entrada a imagem de uma pista de corrida e oferece como saída os parâmetros de controle do carro a cada medida pré-determinada de tempo.

Por fim, Okada et al. (2012) propuseram representar os pesos das redes neurais não como escalares, mas como intervalos. Em vez de evoluir a topologia das arquiteturas, os autores focaram apenas na atualização dos pesos. A rede neural, treinada para previsão de funções senoidais, possui camadas sequenciais com os pesos representados por um limite superior e inferior, assim com o vetor de *bias*. Essa rede neural recebe como entrada um vetor $\vec{x}$ de valores reais e calcula a saída, que é uma camada única, como um vetor $\vec{y}$ também de valores reais. A função aproximada foi uma senoidal da forma

$$\hat{y} = 0.2\text{sen}(2\pi x) + 0.1x^2 + [0.3, 0.6]. \tag{2.2}$$

A Equação 2.2 está em notação americana para melhor entendimento, enquanto a Figura 2.6 representa a mesma com seus limite superiores e inferiores. As Figuras 2.7 e 2.8 representam o desempenho da rede neural antes e depois do treino, respectivamente.

A rede utilizada em (Okada et al., 2012) trata esse intervalo para cada peso como genótipos arquivados num vetor $V = (V_1, V_2, ..., V_D)$, no qual V_i é um intervalo e $D = nm + 2m + l$, uma vez que a RN usada possui nm pesos e $m + l$ *bias*. Cada V_i pode ser

expresso pelo seus limites inferior e superior. No treinamento e na iferência, durante a etapa de *feedforward*, para cada conexão é escolhido aleatoriamente um valor do intervalo V_i correspondente ao peso em questão.

Por não utilizar o algoritmo do gradiente descendente, a atualização dos pesos em (Okada et al., 2012) é feita no momento em que um indivíduo é construído para compor a próxima geração.

2.2 Trabalhos com foco na inicialização de pesos

Considerando especificamente o problema de inicialização de pesos, Desell (2018) propôs uma técnica baseada em epigenética para inicializar os pesos de uma rede neural que irá compor uma nova geração. De maneira ampla, a epigenética faz referência aos mecanismos baseados na cromatina importantes na regulação da expressão gênica que não envolvem alterações na sequência de DNA. A epigenômica, por sua vez, se refere às modificações epigenéticas em um tipo de célula em um tempo específico. Recentemente, a epigenética foi considerada o epicentro da biomedicina moderna, devido ao estudo da hereditariedade não relacionada à sequência de DNA que pode ajudar a explicar a relação entre a origem genética de um indivíduo, do meio ambiente, envelhecimento e de uma doença (Bird, 2007; Leite e Costa, 2017).

No cenário da NeuroEvolução, a inicialização epigenética (também chamada de Lamarckiana) dos pesos iniciais (genoma) de um novo indivíduo consiste em herdá-los dos genomas dos indivíduos que o geraram, em contraste com uma inicialização aleatória (Real et al., 2017; Lyu et al., 2020). Assim, ao invés de reinicializarmos aleatoriamente todos os pesos de uma nova geração, os seus indivíduos recebem pesos já treinados por herdar estruturas topológicas dos melhores indivíduos da geração anterior. Para conseguir isso, Desell (2018) utilizou o algoritmo neuroevolutivo *Evolutionary eXploration of Augmenting Convolutional Topologies* (EXACT). Proposto previamente em (Desell, 2017), o EXACT é uma extensão do algoritmo NeuroEvolution of Augmenting Topologies (NEAT) (Stanley e Miikkulainen, 2002), um dos principais algoritmos de NeuroEvolução, abordado com mais afinco na Seção 3.3.2. Diferentemente do NEAT, o EXACT foi projetado para evoluir redes neurais com camadas convolucionais.

A geração da primeira população no EXACT começa com a criação de uma arquitetura mínima, consistindo apenas de um nó de entrada e um nó de saída (ou um nó de saída para cada classe, se necessário). A imagem de entrada é enviada como o genoma da CNN para a primeira requisição de trabalho. Este genoma é inserido na população com aptidão de ∞, denotando que ainda não foi avaliado. Posteriormente, cópias são feitas desse indivíduo inicial e mutadas, para popular o restante dessa população. Depois que a população atinge um tamanho especificado pelo usuário por meio da inserção de mutações recém-geradas, os indivíduos são avaliados e selecionados para mutação e cruzamento. As operações de

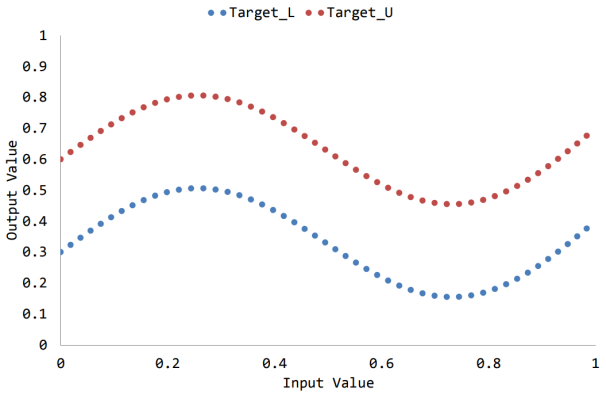

Figura 2.6: Ilustração da função apresenta na Equação 2.2. Fonte: (Okada et al., 2012)

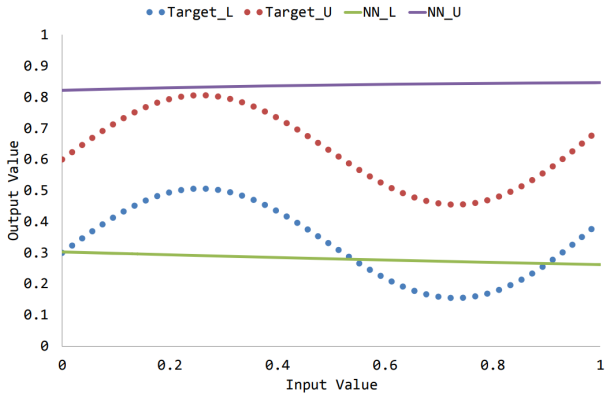

Figura 2.7: Saída da rede usada por Okada et al. (2012) para aproximar a Equação 2.2, antes do treino. Fonte: (Okada et al., 2012)

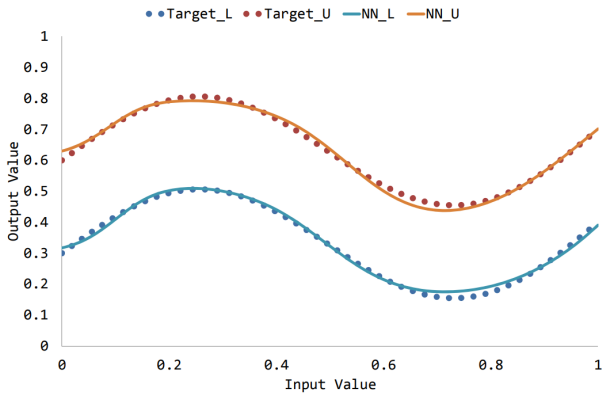

Figura 2.8: Saída da rede usada por Okada et al. (2012) para aproximar a Equação 2.2, após o treino. Fonte: (Okada et al., 2012)

mutação incluem adicionar ou retirar conexões, habilitar ou desativar conexões, e alterar o tamanho dos nós.

Como *benchmark*, o algoritmo EXCAT foi testado na base de dados MNIST, na qual o melhor indivíduo obteve 99,46% de acurácia. Ao todo, Desell (2018) evoluiu mais de 225.000 redes neurais, e contou com mais de 3.500 voluntários que seviram como *hosts* para executar os experimentos.

Lyu et al. (2020) compararam o desempenho do algoritmo NeuroEvolutivo *Evolutionary eXploration of Augmenting Memory Models* (EXAMM), baseado em gradiente descendente para redes neurais recorrentes, utilizando quatro funções distintas para inicializar os pesos das redes neurais: Glorot/Xavier Glorot e Bengio (2010), He/Kaiming He et al. (2015), Distribuição Uniforme Aleatória e a inicialização epigenética – como em (Desell, 2018). O EXAMM evolui redes neurais recorrentes por meio de uma série de operações de mutação e cruzamento. As mutações nas conexões podem dividir, adicionar ou retirar uma conexão recorrente. As mutações a nível de neurônios podem acrescentar ou remover neurônios. Ao adicionar um novo neurônio, há um conjunto pré-definido de possibilidades: unidades Δ-RNN (Ororbia II et al., 2017), unidades recorrentes fechadas (GRUs) (Chung et al., 2014), células de memória de longo prazo (LSTMs) (Hochreiter e Schmidhuber, 1997), unidades com portas mínimas (MGUs) (Zhou et al., 2016) e neurônios de portão de atualização (UGRNNs) (Collins et al., 2016). Esse leque de variedade garante que sejam selecionadas unidades de memória recorrente que tragam o melhor desempenho. Com o EXAMM também é possível criar conexões recorrentes profundas, o que permite que a rede use informações diretas da entrada e de outras camadas.

Para testar o desempenho, 0 EXAMM foi aplicada à previsão de séries temporais com duas bases de dados do mundo real. Como *hardware* foram utilizados 2.304 cores de processamento (Intel Xeon Gold 6150 CPU 2.70GHz) com um total de 24 TB de RAM. Dentre os artigos pesquisados, este foi o único trabalho que mencionou diretamente o problema de *exploding* e *vanishing gradients*. O trabalho não cita o tempo de processamento necessário.

Capítulo 3

Computação Evolutiva

Neste capítulo, são feitas as introduções teóricas dos assuntos pertinentes à pesquisa proposta. Dentro desse contexto, faz-se necessário compreender que a verdadeira grande mudança na área de NeuroEvolução surge com o Algoritmo NEAT, em 2022, e posteriormente desenvolvido sob várias outras formas, em especial, DeepNEAT e CoDeepNEAT, sendo esta último o principal algoritmo utilizado.

3.1 Computação Evolutiva

A Computação Evolutiva (CE) é uma área da Computação que utiliza modelos computacionais dos processos de evolução e seleção para resolver problemas de otimização. Os conceitos e mecanismos da evolução darwiniana e da seleção natural são codificados em algoritmos evolutivos (AEs) e usados para resolver problemas em variados campos da Engenharia e da Ciência (Papavasileiou et al., 2020).

É importante ressaltar que AEs representam uma subárea dentro de CE porque utilizam conceitos e princípios baseados na evolução biológica para resolver problemas computacionais. Esses algoritmos são inspirados em processos evolutivos naturais, como seleção natural, reprodução, mutação e recombinação genética, para criar uma população de soluções candidatas a um determinado problema, que são avaliadas e evoluídas ao longo de gerações. A CE, por sua vez, busca soluções para problemas complexos de otimização e Aprendizado de Máquina por meio de simulação de processos evolutivos não necessariamente biológicos. Além dos AEs, outras técnicas de CE incluem programação genética, sistemas imunológicos artificiais, algoritmos de enxame e redes neurais evolutivas.

Embora as origens da Computação Evolutiva possam ser rastreadas até meados da década de 1950 (a citar, por exemplo, os trabalhos influentes de Bremermann et al. (1962), Friedberg (1958), Friedberg et al. (1959), Box (1957) e outros), o campo permaneceu relativamente desconhecido para a comunidade científica mais ampla por quase três décadas. Isso se deveu em grande parte às limitaçõesdas plataformas computacionais disponíveis na época, mas também devido a algumas deficiências metodológicas dessas abordagens

iniciais (Fogel, 2006; Kicinger et al., 2005). O trabalho fundamental de Holland (1962), Rechenberg (1965), Schwefel (1968) e Fogel (1962) serviu para mudar lentamente esse quadro durante a década de 1970, e atualmente observamos um aumento notável no número de publicações (Kicinger et al., 2005; Eiben e Smith, 2015; Rada, 2008) e conferências neste campo, uma demonstração clara desse assunto.

A forte semelhança com processos biológicos, bem como suas aplicações iniciais para modelagem de sistemas adaptativos complexos, influenciou a terminologia usada pelos pesquisadores da CE. Ela se baseia muito na Genética, na Teoria da Evolução e na Biologia Celular. Assim, uma solução candidata para um problema é chamada *indivíduo*, enquanto um conjunto inteiro (ou mais precisamente um superconjunto) de soluções candidatas é chamado *população*. Para alguns domínios de problemas, uma população pode ser dividida em várias subpopulações. A representação real (codificação) de um indivíduo é chamada *genoma* ou *cromossomo*. Cada genoma consiste em uma sequência de genes, ou seja, atributos que descrevem um indivíduo. Um valor de um gene é chamado *alelo*. Quando soluções individuais são modificadas para produzir novas soluções candidatas, dá-se o nome de *reprodução* e a nova solução candidata é chamada *descendente* ou *filho*. Durante a avaliação de uma solução candidata, ela recebe uma nota chamada aptidão (*fitness*, que indica a qualidade da solução no contexto de um determinado problema). Quando a população atual é substituída por descendentes, a nova população é chamada *nova geração*. Finalmente, para todo o processo de busca por uma solução melhor dá-se o nome de *evolução*.

3.2 Algoritmos Evolutivos

Algoritmos Evolutivos (AEs) são uma família de algoritmos dentro de CE que simulam a evolução de estruturas individuais por processos inter-relacionados de seleção, reprodução e variação. Existe uma variedade de AEs que foram propostos e estudados, em que todos eles compartilham um conjunto comum de suposições subjacentes, mas diferem na estratégia de criação a ser usada e na representação na qual os AEs operam.

Historicamente, três AEs principais foram desenvolvidos: (i) estratégias de evolução (EE) (Rechenberg, 1965; Schwefel, 1965), (ii) programação evolutiva (PE) (Fogel, 1998) e (iii) algoritmos genéticos (AGs) (Holland, 1992). O quarto mais popular AE, a programação genética (PG) (Koza, 1994), tem sido usado para resolver uma série de tarefas computacionais reais (Luke e Panait, 2006; Aaltonen et al., 2009).

Do ponto de vista da Engenharia, CE pode ser entendido como um processo de busca e otimização no qual uma população de soluções sofre um processo de mudanças graduais. Esse processo depende da aptidão (uma medida formal do desempenho percebido) das soluções individuais definidas pelo ambiente (função objetivo).

Um AE canônico consiste nas seguintes etapas:

Algorithm 3: Algoritmo Evolutivo Canônico

1. Inicializar população.
2. Avaliar todos os membros da população.

while *Condição não for alcançada* **do**

3. Selecionar indivíduos na população para popularem a próxima geração
4. Criar novos indivíduos através dos operadoras de variação
5. Avaliar os novos indivíduos
6. Substituir alguns/todos indivíduos na nova população com todos os indivíduos

end while

Antes que um processo evolutivo real comece, uma população inicial de indivíduos (soluções) é criada. Tradicionalmente, a população inicial é criada aleatoriamente, mas várias outras técnicas de inicialização também têm sido usadas (por exemplo, a partir de um conjunto de soluções previamente conhecidas ou assumidas arbitrariamente). Em seguida, cada indivíduo da população inicial é avaliado e recebe um valor de aptidão. Usando os valores de aptidão, o mecanismo de seleção escolhe um subconjunto da população atual como pais para criar novos indivíduos. Uma vez selecionado o conjunto de pais, os novos indivíduos são criados copiando-os e aplicando operadores de variação (mutação).

Uma forma comum de seleção dos pais é a classificação linear (Whitley et al., 1989; Greffenstette e Baker, 1989) em que os indivíduos são primeiro classificados em ordem crescente de acordo com seus valores de aptidão. Cada indivíduo é então selecionado com uma probabilidade baseada em alguma função linear de sua classificação. Outra estratégia de seleção popular é uma seleção de torneio. Nesta estratégia, um conjunto de n indivíduos é escolhido aleatoriamente da população. Cada um dos indivíduos no conjunto é selecionado independentemente e pode ser que o mesmo indivíduo seja selecionado várias vezes. Em seguida, um indivíduo do conjunto com maior valor de aptidão é selecionado para formar a nova população. Este procedimento é repetido quantas vezes forem necessárias para criar uma população inteiramente nova ou um subconjunto da mesma.

Finalmente, a seleção por truncamento escolhe apenas uma certa proporção dos melhores indivíduos da população. Essa estratégia é mais popular na comunidade ES, onde são usados em dois tipos básicos: (l, k) e $(l + k)$ (Schwefel, 1977). No primeiro caso, a seleção opera apenas na população de filhos, enquanto no segundo caso ela seleciona indivíduos de uma população conjunta de pais e filhos. Os dois operadores de variação mais populares são a mutação e a recombinação (*crossover*). A mutação atua em um único indivíduo e funciona aplicando alguma variação a um ou mais genes no cromossomo do indivíduo (semelhante a um operador de variação usado em outros mecanismos de pesquisa, como subida de colina ou recozimento simulado). A recombinação, por outro lado, opera em múltiplos indivíduos (geralmente dois) e combina partes desses indivíduos para criar novos. Os indivíduos recém-criados são avaliados e recebem valores de aptidão.

Então, todo ou apenas um subconjunto da população atual é substituído por esses novos indivíduos. Se toda a população for substituída pelos novos indivíduos, o algoritmo é chamado de AE geracional. Por outro lado, se apenas um subconjunto da população original for substituído, o algoritmo é chamado de AE de estado estacionário. As etapas 3 a 6 do AE canônico, definido anteriormente no Algoritmo 3, são executadas até que um critério de parada assumido seja atendido, que geralmente é definido como um número arbitrário de gerações ou avaliações da função de aptidão.

3.3 Algoritmos NeuroEvolutivos

Algoritmos Neuroevolutivos são uma classe de técnicas que combinam a eficácia das Redes Neurais com a inspiração biológica dos Algoritmos Evolutivos. Ao explorar a evolução natural como um meio de otimização, esses métodos oferecem uma abordagem alternativa e eficiente para treinar e melhorar redes neurais, abordando aspectos que muitas vezes são desafiadores para métodos baseados em gradiente descendente.

3.3.1 Visão geral

Uma abordagem alternativa, que se inspira no processo biológico que criou o cérebro humano, é treinar RNs com Algoritmos Evolutivos. Esse campo é chamado de NeuroEvolução (ou Neuroevolução) (NE) e permite recursos importantes que normalmente não estão disponíveis para abordagens baseadas em gradiente descendente. Esses recursos para RNs incluem otimizar: (i) seus blocos de construção (por exemplo, funções de ativação), (ii) hiperparâmetros (por exemplo, taxas de aprendizado), (iii) arquiteturas (por exemplo, o número de neurônios por camada, quantas camadas existem e quais camadas se conectam a quais), e, até mesmo, (iv) as próprias regras de aprendizagem (Stanley et al., 2019).

Nas primeiras abordagens de NeuroEvolução uma topologia é escolhida para evoluir redes neurais antes do início do experimento. Normalmente, a topologia dessa rede é uma única camada oculta de neurônios, com cada neurônio oculto conectado a todas as entradas e saídas da rede. A evolução, nesse caso, buscava otimizar o espaço de soluções criado pela conexão dos pesos dessa topologia totalmente conectada, permitindo a reprodução de redes de alto desempenho (Stanley e Miikkulainen, 2002). O espaço desses pesos é explorado através do cruzamento de vetores de peso da rede e da mutação dos mesmos. Assim, o objetivo da Neuroevolução com uma topologia fixa é otimizar os pesos que determinam a funcionalidade de uma rede. No entanto, os pesos, como parâmetros treináveis, não são o único aspecto em redes neurais que contribuem para esse ou aquele comportamento. A topologia das redes neurais, ou estrutura, também afeta sua funcionalidade (Stanley et al., 2019).

3.3.2 NEAT

O algoritmo *NeuroEvolution of Augmenting Topologies* (NEAT), desenvolvido por Stanley e Miikkulainen (2002) durante Doutorado na Universidade de Texas, em Austin, é considerado um marco na Neuroevolução e na própria Computação Evolutiva por ser o primeiro algoritmo a considerar a evolução simultânea de uma topologia com seus pesos. Além disso, no NEAT as topologias se tornam cada vez mais complexas à medida que se tornam mais otimizadas.

No NEAT, os genomas são representações lineares da conectividade da rede, que codificam grafos. Cada genoma inclui uma lista de genes de nós e genes de conexão. Um gene de nó representa um neurônio, que pode ser de entrada, oculto ou de saída. Os genes de conexão especificam qual é o nó de entrada e saída em cada camada, bem como o peso da conexão, se a a conexão está ativa ou não (uma variável booleana) e um indicador denominado número de inovação, que permite encontrar topologias similares e facilitar o processo de *crossover*. O *crossover* entre indivíduos se dá na mistura do genoma, nesse caso, no compartilhamento de determinadas estruturas topológicas representadas em grafos como arestas e vértices. As Figuras 3.1 e 3.2 contêm exemplos de genoma e da operação *crossover* no NEAT, respectivamente.

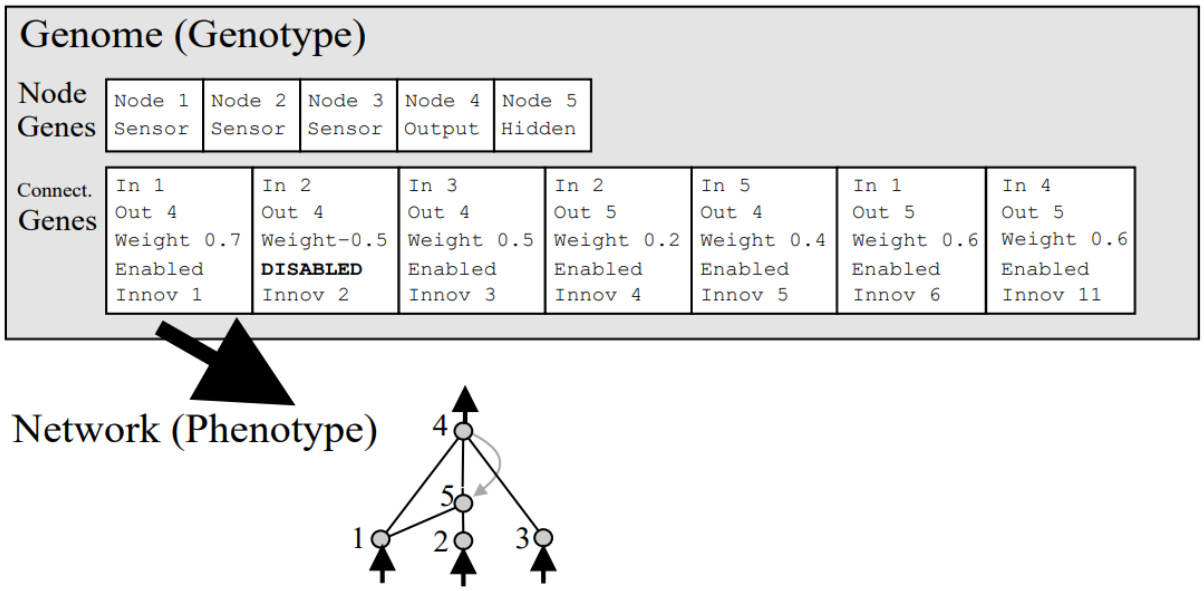

Figura 3.1: Um exemplo de mapeamento de genótipo para fenótipo. Existem três nós de entrada, um oculto e um nó de saída, e sete definições de conexão, uma das quais é recorrente. O segundo gene está desativado, então a conexão que ele especifica (entre os nós 2 e 4) não é expressa no fenótipo. Fonte: (Stanley e Miikkulainen, 2002).

A etapa de mutação pode alterar os pesos e as estruturas da rede. Os pesos mudam como em qualquer sistema NeuroEvolutivo que não considere a utilização do gradiente descendente, isto é, de forma aleatória, através da adição ou subtração de valores dentro de

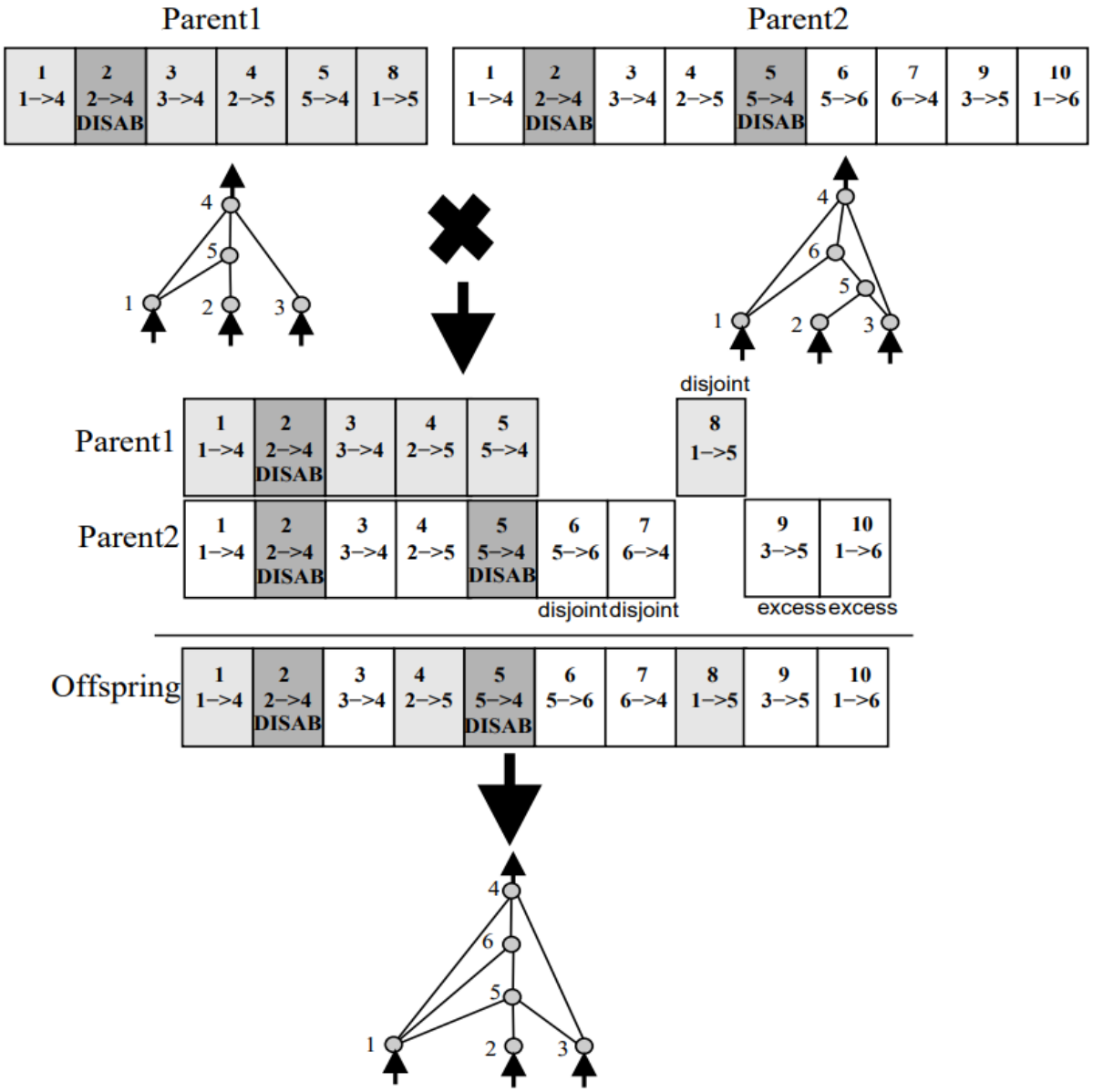

Figura 3.2: Combinação de genomas para diferentes topologias de rede usando números de inovação. Fonte: (Stanley e Miikkulainen, 2002).

um limite previamente estabelecido. As conexões podem ser alteradas entre as gerações. É por meio da mutação que os genomas no NEAT aumentarão gradualmente e resultarão em redes de tamanhos variados. Consequentemente, ao adicionar novos genes à população, e realizar o *crossover*, os genomas que representam diferentes estruturas formam uma população variada de topologias.

As mutações estruturais ocorrem de duas maneiras, conforme a Figura 3.3. Cada mutação expande o tamanho do genoma adicionando genes. Na mutação de adição de conexão, um único novo gene de conexão com um peso aleatório é adicionado conectando dois nós não conectados anteriormente. Na mutação de adição de nó, uma conexão existente é dividida e um novo nó é colocado onde a conexão antiga costumava estar. A

conexão antiga é desativada e duas novas conexões são adicionadas ao genoma. A nova conexão que leva ao novo nó recebe um peso de 1, e a nova conexão que sai recebe o mesmo peso da conexão antiga. Este método de adição de é usado para minimizar o efeito inicial da mutação. A nova não linearidade na conexã provoca uma leve modificação na função representada pela rede neural. Essa alteração permite que novos nós sejam imediatamente integrados à estrutura da rede, evitando a adição de elementos incomuns ou desconexos. Dessa forma, a evolução da rede ocorre de maneira mais eficiente, sem necessidade de ajustes posteriores para acomodar tais estruturas peculiares.

Na mutação, os genomas no NEAT ficarão gradualmente maiores. Isso resultará em genomas de tamanhos variados, às vezes com diferentes conexões nas mesmas posições. As inúmeras topologias diferentes e combinações de peso é um resultado inevitável de permitir que os genomas cresçam sem limites.

Sempre que um novo gene aparece (através de mutação estrutural), um número de inovação global é incrementado e atribuído a esse gene. Os números de inovação representam, portanto, uma cronologia do aparecimento de cada gene no sistema. Como exemplo, digamos que as duas mutações na Figura 3.3 ocorreram uma após a outra no sistema. O novo gene de conexão criado na primeira mutação recebe o número 7, e os dois novos genes de conexão adicionados durante a nova mutação do nó recebem os números 8 e 9, respectivamente. No futuro, sempre que esses genomas forem combinados, a descendência herdará o mesmo números de inovação em cada gene; números de inovação nunca são alterados. Assim, a origem histórica de cada gene do sistema é conhecida ao longo da evolução.

As marcações históricas dão ao NEAT uma nova e poderosa capacidade. O sistema agora sabe exatamente quais genes combinam com quais (Figura 3.2). Ao cruzar, os genes em ambos os genomas com os mesmos números de inovação são alinhados. Esses genes são chamados de genes correspondentes. Os genes que não combinam são chamados de *disjoints* ou em *excess*, dependendo se ocorrem dentro ou fora da faixa dos números de inovação do outro pai. Eles representam uma estrutura que não está presente no outro genoma. Ao compor a próxima geração, os genes são escolhidos aleatoriamente várias vezes entre qualquer um dos pais em genes correspondentes, enquanto todos os genes *disjoints* ou *excess* são sempre incluídos do pai mais apto. Dessa forma, as marcações históricas permitem que o NEAT realize o cruzamento usando genomas lineares sem a necessidade de análises topológicas mais complexas.

Mas adicionar uma nova estrutura inicialmente faz com que a aptidão de uma rede diminua, mesmo que essa nova estrutura seja vista como uma inovação. Isso ocorre porque as estruturas menores otimizam mais rapidamente do que as estruturas maiores, e a adição de nós e conexões geralmente inicialmente diminui a adequação da rede. Consequentemente, as estruturas recentemente criadas têm pouca esperança de sobreviver mais de uma geração, embora as inovações que elas representam possam ser cruciais para

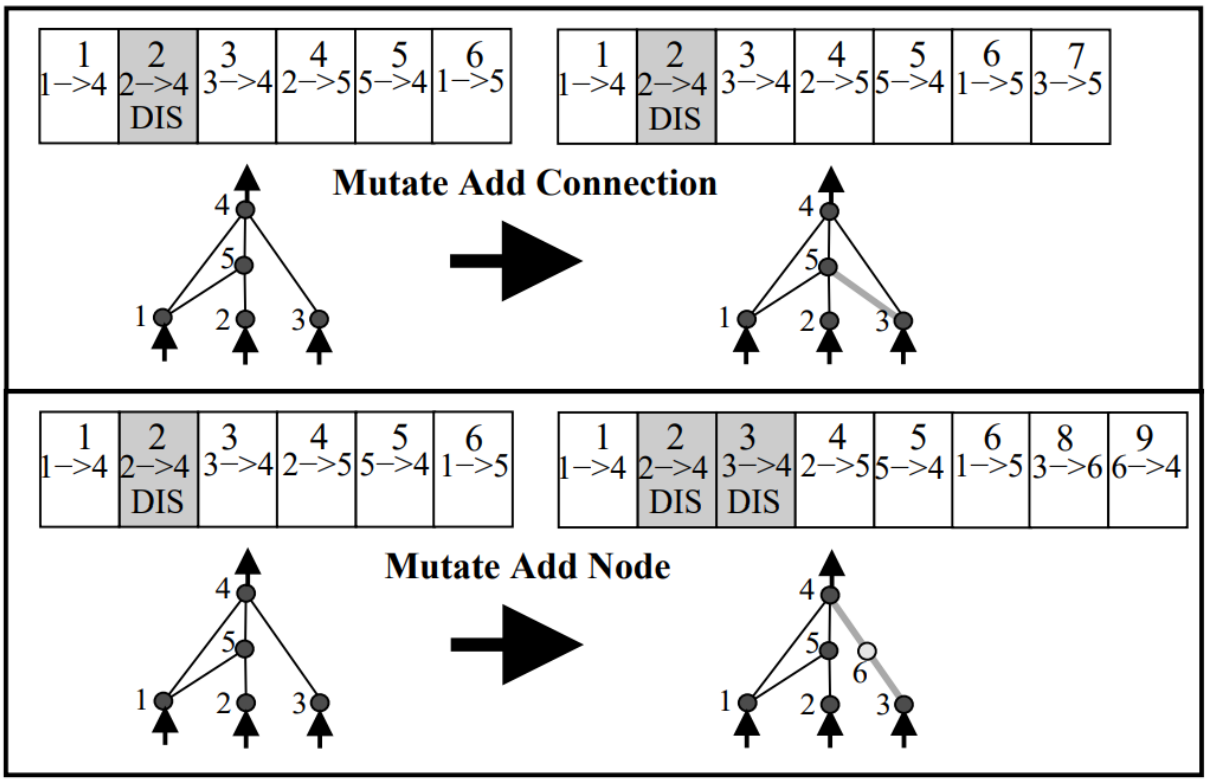

Figura 3.3: Representação dos dois tipos de mutação estrutural no NEAT. Fonte: (Stanley e Miikkulainen, 2002).

resolver a tarefa no longo prazo.

O NEAT soluciona esse problema ao introduzir o conceito de especiação (no original, *speciation*), dividindo a população em espécies — de maneira aleatória na primeira geração — de tal forma que topologias semelhantes são categorizadas na mesma espécie. Com inúmeras topologias e combinações de pesos diferentes, um resultado inevitável é permitir que os genomas cresçam sem limites, mas devido à especiação, estruturas úteis sobrevivem mesmo que sejam inicialmente prejudiciais.

Selecionar topologias parecidas é um problema de correspondência topológica. No entanto, as marcações históricas oferecem uma solução eficiente aqui também. O número de genes em excesso e disjuntos entre um par de genomas é uma medida natural de sua distância de compatibilidade. Quanto mais disjuntos dois genomas são, menos história evolutiva eles compartilham e, portanto, menos compatíveis eles são. Portanto, podemos medir a distância de compatibilidade δ de diferentes estruturas no NEAT como uma combinação linear simples do número de genes em excesso (E) e disjuntos (D), bem como as diferenças médias de peso de genes correspondentes (W), incluindo genes desativados:

$$\delta = \frac{Ec_1}{N} + \frac{Dc_2}{N} + c_3 W. \tag{3.1}$$

Os coeficientes c_1, c_2 e c_3 permitem ajustar a importância dos três fatores. O fator

N é o número de genes no genoma maior e normaliza δ de acordo com o tamanho do genoma. Assim, o NEAT não apenas desempenha a função de otimização na evolução, mas também atua como uma função de complexificação (no original, *complexifying*) das soluções candidatas, permitindo que as arquiteturas se tornam cada vez mais complexas e profundas ao mesmo tempo em que se tornam melhores.

É devido às características de complexificação, de especiação e de marcadores de inovação, que o NEAT é reconhecido como o principal algoritmo NeuroEvolutivo mesmo vinte anos após a sua proposição, sendo utilizado tanto para treinar agentes inteligentes em jogos simples e complexos — como Snake e Super Mario World (Stanley, 2016; Achim, 2020; Eggers, 2020; Bling, 2015), respectivamente —, quanto para calcular a massa de um *quark top* no colisor Tevatron (Aaltonen et al., 2009; Hirschbühl, 2008; Heinson, 2010; Stanley, 2016).

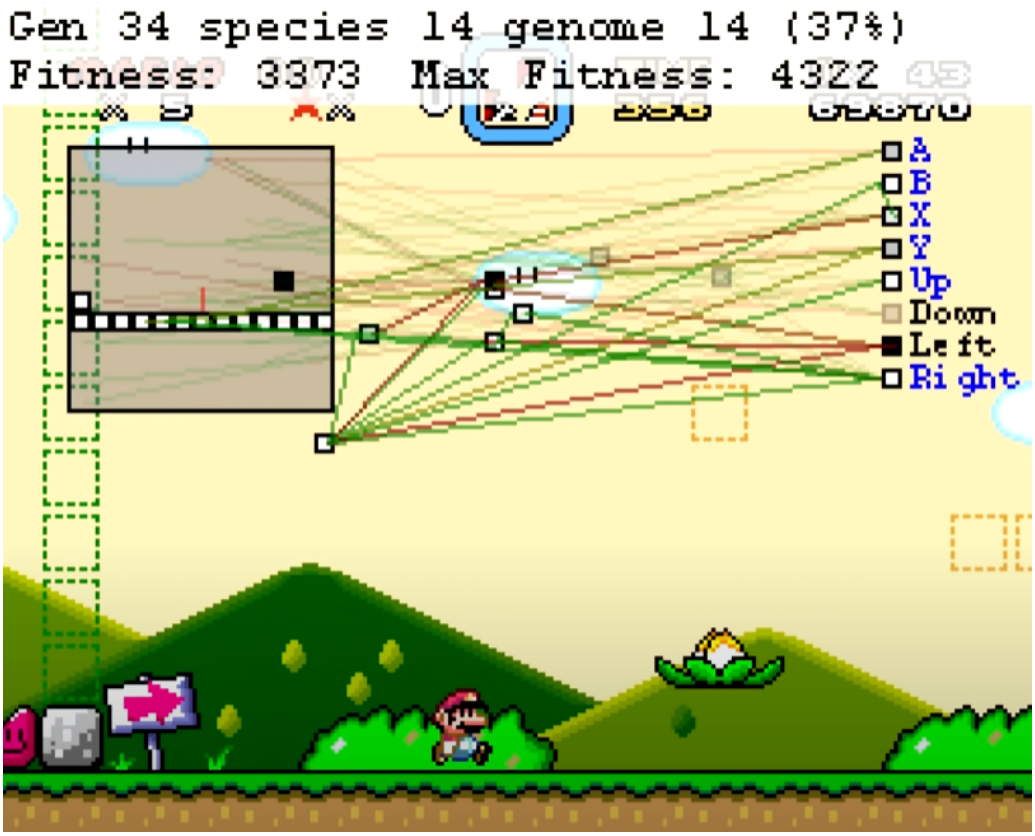

Figura 3.4: Um agente inteligente evoluído com o NEAT jogando *Super Mario World*. Fonte: (Bling, 2015).

3.3.3 DeepNEAT

O DeepNEAT é uma extensão do método de evolução de topologia do NEAT focado em RNPs (Miikkulainen et al., 2019). Ele segue o mesmo processo fundamental do NEAT: primeiro, uma população de cromossomos de baixa complexidade é criada, onde cada

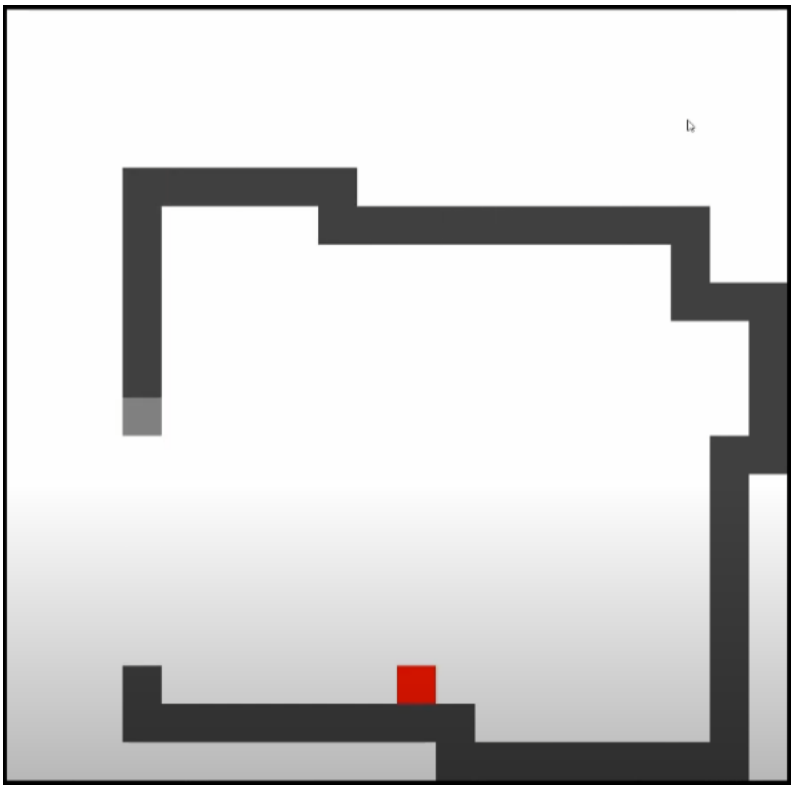

Figura 3.5: Um agente inteligente evoluído com o NEAT para jogar o popular jogo *Snake*. Fonte: (Eggers, 2020).

indivíduo é representado por um grafo. Ao longo das gerações, novas estruturas de nós e arestas são adicionadas ao grafo de forma incremental através de mutações. Durante o *crossover* marcações históricas são usadas para determinar como os genes de dois cromossomos podem ser alinhados.

A população é dividida em espécies (ou subpopulações) com base em uma métrica de similaridade e cada espécie cresce proporcionalmente à sua aptidão enquanto a evolução ocorre separadamente nas outras espécies. O DeepNEAT difere do NEAT pelo fato de que cada nó do cromossomo não representa um neurônio, mas sim uma camada de uma RNP (Miikkulainen et al., 2019). No DeepNEAT, por tanto, surge uma nova perspectiva sobre hiperparâmetros evolutivos e neurais: quantidade e tipo de nós/camadas. Esses hiperparâmetros determinam o tipo de camada (como convolucional, totalmente conectada ou recorrente) e as propriedades dessa camada (como número de neurônios, tamanho do kernel e função de ativação). As arestas de um cromossomo não são mais marcadas com pesos; em vez disso, elas simplesmente indicam como os nós (camadas) estão conectados entre si. Para construir uma RNP a partir de um cromossomo DeepNEAT basta percorrer o grafo cromossômico, substituindo cada nó pela camada correspondente. O cromossomo também codifica um conjunto de hiperparâmetros globais aplicáveis à toda rede, como a taxa de aprendizado, a taxa de espera ("paciência") usada em *early stopping*, dentre outros. Para calcular a aptidão, cada cromossomo é convertido em uma RNP. Essas redes

são então treinadas com um número fixo de épocas. Após o treinamento, uma métrica que indica o desempenho da rede é retornada ao DeepNEAT e atribuída ao cromossomo correspondente na população.

Embora o DeepNEAT possa ser usado para evoluir redes muito profundas, as estruturas resultantes geralmente são complexas e sem nexo. Elas contrastam com arquiteturas típicas que utilizam a repetição de componentes básicos como redes convolucionais canônicas ou, até mesmo, conexões residuais.

3.3.4 CoDeepNEAT

O algoritmo *coevolution of Deep NeuroEvolution of Augmenting Topologies* (CoDeepNEAT) é a continuação direta do algoritmo DeepNEAT (Miikkulainen et al., 2019). Conforme dito por (Papavasileiou et al., 2020), o CoDeepNEAT se destaca como uma das técnicas mais inovadoras de NeuroEvolução por incorporar o uso do gradiente descendente, uma vez que normalmente os algoritmos de NeuroEvolução não utilizam a mesma. Assim como no DeepNEAT, pedaços de camadas no CoDeepNEAT são evoluídos, mas agora tratados sob módulos, que posteriormente são mesclados para criar um *blueprint*, que por sua vez é usado para criar várias arquiteturas neurais (Figura 3.6). Dessa forma, o CoDeepNEAT trabalha com duas populações: módulos e *blueprints*.

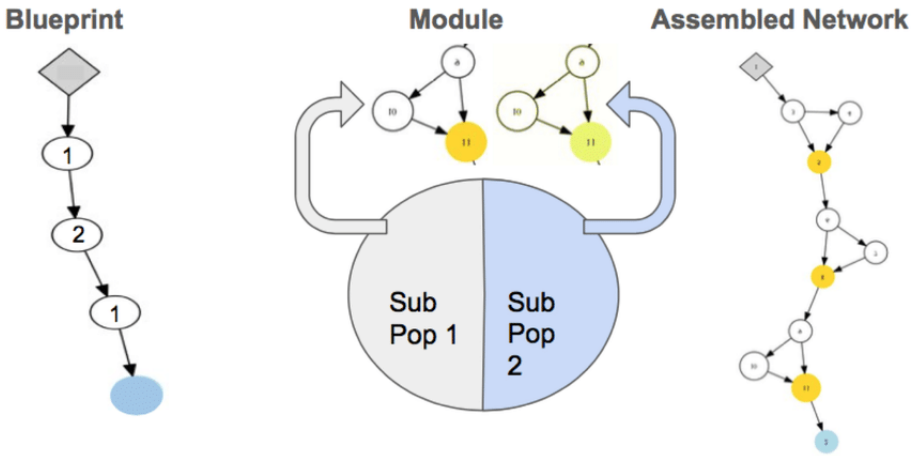

Figura 3.6: Estruturação de uma rede neural a partir de um indivíduo *blueprint* e um conjunto de indivíduos de módulos. O número dentro de cada nó no *blueprint* representa as espécies na população dos módulos. As subpopulações na população de módulos acima ilustra, em parte, o processo de especiação. Fonte: (Miikkulainen et al., 2019).

O algoritmo trabalha com ambas as populações, mas evoluindo-as separadamente. O cromossomo *blueprint* é um grafo onde cada nó contém um módulo particular, enquanto cada cromossomo de módulo é um grafo que representa camadas de RNPs (Papavasileiou et al., 2020).

A população inicial de módulos é gerada de acordo com os hiperparâmetros do algoritmo CoDeepNEAT. Um módulo pode conter camadas de convolução, camadas to-

talmente conectadas ou camadas residuais (*skip connections*), e a probabilidade de obter qualquer uma delas é especificada pelo usuário do algoritmo. Se o módulo contém camadas de convolução, seus hiperparâmetros, como o número de filtros e o tamanho do kernel, são escolhidos aleatoriamente dentro de um conjunto de valores possíveis que também devem ser previamente especificadas.

No início de cada geração, um conjunto de *blueprints* é escolhido aleatoriamente da população de *blueprints*. Cada nó em um *blueprint* é substituído por um módulo, que é também escolhido aleatoriamente da sua própria população inicial naquela geração. Isso resulta em um conjunto de redes neurais montadas que compreendem os indivíduos, de fato, dessa geração. Essas redes neurais ainda não foram treinadas e seus pesos devem ser inicializados. Cada indivíduo é então treinado em um pequeno número de épocas e avaliado em uma partição *hold-out* do conjunto de dados de treinamento.

Como o CoDeepNEAT trabalha com duas populações, há dois valores de *fitness* a serem calculados. A *fitness* de cada indivíduo do *blueprint* é a média de uma métrica (*loss*, precisão ou *F1-score* etc.) de todas as redes neurais que foram baseadas a partir desse *blueprint*. Se um determinado *blueprint* não foi escolhido para gerar redes neurais nesta geração, seu valor de *fitness* é zero. Da mesma forma, a *fitness* de cada módulo é a média de todas as redes que empregaram aquele módulo como parte de sua arquitetura naquela geração.

Ao final de cada geração, com exceção da última, novos *blueprints* são gerados a partir do cruzamento (*crossover*) dos melhores indivíduos, produzindo uma nova população de *blueprints*. O mesmo é feito para a população de módulos. Ao longo das gerações, as novas estruturas são adicionadas de forma incremental por meio de mutações aleatórias. Após a última geração ser concluída, o algoritmo está pronto para apresentar a melhor rede neural indicada para a tarefa em particular. Esta pode ser o indivíduo com melhor valor de *fitness* da geração final, ou o melhor indivíduo de todas as demais gerações. Esse indivíduo é então treinado com todo o conjunto de dados por um número determinado de épocas, que geralmente é significativamente maior do que o número de épocas usado para calcular a *fitness* de *blueprints* e módulos durante a evolução. O Apêndice C contém um exemplo prático do cálculo de aptidão durante uma geração do CoDeepNEAT.

Miikkulainen et al. (2019) também introduzem o conceito de *fast learners* enquanto discute os resultados obtidos. Como o algoritmo trabalha com ambas as populações evoluindo separadamente, e o cromossomo *blueprint* é um gráfico onde cada nó contém uma espécie de módulo particular, cada cromossomo de módulo é um grafo que representa camadas de RNPs (Papavasileiou et al., 2020).

Isso é importante porque o processo evolutivo do CoDeepNEAT é enviesado pelo treino parcial dos indivíduos durante as gerações. Na prática, isso significa que a evolução pode ser guiada com/por outros objetivos além da simples precisão, incluindo tempo de treinamento, tempo de execução ou requisitos de memória da rede. Esse detalhe

reforça a necessidade de um estudo mais aprofundado das funções de inicialização de no CoDeepNEAT.

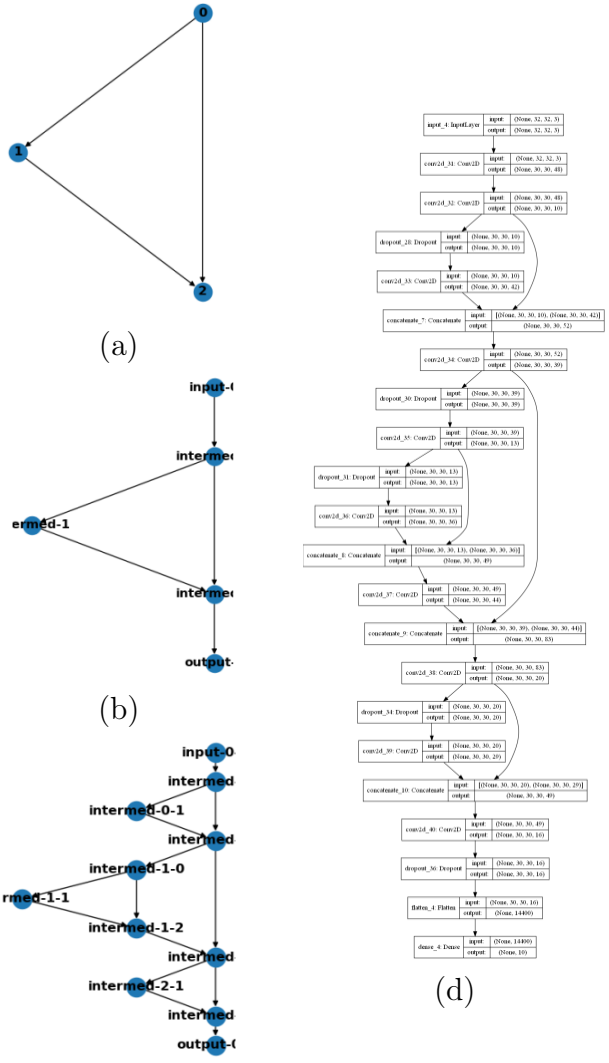

Figura 3.7: (a) Grafo de um módulo estruturando a conexão de três camadas diferentes. (b) Grafo de um *blueprint* estruturando a conexão de três instâncias do módulo (a), e adicionando camadas de entrada e saída. (c) Grafo exemplificando o resultado final do módulo em (a) com a informação do *blueprint* (b). (d) Representação em framework Keras do modelo criado em (c). Fonte: (Bohrer et al., 2020).

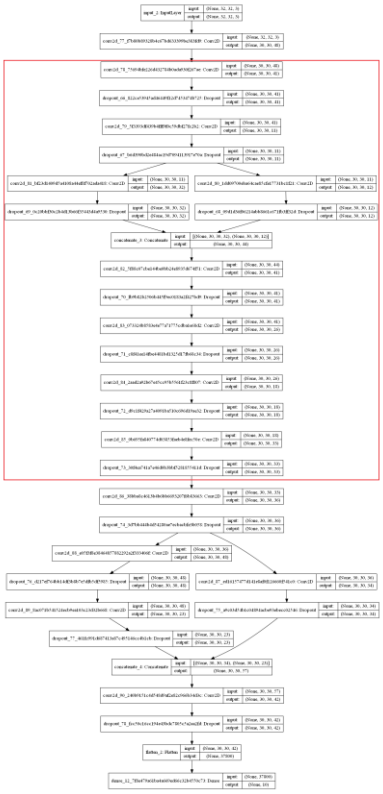

Figura 3.8: Rede Neural geradora 1 (RNG1). Fonte: (Bohrer et al., 2020).

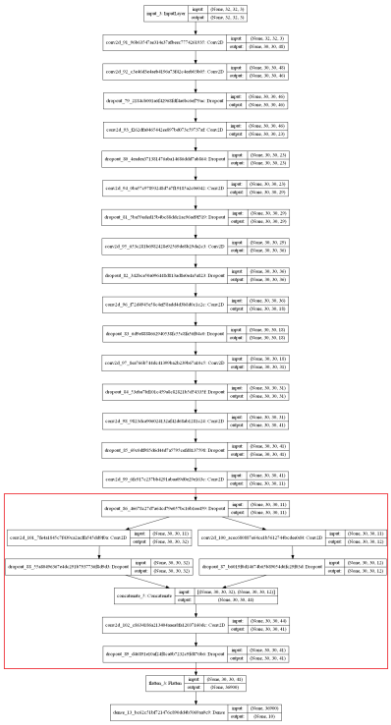

Figura 3.9: Rede Neural geradora 2 (RNG2). Fonte: (Bohrer et al., 2020).

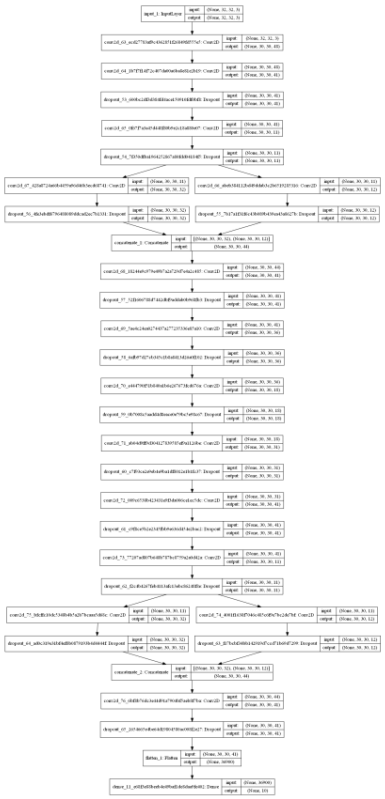

Figura 3.10: Rede Neural descendente do *crossover* entre RNG1 e RNG2. Fonte: (Bohrer et al., 2020).

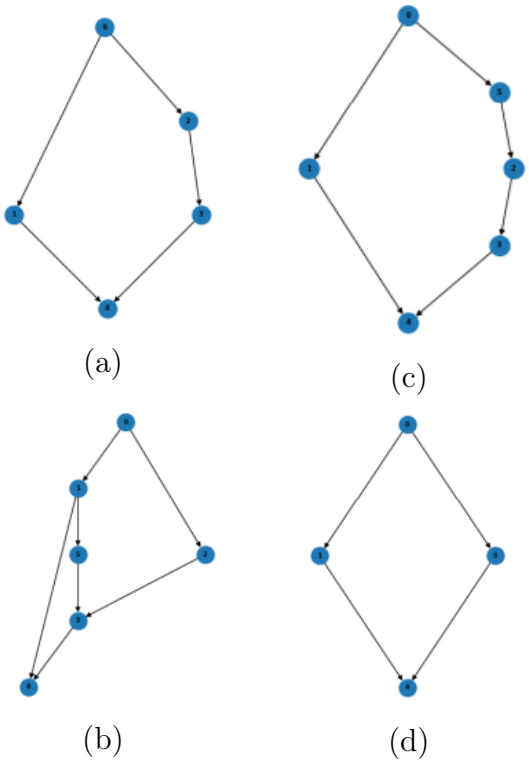

Figura 3.11: (a) *Blueprint* original. (b) Mutação de adição de nó. (c) Mutação de adição de nó. (d) Mutação de remoção de nó. Fonte: (Bohrer et al., 2020).

Algorithm 4: CoDeepNEAT

1. Definir limite superior/inferior para hiperparâmetros topológicos (número de filtros, taxa de *dropout*, *max pooling*, *batch normalization* etc.)

2. Definir limite superior/inferior para hiperparâmetros evolutivos (gerações, tamanho da população size, tamanho da populaçãod e módulos, tamanho da população de *blueprints*, época de treino, taxa de elitismo, taxa de mutação, taxa de *crossover*)

3. Gerar a primeira população de módulos e de *blueprints*

4. Inicializar duas listas de arrays, B e M, onde cada célula representa cada *blueprint* e módulos criados, respectivamente (esses arrays são usados para calcular a função *fitness* de cada população ao final de cada geração)

for *Para cada Geração g* **do**
  for *Para cada Rede Neural montada* **do**
    5. Treinar por e épocas em uma partição Po do conjunto de treino

    6. Estimar a performance P da atual Rede Neural no conjunto de validação V

    7. Guardar o custo P nos arrays B e M, mapeando cada *blueprint* e módulo usados, isto é, concatenando o valor P com o valor prévio armazenado em cada célula representando os módulos e *blueprints* da Rede Neural atual. Por exemplo, se uma Rede Neural N1 usa o *blueprint* B1 e módulos M1, M2 e M3, e obtém um custo P', deve-se concatenar o custo P' dentro da célula correspondente a B, que representa B1, e concatenar P' dentro das células do array M, que representa M1, M2, M3, respectivamente
  end for
  8. Calcular a *fitness* para *blueprints*: para cada célula em B, calcular a média. Selecionar apenas a melhor porção Be de indivíduos para *crossover*

  9. Calcular a *fitness* para módulos: para cada célula em M, calcular a média. Selecionar apenas a melhor porção Be de indivíduos para *crossover*

  10. Crossover: reunir indivíduos selecionados no processo de cálculo de *fitness* de ambas as populações, módulos e *blueprints*, para gerar a próxima população de Redes Neurais

  11. Mutação aleatória nos novos indivíduos
  12. Reinicializar listas B e M
end for
13. Treinar a melhor Rede Neural obtida até a última geração por E épocas utilizando todo o conjunto de treino

14. Avaliar o desempenho no conjunto de teste T

Capítulo 4

Blocos fundamentais de Redes Neurais Convolucionais

Existem vantagens estruturais, representados por blocos estruturais ou fundamentais, em Redes Neurais Convolucionais (RNCs) que as tornam adequadas para capturar características complexas de imagens. Esta é a principal razão pela qual esse gênero de modelos alcança desempenhos superiores em muitas tarefas envolvendo imagens, como classificação, agrupamento, segmentação etc. Todos esses blocos são utilizados pela CoDeepNEAT para construir *blueprints*, módulos e arquiteturas contruídas pelo algoritmo. O Capítulo a seguir apresenta os conceitos desses blocos com maior profundidade pois serão utilizados como parâmetros pelo CoDeepNEAT durante os experimentos propostos.

4.1 Camada Convolucional

Responsável por usar um ou vários filtros (de tamanho geralmente de 3×3, 5×5, 7×7 etc.) para convolucionar a entrada, gerando diferentes mapas de características para processamento posterior. Uma convolução é uma operação na forma

$$\mathcal{F}_j^{l+1} = \sum_i W_{i,j}^l F_i^l + b_j^l, \tag{4.1}$$

onde $\mathcal{F}_j^l$ representa o j-ésimo mapa de característica da l-ésima camada, $W_{i,j}^l$ representa o *kernel* de convolução e b_j^l o viés. Os filtros $W_{i,j}^l$ e o viés b_j^l em cada camada de uma RNC não são fixos, mas podem ser automaticamente aprendidos através do algoritmo de gradiente descendente (Goodfellow et al., 2016; Wu et al., 2018).

4.2 Camada de Função de Ativação

Essa cama transforma a entrada em um mapa de características através de um mapeamento não-linear:

$$\mathcal{F}_j^{l+i} = f(F_j^l), \tag{4.2}$$

onde $f(\cdot)$ é uma função não-linear, como a sigmoide, tangente hiperbólica (tanh), ReLU etc. O mapeamento não linear é uma das camadas mais importantes de uma RNC, uma vez que permite uma extração mais complexa de correlações da entrada.

4.3 Camada de *Pooling*

Reduz a dimensionalidade da entrada do mapa de características, compactando-o:

$$\mathcal{F}_j^{l+i} = pool(F_j^l), \tag{4.3}$$

onde $pool(\cdot)$ denota a função de *pooling*. Geralmente, há dois tipos de função de *pooling* em RNCs: (i) *maximum pooling*, e (ii) *average pooling*. *Maximum pooling* seleciona o valor mais alto em uma região local como saída, enquanto o *average pooling* calcula a média de uma região local como saída. Independente disso, o papel principal de uma camada de pooling é agregar os mapas de características de forma compacta. Ademais, uma grande distância de correlação entre informações de entrada pode ser capturada através do *pooling* do mapa de característica em um tamanho menor (Goodfellow et al., 2016; Wu et al., 2018).

4.4 Camada de *Batch Normalization*

Camada responsável por normalizar cada dado x_i de um *batch* de treino $\mathcal{B}$ em $\mathcal{Y}_i$:

$$\mathcal{Y}_i = \hat{x}_i + \beta, \tag{4.4}$$

onde $\mathcal{Y}_i$ e β são parâmetros de *batch normalization*. Na Eq. (4.4) $\hat{x}_i$ denota:

$$\hat{x}_i = \frac{x_i - \mathbb{E}_{\mathcal{B}}(x_i)}{\sqrt{Var_{\mathcal{B}}(x_i)}} \tag{4.5}$$

Na Equação 4.5, $\mathbb{E}_{\mathcal{B}}(x_i)$ e $Var_{\mathcal{B}}(x_i)$ representam a média e a variância de x_i em termos do *batch* $\mathcal{B}$. O objetivo principal do *batch normalization* é garantir que os dados estão distantes de regiões de saturação. Para fins práticos, uma RNC com *batch normalization* é relativamente não-sensível à inicialização dos pesos e converge mais rápido do que uma RNC sem *batch normalization* (Goodfellow et al., 2016; Wu et al., 2018).

4.5 Conexões Residuais

Quando RNPs são capazes de começar a convergir, um problema de degradação surge: com o aumento da profundidade da rede, o gradiente satura e então degrada rapidamente. Inesperadamente, tal degradação não é causada por]emphoverfitting, e adicionar mais camadas a um modelo adequadamente profundo leva a um maior erro de treinamento (He e Sun, 2015; Srivastava et al., 2015). A degradação (da precisão de treinamento) indica que nem todos os sistemas são igualmente fáceis de otimizar.

Para tratar esse problema, He et al. (2016) propuseram um novo (à época) *framework* de aprendizado profundo residual. Tal componente reformula explicitamente as camadas como funções de aprendizado residuais com referência às entradas da camada, em vez de aprender com funções não-referenciadas. Assim, ao invés de depender exclusivamente das camadas em sequência para mapear diretamente a saída desejada, as conexões residuais proporcionam uma maneira explícita de mapear a entrada e agregar uma nova via de informação ao longo da rede neural.

Formalmente, um bloco residual pode ser definido como:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, W_i) + \mathbf{x} \tag{4.6}$$

onde $\mathbf{x}$ e $\mathbf{y}$ são vetores de entrada e saída, respectivamente, da camada considerada. A função $\mathcal{F}(\mathbf{x}, W_i)$ representa o mapeamento residual a ser aprendido pela RN. A operação $\mathcal{F} + \mathbf{x}$ opera por uma conexão de curto alcance (*shortcut*) e uma adição *element-wise* (He et al., 2016).

Outro fator importante é que conexões residuais não introduzem nenhum parâmetro extra nem complexidade computacional. Isso não é apenas atraente na prática, mas também importante em comparações entre redes sequenciais e residuais. Não à toa, desde seu surgimento, muitas pesquisas, em diversos âmbitos, tomaram como ponto central a utilização de conexões residuais (Drozdzal et al., 2016; Ronneberger et al., 2015; Shafiq e Gu, 2022; Zhang et al., 2017; Khan et al., 2022).

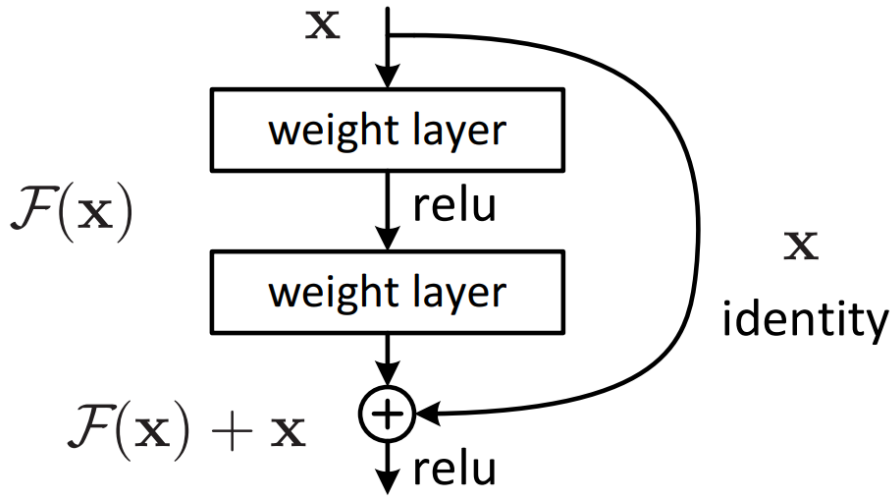

Figura 4.1: Representação de um bloco de aprendizado residual. Fonte: (He et al., 2016).

4.5.1 Importância de conexões residuais na visualização da *loss landscape*

Sabe-se que a função de *loss* tende a ser caótica em arquiteturas muito profundas, o que é desfavorável à treinabilidade e dificulta a generalização (Zhou et al., 2021). As arquiteturas residuais são muito populares em aprendizado profundo (Zhou et al., 2021; Dogan et al., 2022; Hoorali et al., 2022) porque tendem a simplificar esse espaço de busca. A implementação de conexões residuais apresenta um fluxo ininterrupto de gradiente de uma determinada camada para a mais próxima da saída, evitando o problema do *vanishing gradient*. Consequentemente, múltiplas conexões residuais (*skip connections*) são uma alternativa para garantir a reutilização de recursos da mesma dimensionalidade das camadas anteriores. Por outro lado, essas conexões também são úteis para recuperar informações espaciais perdidas durante o *downsampling* e parecem estabilizar atualizações de gradiente em arquiteturas muito profundas, garantindo uma convergência rápida. Tal estrutura é tão importante que foi o principal motivador utilizado por Miikkulainen et al. para introduzir a mutação de conexões entre neurônios no CoDeepNEAT (Miikkulainen et al., 2019; Adaloglou, 2020) original e empregar um número alto de gerações.

Li et al. (2018) apresentaram um método que torna possível visualizar a *loss landscape* de uma função de grande dimensão, demonstrando experimentalmente que a *loss landscape* muda significativamente quando conexões residuais são introduzidas. E apesar de seu impacto aparentemente limitado, múltiplas conexões residuais permitem que a *landscape* resultante seja muito mais suave, conforme ilustrado na Figura 4.2.

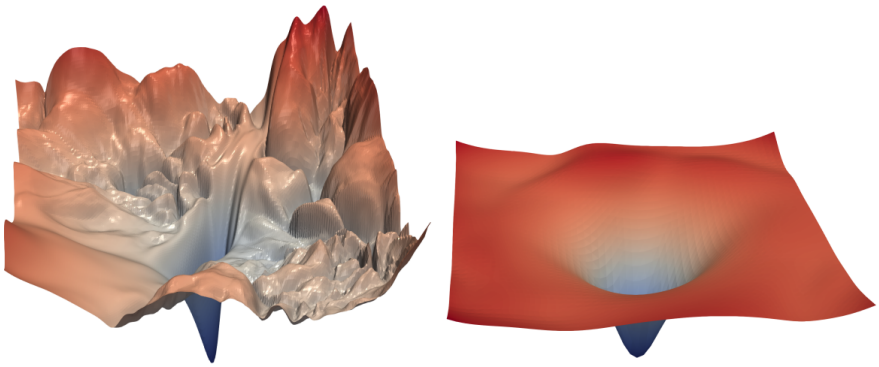

Figura 4.2: *Loss surface* de uma ResNet-56 sem (esquerda) e com (direita) conexões residuais. Fonte: (Li et al., 2018).

À medida que as redes se aprofundam, a *loss landscape* torna-se cada vez mais retorcida (ou menos convexa). Isso torna a rede sensível às condições iniciais e difícil de treinar. No entanto, as conexões residuais evitam a explosão de não-convexidade que ocorre quando as redes se aprofundam. Curiosamente, o efeito de pular conexões parece ser mais importante para redes profundas. Veja a diferença entre os cenários de perda da Resnet-56 (com conexões residuais) e da Resnet-56 (sem conexões residuais). Esse efeito parece se aplicar também a conexões residuais usadas na DenseNet, cujo cenário de perda é muito bem comportado.

De modo geral, é possível extrair dois pontos principais dos resultados do trabalho de Li et al. (2018):

1. **Efeito da profundidade da rede:** a profundidade da rede tem um efeito drástico na *loss landscape* das redes neurais quando conexões residuais não são usadas. A rede ResNet-20 tem uma *landscape* bastante suave dominada por uma região com contornos convexos no centro e nenhuma não convexidade dramática. No entanto, à medida que a profundidade da rede aumenta, a *landscape* das redes sem conexões residuais transita espontaneamente de (quase) convexa para caótica. A Resnet-56 tem não-convexidades drásticas e grandes regiões onde as direções do gradiente não apontam para o minimizador no centro.

2. **Conexões residuais evitam a explosão de não-convexidade:** As conexões residuais têm um efeito dramático na geometria das funções de *loss*, pois permitem que os gradientes fluam mais facilmente pelos modelos de rede neural, evitando o problema de degradação do gradiente. Isso significa que, com as conexões residuais, os modelos podem treinar mais rapidamente e atingir desempenho mais alto, sem sofrer com o desaparecimento do gradiente. Essa melhoria na geometria das funções de loss resulta em modelos mais eficientes e precisos, o que é especialmente relevante em aplicações que exigem alto desempenho e precisão, como em reconhecimento de imagens e processamento de linguagem natural.

Por consequência, o assunto relaciona-se imediatamente com o tema central deste trabalho. O surgimento de estruturas de conexões residuais é um fator a ser também avaliado a partir das funções de inicialização de peso empregadas no decorrer do processo evolutivo do CoDeepNEAT e podem, inclusive, referenciando o Capítulo 3, serem consideradas um fator relevante para candidatos a *fast learners*.

Capítulo 5

Funções de Inicialização e Ativação

5.1 Funções de Inicialização

Parte da dificuldade em treinar modelos de RNPs está em determinar a estratégia de inicialização adequada para os parâmetros do modelo. É bem sabido que inicializações arbitrárias podem desacelerar ou até mesmo impedir completamente o processo de convergência (Mishkin e Matas, 2015). A desaceleração surge porque as inicializações arbitrárias podem resultar em camadas mais profundas recebendo entradas com pequenas variações, o que, por sua vez, diminui o valor do gradiente descendente e retarda o processo geral de convergência.

De forma simples, o problema pode ser definido da seguinte forma: se os pesos em uma RN são inicializados usando amostras de uma distribuição normal, $\mathcal{N}(0, v^2)$, como v^2 deve ser escolhido para garantir que a variância das saídas de as diferentes camadas são aproximadamente as mesmas?

A inicialização do peso é uma área de pesquisa ativa, e vários métodos foram propostos para lidar com o problema da variação de encolhimento nas camadas mais profundas (Mishkin e Matas, 2015; Saxe et al., 2013; Sussillo e Abbott, 2014). Entretanto, dois trabalhos se distacam entre os demais: (Glorot e Bengio, 2010) e (He et al., 2015). Essas pesquisas, por instituírem o padrão na área, serão destrinchadas com mais afinco nas subções a seguir.

5.1.1 Inicialização Glorot (ou Xavier)

Glorot e Bengio (2010) mostraram que, para uma função de ativação linear, o valor ótimo de variância é $\sigma^2 = 1/N$, onde N é o número de nós que alimentam aquela camada. Embora esse método dependa de várias suposições sobre as entradas do modelo, ele funciona extremamente bem em muitos casos e é amplamente utilizado na inicialização de redes neurais até o momento. Esse esquema de inicialização é comumente chamado de *inicialização Xavier* ou *inicialização Glorot.*

A função é idealizada para RNs com funções de ativação simétricas como tangente hiperbólica (*tanh*) e *softsign* (Glorot e Bengio, 2010; Lyu et al., 2020) e extrai os pesos, dentro de uma distribuição normal $\mathcal{N}(0, \sigma^2)$ e uma distribuição uniforme $\mathcal{U}(-a, a)$.

Prova matemática - *Forward Pass*

Conforme dito por Goodfellow et al. (2016), as fórmulas são derivadas da suposição de que a rede a ser aplicada consiste apenas de uma cadeia de multiplicações de matrizes, sem ativações não-lineares.

Para encontrar a prova da Glorot, conforme apresentada por (Hlav, 2022), procuramos por W^i tal que a variância de cada camada z subsequente seja igual, isto é, $\text{Var}[z^i] = \text{Var}[z^{i+1}]$. Considerando que $z^{i+1} = f(z^i W^i + b^i)$, podemos escrever:

$$Var[z_k^{i+1}] = Var[f(z^i W_{.,k}^i) + b_k^i] \tag{5.1}$$

$$= Var[z^i W_{.,k}^i + b_k^i] \tag{5.2}$$

$$= Var\left[\sum_{j=1}^{n^i} z_j^i W_{j,k}^i\right] \tag{5.3}$$

Na Equação 5.2, partindo do pressuposto que a função de ativação usada é ímpar e com derivada $f'(0) = 1$, nós podemos aproximar $f(x) \simeq x$ em 0. Então, $z^i W^i + b^i = 0$ na inicialização, uma vez W^i e b^i partem de distribuições centradas em 0, e z^0, o vetor de entrada para a RN, foi normalizado. Logo, cada camada z^i seguinte terá média de 0.

Na Equação 5.3, aplicamos a propriedade aditiva da variância sob a independência das variáveis, isto é, $Var[X + Y] = Var[X] + Var[Y]$ com $X \perp Y$, e a propridade para constantes na variância ($Var[c] = 0$ e $Var[X + c] = Var[X]$). Por conseguinte:

$$Var\left[\sum_{j=1}^{n^i} z_j^i W_{j,k}^i\right] = \sum_{j=1}^{n^i} Var[z_j^i W_{j,k}^i] \tag{5.4}$$

$$= \sum_{j=1}^{n^i} \mathbb{E}[z_j^i]^2 Var[W_{j,k}^i] + \mathbb{E}[W_{j,k}^i]^2 Var[z_j^i] + Var[W_{j,k}^i] Var[z_j^i] \tag{5.5}$$

Assumindo a independência $z \perp W$ entre o vetor de entrada z e a matriz de peso W (Equação 5.4), resulta que todas as variáveis não correlacionados na inicialização. Sob independência, a variância da soma é a soma das variâncias. Já no passo seguinte, Equação 5.5, analogamente a regra da soma da variância, a variância de produtos independentes é igual ao produto das variâncias mais duas vezes o produto trocado de ambos envolvendo média e variância, isto é, $Var[XY] = Var[X]Var[Y] + \mathbb{E}[X]^2 Var[Y] + \mathbb{E}[X]^2 Var[Y] +$

$\mathbb{E}[Y]^2 Var[X]$. Deste ponto, segue:

$\mathbb{E}[z_j^i] = \mathbb{E}[W_{j,k}^i] = 0$

$$Var[XY] = \sum_{j=1}^{n^i} Var[W_{j,k}^i] Var[z_j^i] \tag{5.6}$$

E $z_j^i, W_{j,k}^i$ são variáveis aleatórias independentes e identicamente distribuídas, logo:

$$Var[XY] = n^i Var[W^i] Var[z^i] \tag{5.7}$$

A Equação 5.6 permite uma simplificação porque os primeiros dois termos podem ser eliminados pelo fato de que tanto z quanto W possuíram média 0. Isto se justifica pelo fato da normalização das entradas e W vir de uma distribuição centrada em 0. No passo seguinte, Equação 5.7, cada termo da soma é o mesmo uma vez que z e W são variáveis independentes e indeticamente distribuídas.

Finalmente, isso nos leva a conclusão que a variância dos pesos W^i é inversamente proporcional ao número de entradas n^i.

$$Var[W^i] = \frac{1}{n^i} \tag{5.8}$$

Prova matemática - *Backward Pass*

Nós precisamos encontrar W^i igual a $Var[\partial L / \partial s^{i+1}]$. A forma matricial da Regra da Cadeia é:

$$\frac{\partial L}{\partial s^i} = \frac{\partial L}{\partial s^{i+1}} W^{(i+1)^T} \nabla f(s^i) \tag{5.9}$$

$$\frac{\partial L}{\partial s_k^i} = \frac{\partial L}{\partial s^{i+1}} W_{.,k}^{1+1} \nabla f'(s_k^i) \tag{5.10}$$

$$= \sum_{j=1}^{n^{i+2}} W_{j,k}^{i+1} \frac{\partial L}{\partial s_j^{i+1}} f'(s_k^i) \tag{5.11}$$

$$Var[\frac{\partial L}{\partial s_k^i}] = Var\left[\sum_{j=1}^{n^{i+2}} W_{j,k}^{i+1} \frac{\partial L}{\partial s_k^{j+1}} f'(s_k^i)\right] \tag{5.12}$$

$$\sum_{j=1}^{n^{i+2}} Var\left[W_{j,k}^{i+1} \frac{\partial L}{\partial s_j^{i+1}}\right] \tag{5.13}$$

Para alcançarmos a Equação 5.13, podemos perceber que na Equação 5.12 a média de s^i é 0 na inicialização. Isso significa que f é linear em 0 com $f'(0) = 1$. Logo, $\mathbb{E}[f'(s^i)] = \mathbb{E}[f'(0)] = 1$. Depois, sabendo que a variância da soma de variáveis independentes é igual

a soma das variâncias, aplicamos essa regra no lado direito da Equação e chegamos na Equação 5.13.

$$\sum_{j=1}^{n^{i+2}} \mathbb{E}\left[W_{j,k}^{i+1}\right]^2 Var\left[\frac{\partial L}{\partial s_j^{i+1}}\right] + \mathbb{E}\left[\frac{\partial L}{\partial s_j^{i+1}}\right]^2 Var[W_{j,k}^{i+1}] + Var[W_{j,k}^{i+1}] Var\left[\frac{\partial L}{\partial s_j^{1+1}}\right] \tag{5.14}$$

A Equação 5.14 deriva da aplicação da regra do produto da variâncoa para variáveis independentes. Consequentemente, a equação simplifica para uma soma individual de produtos de médias e variâncias.

$$\sum_{j=1}^{n^{i+2}} \mathbb{E}\left[W_{j,k}^{i+1}\right]^2 Var\left[\frac{\partial L}{\partial s_j^{i+1}}\right] + \mathbb{E}\left[\frac{\partial L}{\partial s_j^{i+1}}\right]^2 Var[W_{j,k}^{i+1}] + Var[W_{j,k}^{i+1}] Var\left[\frac{\partial L}{\partial s_j^{1+1}}\right] = \sum_{j=1}^{n^{i+2}} Var[W_{j,k}^{i+1}] Var\left[\frac{\partial L}{\partial s_j^{i+1}}\right] \tag{5.15}$$

$$= n^{i+2} Var[W_{j,k}^{i+1}] Var\left[\frac{\partial L}{\partial s^{i+1}}\right] \tag{5.16}$$

Na Equação 5.15 ambos os termos com a média são iguais a 0, restando apenas a soma das variâncias. Já na Equação 5.16 podemos ver que cada termo da soma é o mesmo uma vez que a derivada parcial de L, em relação a s^{i+1}, e e W são independetes e indenticamente distribuídos. Por fim, temos:

$$Var[W^i] = \frac{1}{n^{i+1}} \tag{5.17}$$

Prova matemática - Distribuição de peso

Seguindo as demonstrações acima, podemos concluir:

$$\forall i, n^i Var[W^i] = 1 \tag{5.18}$$

$$\forall i, n^{i+1} Var[W^i] = 1 \tag{5.19}$$

Glorot e Bengio (2010) propuseram a média das variâncias encontradas. A justificativa principal para isso surge do fato que quando RNs tem subsequentes camadas de mesma largura, as duas equações (Equação 5.8 e Equação 5.17) são iguais, isto é, $n^i = n^{i+1}$, implica que o valor médio satisfaz ambas equações, ou seja:

$$\frac{n^i + n^{i+1}}{2} Var[W^i] = 1 \tag{5.20}$$

$$Var[W^i] = \frac{2}{n^i + n^{i+1}} \tag{5.21}$$

Na prática, uma vez que a variância das distribuições é conhecida, os pesos são inicializados com uma distribuição normal $\mathcal{N}(0, \sigma^2)$ ou distribuição uniforme $\mathcal{U}(-a, a)$.

Na Glorot, é fundamental que a distribuição escolhida seja centrada em 0. Se $X \sim \mathcal{N}(0, \sigma^2)$, então $Var[X] = \sigma^2$. Consequentemente, a variância e o desvio padrão podem ser escritos da seguinte forma:

$$\sigma^2 = \frac{2}{n^i + n^{i+1}} \tag{5.22}$$

$$\sigma = \sqrt{\frac{2}{n^i + n^{i+1}}} \tag{5.23}$$

Portanto, podemos concluir que W^i obedece a uma distribuição normal com os coeficientes:

$$W^i = \mathcal{N}\left(0, \frac{2}{n^i + n^{i+1}}\right), \tag{5.24}$$

onde n^i é o tamanho de entrada na camada e n^{i+1} é o tamanho na saída da mesma camada. De outra forma, se $X \sim \mathcal{U}(-a, a)$, podemos utilizar a fórmula da variância de uma variável aleatória que obedece uma distribuição uniforme (Glorot e Bengio, 2010), podemos encontrar os valores limites de a:

$$Var[X] = \frac{1}{6} \cdot 2a^2 = \frac{1}{3}a^2 \tag{5.25}$$

$$\frac{1}{3}a^2 = \frac{2}{n^i + n^{i+1}} \tag{5.26}$$

$$a^2 = \frac{6}{n^i + n^{i+1}} \tag{5.27}$$

Finalmente, podemos concluir que:

$$W^i \sim \mathcal{U}\left(\pm \sqrt{\frac{6}{n^i + n^{i+1}}} \right) \tag{5.28}$$

5.1.2 Inicialização He (ou Kaiming)

Posteriormente, He et al. (2015) argumentaram que a inicialização de Glorot e Bengio (2010) não funcionava bem com a função de ativação ReLU (que não é simétrica e nem não-linear) e, em vez disso, propuseram uma inicialização de $\sigma^2 = 2/N$ (comumente chamada de *inicialização de He* ou *inicialização Kaiming*). Para motivar essa inicialização, eles

forneceram um contra-exemplo de uma rede neural de 30 camadas que converge com a inicialização He, mas não sob a inicialização Xavier.

Usando uma derivação do trabalho de Glorot e Bengio (2010), He et al. (2015) definiram:

$$Var[W^i] = \frac{2}{n^i} \tag{5.29}$$

Consequentemente, pela mesma lógica utilizada nas Subções 5.1.1,5.1.1 e 5.1.1, temos:

$$W^i = \mathcal{N}\left(0, \frac{2}{n^i}\right), \tag{5.30}$$

$$W^i \sim \mathcal{U}\left(\pm\sqrt{\frac{6}{n^i}}\right) \tag{5.31}$$

Prova matemática

A principal diferença, do ponto de vista conceitual, entre as inicializações Glorot e He, é que a última abrange ativações não diferenciáveis em 0, como o caso da ReLU.

Considerando-se uma RNP com M camadas, conforme proposto e demonstrado por (Kumar, 2017), a relação entre as entradas x_m para alguma camada e as entradas x_{m+1} para a camada seguinte é descrita por:

$$y_m(i) = \sum_{j=1}^{j=N} W_m(i,j)x_m(j) = \sum_{j=1}^{j=N} p_{ij} \tag{5.32}$$

E também por:

$$x_{m+1}(i) = g(y_m(i)), \tag{5.33}$$

onde $p_j = W_m(i,j)x_m(j)$, W_m é a matriz de pesos da m-ésima camada, g é a função de ativação não-linear, e N é o número de nós nas camadas ocultas. Os pesos $W_m(i,j)$ são variáveis aleatórias independentes e indenticamente distribuídos com média 0 e variância σ^2.

Usando as recursões nas Equações 5.32 e 5.33, $x_m(k)$ pode ser representado como uma função não-linear dos pesos nas primeiras $(m-1)$ camadas. Uma vez que os pesos na m-ésima camada são independentes dos valores de entrada na primeira camada e os pesos em todas as outras camadas, os pesos na m-ésima camada serão também independentes de qualquer função não-linear dessas quantidades. Logo, $W_m(i,j)$ é independente de $x_m(k)$ para todos os valores de i, j e k. Além do mais, como $W_m(i,j)$ é independente de $x_m(k)$ e $x_m(l)$, $W_m(i,j)$ será também independente de $x_m(k)x_m(l)$.

Calculando as médias nas Equações 5.32 e tendo como proposição o fato de que na primeira iteração $W_m(i,j)$ é independente de $x_m(k)$ para todos os valores de i, j e k, e o

fato de que $\mathbb{E}(W_m(i,j)) = 0$, obtemos:

$$r_m = \mathbb{E}(y_m(i)) = \sum_{j=1}^{j=N} \mathbb{E}(W_m(i,j))\mathbb{E}(x_m(j)) = 0 \tag{5.34}$$

Por esse motivo, $u_m^2 = Var(y_m(i)) = \mathbb{E}(y_m(i)^2) - (\mathbb{E}(y_m(i)))^2 = \mathbb{E}(y_m(i)^2)$. Usando a Equação 5.32,

$$u_m^2 = \mathbb{E}\left(\sum_{j=1}^{j=N} W_m(i,j)x_m(j)\right)^2 \tag{5.35}$$

$$= \sum_{j=1}^{j=N} \mathbb{E}(W_m(i,j)^2 x_m(j)^2) + 2\sum \mathbb{E}(W_m(i,j)x_m(j)W_m(k,l)x_m(l)) \tag{5.36}$$

Pela primeira proposição feita, $W_m(i,j)$ e $W_m(k,l)$ serão independentes entre si e independentes de $x_m(j)x_m(l)$. Usando esses resultados, e o fato de que $\mathbb{E}(W_m(i,j)) = 0$, encontramos:

$$\mathbb{E}(W_m(i,j)x_m(j)W_m(k,l)x_m(l)) = \mathbb{E}(W_m(i,j))\mathbb{E}(W_m(k,l))\mathbb{E}(x_m(j)x_m(l)) = 0 \tag{5.37}$$

Substituindo a Equação 5.16 na Equação 5.36, resulta em:

$$u_m^2 = \sum_{j=1}^{j=N} \mathbb{E}(W_m(i,j)^2 x_m(j)^2) \tag{5.38}$$

$$= \sum_{j=1}^{j=N} \mathbb{E}(W_m(i,j)^2)\mathbb{E}(x_m(j)^2) \tag{5.39}$$

$$= N\sigma^2(s_m^2 + \mu_m^2) \tag{5.40}$$

Partindo da Equação 5.34 e de uma segunda proposição — $y_m(i)$ será aproximadamente normalmente distribuído para todos os valores de m e i — , podemos encontrar a prova para a Equação 5.29 aplicada a ReLU. Para a primeira iteração, $y_m \sim \mathcal{N}(0, u_m^2)$. Como estamos interessados na variância de $x_{m+1}(i) = \max(0,\, y_m(i))$, a média é dada por:

$$\mu_{m+1} = \mathbb{E}(y_m(i)I(y_m(i) > 0)) = \frac{1}{u_m\sqrt{2\pi}} \int_0^\infty x e^{-\frac{x^2}{2u_m^2}} dx = \frac{u_m}{\sqrt{2\pi}} \int_0^\infty e^{-t} dt = \frac{u_m}{\sqrt{2\pi}} \tag{5.41}$$

Similarmente,

$$\mathbb{E}(x_{x+1}(i)^2) = \frac{1}{u_m\sqrt{2\pi}} \int_0^\infty x e^{-\frac{x^2}{2u_m^2}} dx = \left(\frac{1}{2}\right) \frac{1}{u\sqrt{2\pi}} \int_{-\infty}^\infty x e^{-\frac{x^2}{2u^2}} dx = \frac{u_m^2}{2} \tag{5.42}$$

Usando Equação 5.32 e 5.33:

$$s_{m+1}^2 = \mathbb{E}(x_{m+1}(i)^2) - \mu_{m+1}^2 = \frac{u_m^2}{2} - \frac{u_m^2}{2\pi} \approx 0.34 u_m^2 \tag{5.43}$$

Para a variação ser mantida em cada iteração, precisamos $s_{m+1}^2 \approx 1$, o que significa:

$$u_m^2 \approx 3 \tag{5.44}$$

Usando a Equação 5.44 na Equação 5.41 temos que $\mu_{m+1} \approx 0.7$. Consequentemente, por simetria, nós podemos esperar que $\mu_{m+1} \approx \mu_m ... \approx \mu_2 \approx 0.7$. Retornando para a Equação 5.40, temos:

$$3 = N\sigma^2(1 + 0.49) \tag{5.45}$$

Ou, também,

$$\sigma^2 \approx 2/N \tag{5.46}$$

5.2 Funções de Ativação

O desenvolvimento de funções de ativação desempenhou um papel vital na melhoria do desempenho de RNPs. Consequentemente, mais e mais esforços se concentraram no estudo de funções de ativação (Sibi et al., 2013; Ramachandran et al., 2017; Nwankpa et al., 2018; Jagtap et al., 2022). Funções de inicialização cooperam na convergência do modelo, assim como corrigem problemas conhecidos com o gradiente. A escolha da função de ativação tem relação direta, muitas vezes, com o desempenho final de uma RNP.

Embora o foco principal deste trabalho seja especificamente a utilização de funções de inicialização, uma das configurações de experimentos, a serem descritas no capítulo seguinte, foi criada para contemplar a aplicação de mais de uma função de ativação, tangente hiperbólica e ReLU, que serão explicadas mais a fundo a seguuir.

5.2.1 Função Tangente Hiperbólica

A função tangente hiperbólica (tanH) pode ser facilmente definida como a proporção entre as funções seno e cosseno.

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.47}$$

É similar à função sigmoide e pode ser deduzida da mesma:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}, \tag{5.48}$$

onde cada $x \in (-\infty, +\infty)$, $g \in (0, 1)$. Logo:

$$tanh(x) = 2sigmoid(2x) - 1 \tag{5.49}$$

A função da tangente hiperbólica tem valores entre -1 e 1, como visto na Figura 5.1. Ela também é contínua e monotônica, e diferenciável em todos os valores de x. Portanto, as funções tangentes hiperbólicas são mais preferíveis do que as funções sigmóides.

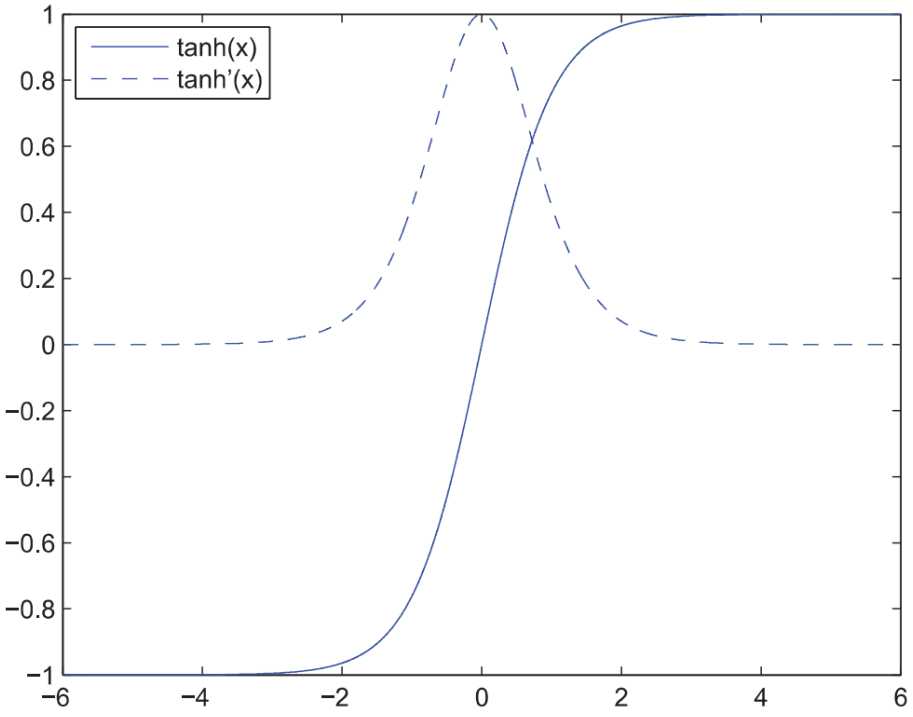

Figura 5.1: Gráfico da função Tangente Hiperbólica e sua derivada.

As redes neurais com funções de ativação tangente hiperbólica, portanto, convergem melhor do que aquelas com função de ativação sigmoide, e têm menor erro de classificação do que aquelas com funções de ativação sigmoide (Glorot e Bengio, 2010; Krizhevsky et al., 2017).

Sua derivada é da forma:

$$tanh'(x) = 2sigmoid'(2x) - 1 = \frac{4e^{-2x}}{(1 + e^{-2x})^2} \tag{5.50}$$

5.2.2 Função ReLU

Lennie (2003), em sua pesquisa em neurociência, descobriu que os neurônios corticais raramente estão em seu regime de saturação máxima e sugere que sua função de ativação pode ser aproximada por um retificador. Ou seja, o modo de operação dos neurônios tem a característica de esparsidade. Mais especificamente, apenas de um a quatro por cento dos neurônios no cérebro podem ser ativados simultaneamente. No entanto, nas redes neurais com funções de ativação sigmoide ou tangente hiperbólica, quase metade das unidades neuronais são ativadas ao mesmo tempo, o que é inconsistente com a realidade na natureza do cerébro. Além disso, ativar mais unidades de neurônios trará mais dificuldades no treinamento de uma rede neural profunda.

As unidades lineares retificadas (do inglês, ReLU), apresentadas pela primeira vez por Nair e Hinton (2010) para máquinas de Boltzmann restritas, ajudam a camada oculta de uma RN a obter uma matriz de saída esparsa, que pode melhorar a eficiência e convergência. Consequentemente, a função ReLU e suas derivadas são as funções de ativação mais populares usadas em RNPs atualmente (Schmidt-Hieber, 2020; Bertoin et al., 2021; DeVore et al., 2021).

A definição da função ReLU é dada por:

$$\text{relu}(x) = \max(0, x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geqslant 0 \end{cases} \tag{5.51}$$

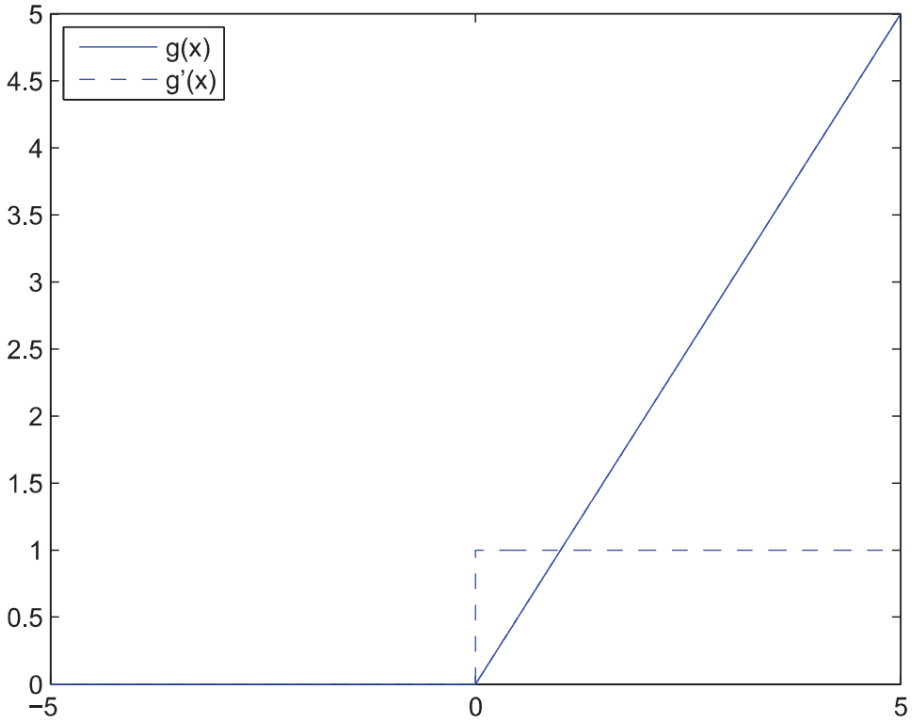

Figura 5.2: Gráfico da função ReLU e sua derivada.

Enquanto sua derivada é definida como:

$$\text{relu}'(x) = \begin{cases} 0, & \text{if } x \geqslant 0 \\ 1, & \text{if } x < 0 \end{cases} \tag{5.52}$$

Capítulo 6

Proposta de trabalho

Neste capítulo, apresenta-se a base da proposta de trabalho para os experimentos realizados, acompanhada de uma breve explanação sobre a implementação do algoritmo utilizado. Formalmente, descrevem-se os hiperparâmetros topológicos empregados por dois grupos distintos de pesquisadores que utilizaram a *CoDeepNEAT*. Estes constituem o primeiro referencial de resultados para os experimentos realizados, sendo considerados os *baselines*. Compara-se também o recurso computacional utilizado e os resultados obtidos nas mesmas bases de dados. Aludem-se aos valores empregados na pesquisa, dividindo os experimentos em três variantes: *Curto Prazo*, *Médio Prazo* e *Longo Prazo*.

6.1 *Baselines*

Experimentos de curto, médio e longo prazo são fundamentais para investigar como o desempenho dessas redes é afetado por diferentes fatores, como a complexidade da tarefa, a quantidade de dados de treinamento e o tipo de função de inicialização de pesos utilizada. Além disso, esses experimentos permitem explorar as características e limitações do algoritmo CoDeepNEAT em diferentes contextos e situações, contribuindo para o avanço do conhecimento na área de NeuroEvolução e aprendizado de máquina em geral.

Os experimentos de curto, médio e longo prazo foram baseados e comparados com dois *baselines*. O primeiro, doravante denominado B1, é o trabalho original que apresentou o CoDeepNEAT (Miikkulainen et al., 2019), e o segundo, B2, é o trabalho de (Bohrer et al., 2020).

O aspecto marcante de B1 é o espaço de busca muito grande do algoritmo coevolutivo, conforme discutido no Capítulo 3, enquanto B2 tem um espaço de busca muito menor. Isso porque o objetivo de Bohrer et al. foi reproduzir os experimentos originais de B1 com recursos limitados; no lugar de várias GPUs, eles usaram uma única CPU com apenas 30 GB de RAM. Tal diferença marcante decorre da disponibilidade de acesso a melhores e mais robustos recursos que a equipe de B1 teve acesso; eles utilizaram um sistema distribuído chamado DarkCycle que, na época, em 2017, utilizava 2.000.000 CPUs

Tabela 6.1: Hiperparâmetros evolutivos e topológicos dos *baselines*. B1 e B2 são *baselines*. As precisões B1 e B2 são relatados em seus artigos originais, respectivamente.

Parâmetros	**B1**	**B2**
Gerações	75	40
População de RNPs	100	10
População de *blueprints*	25	10
População de módulos	45	30
Épocas durante a evolução	8	4
Épocas de treino final	300	40
Quantidade de dados no treino	100%	40%
Filtros	[32, 256]	[16, 48]
Tamanho do kernel	{1, 3}	{1, 3, 5}
Taxa de *dropout*	[0, 0.7]	[0, 0.5]
Max Pooling	{Yes, No}	{No}
Batch Normalization	{No}	{No}
Acurácia MNIST	-	92%
Acurácia CIFAR-10	92.7%	77%

e 5.000 GPUs, espalhadas ao redor do mundo, resultando em um desempenho máximo de 9 PetaFLOPs.

Outra estratégia utilizada por B2 para diminuir o custo computacional foi reduzir a complexidade das redes neurais. Enquanto B1 treinou cada rede neural com todo o conjunto de treino com grandes tamanhos de populações, em B2 o espaço de hiperparâmetros é drasticamente reduzido. Eles evoluíram as redes por apenas 40 gerações, em vez das 70 inicialmente empregadas em B1. Os números de redes neurais, *blueprints* e módulos também foram reduzidos, assim como a complexidade dos módulos (menos filtros, *kernels* maiores e sem *Max Pooling*).

Com um espaço de busca menor, eles também reduziram a quantidade de dados de treinamento. O conjunto de dados CIFAR-10 contém 50.000 de instâncias de treinamento. Enquanto B1 usa todos eles em um esquema de hold-out para treinar e validar as RNPs, B2 emprega apenas 40% desses dados. A partição de teste permaneceu igual em ambos os *baselines*.

6.2 Parametrização experimental

Neste trabalho, adotamos uma abordagem experimental e empírica, realizando experimentos em três diferentes contextos de hiperparâmetros: curto prazo (CP), médio prazo (MP) e longo prazo (LP). A motivação para conduzir experimentos em curto e médio prazo é que, embora os experimentos de longo prazo demandem muito tempo, os experimentos de curto e médio prazo podem auxiliar na obtenção de resultados estatisticamente mais significativos em um período de tempo mais curto, ressaltando o caráter exploratório

da pesquisa. No entanto, é importante destacar que essa abordagem com foco na exploração empírica também apresenta limitações naturais, como a dependência de recursos computacionais e a dificuldade em estabelecer generalizações teóricas abrangentes.

Os experimentos de curto prazo (CP) foram inspirados nos *baselines*, mas com épocas de treino mais curtas. Os tamanhos das populações foram semelhantes a B2, porém, mantivemos os possíveis hiperparâmetros de RNPs mais próximos de B1. Adicionalmente, empregamos apenas um subconjunto dos dados de treinamento e reduzimos ainda mais a duração do algoritmo coevolutivo (4 gerações em vez de 40). Para médio prazo (MP) e longo prazo (LP), a duração do experimento e treinamento dos indivíduos são utilizados valores substancialmente maiores que em CP.

Tabela 6.2: Hiperparâmetros evolutivos e topológicos a serem utilizados. CP, MP e LP significam curto, médio e longo prazos, respectivamente.

Parâmetros	**CP**	**MP**	**LP**
Gerações	4	25	75
População de RNPs	10	10	100
População de *blueprints*	10	25	25
População de módulos	30	45	45
Épocas durante a evolução	5	10	10
Épocas de treino final	100	150	150
Quantidade de dados no treino	40%	100%	100%
Filtros	[32, 256]		
Tamanho do kernel	{1, 3, 5}		
Taxa de *dropout*	[0, 0.5]		
Max Pooling	{Yes, No}		
Batch Normalization	{Yes, No}		

Considerando as funções de inicialização mais usadas em Aprendizado Profundo (He, Glorot e Random), cada uma das mesmas possui uma variante Uniforme e uma Normal, que determina a distribuição da qual extrairá os pesos. Para os experimentos de curto prazo, avaliamos o algoritmo CoDeepNEAT em dois conjuntos de dados (CIFAR-10 e MNIST) com cinco iterações de *hold-out*. No entanto, como os experimentos de médio e longo prazo demoram significativamente mais para serem concluídos, foi limitado os experimentos a uma única iteração de *hold-out* no conjunto de dados CIFAR-10.

Em experimentos de CP, um objetivo secundário é analisar, através dos resultados, quais das funções de ativação mais utilizadas (tanH ou ReLU) é mais adequada para experimentos com CoDeepNEAT após episódios de treinamento muito curtos.

Para experimentos de MP e LP, o objetivo primordial é buscar alguma relação, de forma empírica, entre conexões residuais, acurácia e tamanho da rede de acordo com a combinação das funções de inicialização já citadas. Nesses experimentos, apenas a função ReLU foi empregada como função de ativação.

6.3 Implementação CoDeepNEAT

A versão do CoDeepNEAT tida como base foi feita por Bohrer et al. (2020)., disponível em [github.com/sbcblab/Keras-CoDeepNEAT], implementada em Python no *framework* de Aprendizado de Máquina Keras. Entretanto, recursos adicionais foram necessários, para além de correções de *bugs* legados em código, como alteração do código fonte para utilização de GPUs, otimização na criação e treino de redes neuras montadas de maneira individual (o código original).

Como os experimentos rodaram em servidores dedicados, distantes fisicamente do aluno, foi implementado também uma rotina de criação e acesso a *checkpoints* salvos rotineiramente ao fim de cada geração, no intuito de evitar qualquer perda substancial de trabalho em decorrência de possíveis contratempos como queda de energia ou manunteção dos servidores.

Capítulo 7

Resultados

Neste capítulo, serão apresentados os resultados obtidos nos três tipos de experimentos realizados, que visaram aferir a influência de diferentes funções de inicialização de pesos em redes neurais profundas evoluídas pelo algoritmo CoDeepNEAT no desempenho de reconhecimento de imagens nas bases de dados MNIST e CIFAR-10. Os resultados apresentados incluirão métricas de desempenho, como acurácia e perda (*loss*), bem como comparações estatísticas entre os diferentes experimentos, a fim de avaliar a significância das diferenças encontradas. Além disso, serão discutidas as implicações práticas e teóricas dos resultados obtidos, com ênfase na importância das funções de inicialização de pesos em redes neurais profundas evoluídas por algoritmos evolutivos. Em especial, para os experimentos de Longo prazo, reutilizamos a parte final da prova matemática da função He para a ativação ReLU para aferir o alinhamento teórico com os resultados práticos encontrados.

7.1 Curto prazo

Os resultados de CP são apresentados nas Tabelas 7.1 e 7.2. As Tabelas 7.1 e 7.2 fornecem os resultados dos melhores indivíduos treinados após a última geração em cada uma das cinco execuções de *hold-out*, para ReLU e tanH, respectivamente. A Tabela 7.3 fornece a média e os desvios padrão.

Primeiro, vamos discutir os experimentos realizados com a ativação ReLU. Observamos que, em todas as execuções, os resultados são melhores que o segundo *baseline* para o conjunto de dados MNIST. O pior resultado de curto prazo para a base MNIST foi de 99%, enquanto o B2 reportou 92%. No CIFAR-10, os resultados de CP tiveram um desvio padrão maior. Em média, as redes encontradas com inicialização He foram semelhantes ao *baseline* em questão, enquanto os indivíduos evoluídos com a Glorot tiveram um desempenho ligeiramente melhor. Importante observar que a principal diferença dos experimentos feito em CP para B2 foi o maior espaço de busca para o número de filtros e a possibilidade de empregar *Max Pooling* e *batch normalization*. Isso sugere que executar

o algoritmo de NEP por gerações mais longas é menos importante do que fornecer ao algoritmo módulos mais flexíveis.

Tabela 7.1: Experimentos de curto prazo com funções de inicialização Glorot e He (ativação ReLU).

CIFAR-10			**MNIST**		
Inicialização	***Run***	**Acurácia**	**Inicialização**	***Run***	**Acurácia**
Glorot Normal	1	0,7681	Glorot Normal	1	0,9872
Glorot Normal	2	0,7861	Glorot Normal	2	0,9893
Glorot Normal	3	**0,8057**	Glorot Normal	3	0,9900
Glorot Normal	4	0,8034	Glorot Normal	4	0,9893
Glorot Normal	5	0,7995	Glorot Normal	5	**0,9903**
Glorot Uniform	1	**0,8462**	Glorot Uniform	1	0,9890
Glorot Uniform	2	0,7188	Glorot Uniform	2	0,9908
Glorot Uniform	3	0,8118	Glorot Uniform	3	0,9876
Glorot Uniform	4	0,7654	Glorot Uniform	4	**0,9914**
Glorot Uniform	5	0,8180	Glorot Uniform	5	0,9881
He Normal	1	0,7798	He Normal	1	0,9879
He Normal	2	0,6894	He Normal	2	0,9910
He Normal	3	0,7159	He Normal	3	0,9870
He Normal	4	0,7706	He Normal	4	**0,9921**
He Normal	5	**0,7919**	He Normal	5	0,9899
He Uniform	1	0,7743	He Uniform	1	0,9902
He Uniform	2	0,7767	He Uniform	2	**0,9923**
He Uniform	3	0,7324	He Uniform	3	0,9903
He Uniform	4	0,7695	He Uniform	4	0,9900
He Uniform	5	**0,7998**	He Uniform	5	0,9870

Em seguida, repetiu-se os experimentos de CP com a função de ativação tanh. Novamente, os experimentos foram realizados em CIFAR-10 e MNIST e repetidos cinco vezes em esquema de *hold-out*. Em ambos os casos, os experimentos foram executados em um sistema de CPU única com 15,5 GiB de RAM e uma GPU de 4 GB (GeForce GTX 1650/PCIe/SSE2). O tempo total de execução foi de 67 horas. Assim, considerando 2 funções de ativação, 4 métodos de inicialização e 2 conjuntos de dados, as combinações totais somam 16 configurações experimentais distintas, cada uma repetida cinco vezes, de modo que cada experimento levou em média 50 minutos para ser concluído.

O desempenho dos indivíduos encontrados com a tanh foi severamente menor se comparado aos indivíduos evoluídos com ReLU. Ainda assim, com exceção da Glorot Normal, os resultados foram melhores que os apresentados em B2. Mais uma vez, isso sugere que é possível obter resultados decentes quando o algoritmo de NEP é ajustado para encontrar "um aprendiz rápido" (Miikkulainen et al., 2019), RNPs que podem ser treinadas com poucas épocas durante o processo evolutivo.

Embora a utilização das bases de dados CIFAR-10 e MNIST seja comum em experimentos de visão computacional e forneça resultados comparáveis com outros estudos, é

Tabela 7.2: Experimentos de curto prazo com funções de inicialização Glorot e He (ativação tanH).

CIFAR-10			**MNIST**		
Inicialização	***Run***	**Acurácia**	**Inicialização**	***Run***	**Acurácia**
Glorot Normal	1	0.6867	Glorot Normal	1	0.3469
Glorot Normal	2	0.9514	Glorot Normal	2	**0.501**
Glorot Normal	3	**0.9841**	Glorot Normal	3	0.408
Glorot Normal	4	0.9547	Glorot Normal	4	0.4702
Glorot Normal	5	0.9676	Glorot Normal	5	0.3997
Glorot Uniform	1	0.9691	Glorot Uniform	1	0.4565
Glorot Uniform	2	**0.9804**	Glorot Uniform	2	0.5921
Glorot Uniform	3	0.9729	Glorot Uniform	3	0.4815
Glorot Uniform	4	0.9681	Glorot Uniform	4	**0.6103**
Glorot Uniform	5	0.9711	Glorot Uniform	5	0.6028
He Normal	1	**0.9758**	He Normal	1	0.4656
He Normal	2	0.9691	He Normal	2	**0.6694**
He Normal	3	0.9635	He Normal	3	0.62
He Normal	4	0.9667	He Normal	4	0.3195
He Normal	5	0.9739	He Normal	5	0.5062
He Uniform	1	0.9732	He Uniform	1	0.4722
He Uniform	2	0.9733	He Uniform	2	0.5088
He Uniform	3	0.9755	He Uniform	3	0.4008
He Uniform	4	**0.9762**	He Uniform	4	0.5931
He Uniform	5	0.9685	He Uniform	5	**0.6111**

Tabela 7.3: Acurácia de curto prazo com funções de inicialização Glorot e He, com funções de ativação não lineares.

Base de dados	**Inicialização**	**Ativação**	**Média da acurácia**	**Desvio padrão**
MNIST	Glorot Normal	tanh	0,9089	± 0,1116
MNIST	Glorot Uniform	tanh	0,9723	± 0,0046
MNIST	He Normal	tanh	0,9698	± 0,0045
MNIST	He Uniform	tanh	0,9733	± 0,0026
CIFAR-10	Glorot Normal	tanh	0,4532	± 0,0547
CIFAR-10	Glorot Uniform	tanh	0,5487	± 0,0658
CIFAR-10	He Normal	tanh	0,5143	± 0,1237
CIFAR-10	He Uniform	tanh	0,5172	± 0,0777
MNIST	Glorot Normal	ReLU	0,9892	± 0,0010
MNIST	Glorot Uniform	ReLU	0,9893	± 0,0014
MNIST	He Normal	ReLU	0,9895	± 0,0018
MNIST	He Uniform	ReLU	0,9899	± 0,0016
CIFAR-10	Glorot Normal	ReLU	0,7925	± 0,0139
CIFAR-10	Glorot Uniform	ReLU	0,7920	± 0,0448
CIFAR-10	He Normal	ReLU	0,7495	± 0,0397
CIFAR-10	He Uniform	ReLU	0,7705	± 0,0217

importante destacar algumas limitações intrínsecas a essas escolhas. Ambas as bases de

dados representam problemas de classificação de imagem mais simples e com entradas de tamanho reduzido, o que pode não capturar adequadamente a complexidade e os desafios encontrados em aplicações do mundo real. Além disso, ao se concentrar apenas em tarefas de visão computacional, a generalização dos resultados para outros domínios da aprendizagem de máquina e aplicações de redes neurais fica comprometida. Seria interessante explorar como as abordagens propostas se comportariam em problemas de processamento de linguagem natural, detecção de anomalias, previsão de séries temporais, entre outros. Ao ampliar o escopo dos experimentos para incluir outras áreas e problemas mais complexos, a robustez e a relevância prática das descobertas seriam fortalecidas, contribuindo para uma compreensão mais completa das metodologias propostas.

7.1.1 Teste de Mann-Whitney

O teste de Mann-Whitney U (Mann e Whitney, 1947) é uma ferramenta estatística não paramétrica utilizada para comparar duas amostras independentes e verificar se elas foram retiradas de populações com a mesma distribuição. A estatística do teste é baseada no cálculo da soma dos postos, onde a diferença entre os postos das duas amostras é computada. A estatística de teste U é então calculada como o menor valor possível entre as somas dos postos de ambas as amostras. A fórmula do teste é representada por:

$$U_i = R - \frac{n_1(n_1 + 1)}{2}$$

Onde R é a soma dos postos de uma das amostras e n é o tamanho dessa amostra. O valor de U é comparado com um valor crítico de acordo com o nível de significância adotado para a verificação da hipótese nula. Se o valor de U for menor ou igual ao valor crítico, então rejeita-se a hipótese nula de que as amostras foram retiradas de populações com a mesma distribuição.

Aplicando o teste de Mann-Whitney nos resultados das Tabelas 7.1 e 7.2, considerando os pares base de dados - inicialização, e comparando entre ReLU e tanH, Tabela 7.4, mostraram que, para alguns pares de configurações, há diferença estatisticamente significativa entre as amostras de ativação ReLU e tanH. Em particular, encontramos diferenças estatisticamente significativas para CIFAR-10 com inicialização Glorot Uniform e ambas as inicializações He (Normal e Uniform), com valor-p menor que 0,05. No entanto, para CIFAR-10 com inicialização Glorot Normal e todos os pares para o conjunto de dados MNIST, o valor-p foi maior que 0,05, indicando que não há diferença estatisticamente significativa entre as amostras de ReLU e tanH para essas configurações.

É importante ressaltar que os resultados do teste de Mann-Whitney devem ser interpretados com cuidado, pois existem outros fatores que podem influenciar no desempenho da rede neural, como a arquitetura da rede e o tamanho do conjunto de dados, por exemplo. Além disso, é preciso considerar que o teste de Mann-Whitney não indica qual ativação é

Conjunto de Dados	Inicialização	Estatística de Teste	Valor-p
CIFAR-10	Glorot Normal	5.0	0.1508
	Glorot Uniform	0.0	0.0079
	He Normal	0.0	0.0079
	He Uniform	0.0	0.0079
MNIST	Glorot Normal	25.0	0.0119
	Glorot Uniform	25.0	0.0079
	He Normal	25.0	0.0079
	He Uniform	25.0	0.0079

Tabela 7.4: Resultados do teste de Mann-Whitney para comparar as ativações ReLU e tanH em cada par de configurações.

melhor do que a outra, mas apenas se há diferença estatisticamente significativa entre as amostras. Portanto, é necessário realizar uma análise mais ampla e cuidadosa para avaliar qual função de ativação é mais adequada para cada conjunto de dados e configuração de rede neural.

7.2 Médio prazo

Os experimentos de médio prazo foram realizados de forma a contemplar uma proposta concreta entre as duas *baselines* utilizadas, uma vez que Bohrer et al. (2020) utilizaram uma quantidade muito pequena de poder computacional quando comparado ao que foi usado no documento original do CoDeepNEAT (2.000.000 CPUs e 5.000 GPUs). Por isso, para MP foi utilizada a mesma máquina que em CP (uma única GPU de 4 GB). Mas como o espaço de busca neste cenário foi bem maior do que no anterior, cada experimento levou aproximadamente 32 horas em vez da média de 50 minutos observada nos experimentos de curto prazo. Portanto, limitamos os experimentos de MP a um único *hold-out*.

As funções de inicialização foram as variantes uniforme e normal das funções He e Glorot, e também a inicialização aleatória (*Random*) – que atribui valores aos pesos dos neurônios sem levar em consideração nem a função de ativação, nem os tamanhos de entrada ou saída. Os resultados são mostrados na Tabela 7.5. Além da acurácia, também é relatado o número de parâmetros treináveis do melhor indivíduo e se esse indivíduo contém conexões residuais ou não.

Considerando o maior número de gerações, as RNPs montadas pelo CoDeepNEAT podem obter alguma conexão residual devido a mutações em seus *blueprints*. No entanto, notamos que apenas a inicialização do tipo He foi capaz de produzir o melhor indivíduo com alguma conexão residual, enquanto todas as demais funções convergiram para um indivíduo cuja arquitetura era totalmente sequencial (ausência total de *skip connections*). É importante mencionar que, durante o processo evolutivo, foram encontrados indivíduos com vários graus de residualidade em todos os casos, mas apenas as arquiteturas inici-

Tabela 7.5: Melhores indivíduos para os experimentos de médio prazo para cada processo evolutivo.

Base de dados	Inicialização	Indivíduo final	Acurácia	Parâmetros
CIFAR-10	*He Uniform*	Residual	0,8787	$\approx 3,33$ M
CIFAR-10	*Glorot Normal*	Sequencial	0,8581	$\approx 1,15$ M
CIFAR-10	*He Normal*	Residual	0,8336	$\approx 1,25$ M
CIFAR-10	*Glorot Uniform*	Sequencial	0,8011	$\approx 2,25$ M
CIFAR-10	*Random Uniform*	Sequencial	0,7380	$\approx 1,06$ M
CIFAR-10	*Random Normal*	Sequencial	0,7271	$\approx 2,03$ M

alizadas com *He Uniform* ou *He Normal* conseguiram criar indivíduos com arquiteturas residuais que superaram o desempenho dos indivíduos com arquiteturas não residuais durante seus respectivos processos evolutivos. A reconstrução em Keras de todos os melhores indivíduos pode ser encontrada no Apêndice B.

Apesar de uma das inicializações He ter o melhor resultado, como visto na Tabela 7.5, com $87,87\%$ de precisão, a inicialização Glorot Normal obteve um resultado semelhante, $85,81\%$, com aproximadamente $34,72\%$ menos pontos flutuantes e $34,53\%$ menos parâmetros totais, o que equivale ao número total de parâmetros para realizar uma inferência no modelo. Assim, a *He Uniform* obteve o melhor resultado, mas o inicializador *Glorot Normal* parece ser tão competitivo quanto ainda que com menos demanda por recursos. Uma explicação razoável para esse fato é que a alta frequência de mutações permite o desenvolvimento de conexões residuais, mas apenas um grande número de gerações pode ser necessário para produzir indivíduos competitivos. Tanto que Miikkulainen et al. afirmam, no artigo original do CoDeepNEAT, que foi somente após a geração 70^{a} que a precisão dos indivíduos em seus experimentos se estabilizou.

Ao utilizar apenas a base de dados CIFAR-10 neste estudo, algumas limitações importantes devem ser consideradas. Primeiramente, a CIFAR-10 é uma base relativamente simples, composta por imagens de baixa resolução (32x32 pixels), o que pode não refletir adequadamente a complexidade encontrada em problemas reais envolvendo imagens de maior resolução e detalhes mais sutis. Além disso, o foco exclusivo na visão computacional restringe a generalização dos resultados para outras tarefas, como processamento de linguagem natural ou análise de séries temporais, onde as arquiteturas de redes neurais e técnicas de inicialização podem ter desempenhos diferentes. Ao avaliar os métodos propostos apenas com a CIFAR-10, limita-se a compreensão de como esses métodos se comportam em diferentes contextos e tarefas, o que dificulta a determinação de sua eficácia geral e aplicabilidade em outros problemas. Para obter uma avaliação mais abrangente e robusta das técnicas propostas, seria benéfico incluir outras bases de dados de diferentes domínios e com características variadas.

7.3 Longo prazo

Executamos experimentos de longo prazo com três vezes mais gerações do que os experimentos de MP. No entanto, à medida que as RNs se tornam mais complexas ao longo das gerações, o tempo total para terminar foi substancialmente maior: 40 dias por experimento ($\sim$ 920 horas), quando comparado às 32 horas necessárias para terminar cada experimento de médio prazo. Apenas um *hold-out* foi considerado para *Glorot Normal* e *He Uniform*, que foram as inicializações mais precisas nos experimentos de MP. Os resultados são apresentados na Tabela 7.6. As reconstruções em Keras das duas melhores redes para cada função de inicialização podem ser encontradas no Apêndice A.

Tabela 7.6: Melhores indivíduos de experimentos de longo prazo para cada processo evolutivo.

Base de dados	**Inicialização**	**Indivíduo final**	**Acurácia**	**Parâmetros**
CIFAR-10	*He Uniform*	Residual	0,8991	$10,23$ M
CIFAR-10	*Glorot Normal*	Sequencial	0,8454	$0,47$ M

Da mesma forma que os resultados anteriores, o melhor indivíduo evoluído coma inicialização He obteve uma melhor precisão ($89,91\%$) do que o indivíduo evoluído com a Glorot ($85,54\%$). A RNP encontrada com a Glorot foi completamente linear, como mostrado na Figura 7.1 (esquerda). Em comparação, a RNP encontrada coma He foi rica em conexões residuais.

A utilização exclusiva da base CIFAR-10 nos experimentos de longo prazo, assim como mencionado na seção anterior, também apresenta limitações. Dada a simplicidade das imagens de baixa resolução, os resultados podem não abranger adequadamente a complexidade de problemas reais com imagens mais detalhadas. Além disso, ao focar apenas em visão computacional, a generalização para outras tarefas é limitada. Para uma avaliação mais abrangente, seria aconselhável explorar bases de dados variadas e de diferentes domínios.

7.3.1 Convergência

A explicação mais plausível para a diferença notável de Redes geradas pela Glorot do que a He, nos experimentos de médio e longo prazo, muito provavelmente se dá a fórmula originária de ambas as inicializações. Conforme Kumar, se calcularmos os momentos centrais de x_{m+1} em termos de x_m quando $\sigma^2 = 1/N$ (Kumar, 2017), a partir da Equação 5.40, para a função de ativação ReLU na inicialização Glorot, obtemos o seguinte resultado:

$$u_m^2 = s_m^2 + \mu_m^2 \tag{7.1}$$

Usando a Equação 7.1 na Equação 5.41 produzimos a recursão seguinte:

$$\mu_{m+1}^2 = \frac{1}{2\pi}(s_m^2 + mu_m^2) \approx 0,16(s_m^2 + \mu_m^2) \tag{7.2}$$

Similarmente, ajustando os resultados das Equações 7.1 e 7.2 na Equação 5.43:

$$s_{m+1}^2 \approx 0,34(s_m^2 + \mu_m^2) \tag{7.3}$$

Manipulando, portanto, as Equações 7.1, 7.2, 7.3:

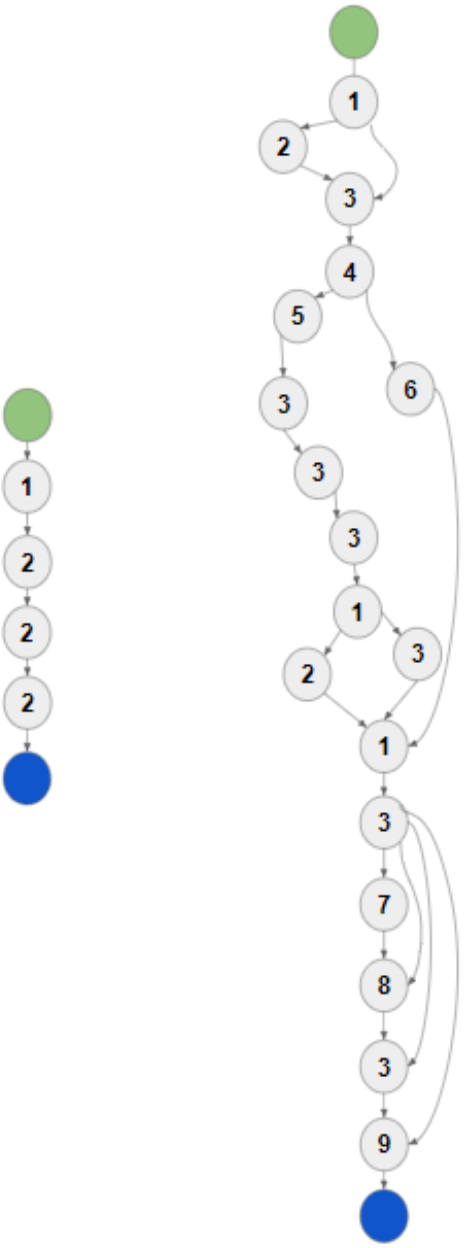

Figura 7.1: A estrutura geral de duas redes diferentes evoluiu com a inicialização Glorot (esquerda) e He (direita), respectivamente. Cada nó representa uma camada e os diferentes nós representam módulos diferentes. O Apêndice A contém as duas redes separadamente, estruturadas em Keras.

$$\mu_m^2 \approx 0,16 \times (0,51)^{m-1} \tag{7.4}$$

$$s_m^2 \approx 0,34 \times (0,51)^{m-1}, \tag{7.5}$$

para todo $m \geq 1$.

A Equação 7.5 mostra que a variância das entrada até as camadas mais profundas é exponecialmente menor do que a variância da entrada de camadas mais rasas ou próximas. Consequentemente, quando mais profunda a RN, pior a performance da inicialização Glorot. Isso nos leva a crer que apesar das múltiplas mutações no decorrer de gerações, em especial da adição de camadas, o que permite o aumento crescente da profundidade do melhor indivíduo em cada geração, a Glorot continuará a beneficiar a convergência de indivíduos menores e compactos. Nesse sentido, nem mesmo as mutações que criam conexões residuais parecem ser mutações boas o suficientes para ultrapassar esse entrave na convergência.

Capítulo 8

Considerações finais

8.1 Conclusões e Contribuições

Esse trabalho visou analisar o impacto na acurácia e na topologia de arquiteturas de Redes Neurais Artficiais evoluídas por um algoritmo NeuroEvolutivo sujeio a múltiplas funções de inicialização de pesos. Como resultado foram realizados experimentos com o algoritmo *coevolution of Deep NeuroEvolution of Augmenting Topologies* (CoDeepNEAT) em dois conjuntos de dados relevantes (MNIST e CIFAR-10) utilizando múltiplas funções de inicialização de pesos diferentes. No processo NeuroEvolutivo uma população de codificações genéticas de Redes Neurais é desenvolvida para encontrar uma rede que resolva uma determinada tarefa. Entretanto, as funções de inicialização de peso corriqueiramente utilizadas foram criadas para servirem modelos de Aprendizado Profundo com arquiteturas fixas. Os efeitos principais resultantes dessa interação entre inicializações de pesos e algoritmos NeuroEurovolutivos ainda são desconhecidos e pouco explorados na literatura.

Os experimentos propostos consideraram três cenários diferentes que envolvem espaços de busca diferenciados: curto, médio e longo prazo. Em cada etapa, a quantidade de gerações e indivídíduos aumenta consideravelmente, com impacto direto nos recursos computacionais. Os resultados obtidos foram comparados com duas funções de inicialização populares, He e Glorot, bem como uma estratégia de inicialização aleatória servindo como *benchmark*. Ademais, para experimentos de curto prazo, também foram realizados experimentos com duas funções de ativação não lineares comumente utilizadas: tangente hiperbólica (tanH) e ReLU.

Nos experimentos iniciais, com foco em parâmetros de curto prazo, a ativação ReLU superou os resultados obtidos pela tanH em todos os experimentos. No geral, considerando todos os três possíveis cenários de hiperparâmetros evolutivos, os melhores resultados obtidos ficaram acima de 99,20% no conjunto de dados MNIST e 89,91% no conjunto de dados CIFAR-10, superando o estudo de Bohrer et al. (2020), tido como um dos *baselines*.

Os resultados com o CoDeepNEAT, em especial para os experimentos Médio e Longo

prazo, comprovam a previsão teórica proposta por Kumar (2017) para o desempenho inferior da Glorot para Redes Neurais mais profundas. As mutações aleatórias nos *blueprint* e módulos, inserindo novas camadas, adicionando conexões residuais e aumentando o tamanho e complexidade da Rede montada, não são suficientes para ultrapassar essa barreira imposta pela diferença exponencial entre a variância das camadas mais próximas da entrada e das camadas mais profundas. Os experimentos também mostram que pode ser mais importante ajustar os hiperparâmetros do algoritmo coevolucionário e tentar evoluir um "aprendiz rápido" em menos gerações do apenas que realizar experimentos mais longos.

É importante notar que o melhor resultado obtido para o CIFAR-10 ($89,91\%$) foi derivado de experimentos de longo prazo. Além disso, chamamos a atenção para um fator crucial: considerando o universo de experimentos de médio e longo prazo, todos os melhores indivíduos evoluídos com Glorot (variantes Uniform e Normal) não continham conexões residuais, enquanto todos os melhores indivíduos He (Uniform ou Normal) tinham várias conexões residuais e eram arquiteturas muito mais profundas.

A discrepância de poder computacional disponível para os pesquisadores do artigo original do CoDeepNEAT (Miikkulainen et al., 2019) impactou drasticamente a análise dos experimentos pela incapacidade de replicar os experimentos necessários, em médio e longo prazo, da maneira elaborada em curto prazo (com múltiplos *hold-out*).

Dispondo de um *cluster* com 2.000.000 CPUs e 5.000 GPUs, os resultados de Miikkulainen et al. (2019) não puderam ser repetidos em perfeição. O custo elevado é um tema ainda recorrente na NeuroEvolução. Como relatado no Capítulo 2, para realizar todos seus experimentos Desell (2018) precisou de mais de 225.000 Redes Neurais, ao mesmo tempo que utilizou os computadores de mais 3.500 voluntários para rodar os experimentos. A necessidade de se ter múltiplas RNs, que ficam mais complexas e profundas com o decorrer das gerações, e treiná-las em cada etapa é ao mesmo tempo o diferencial e o principal problema dessa área.

Tamanha discrepância entre disponibilidade de poder computacional e concepção de ideias a serem realizadas é o principal fator de entrave para que a NeuroEvolução obtenha mais popularidade na área acadêmica e na indústria. Atestar empiricamente que o melhor indivíduo de Médio e Longo Prazo evoluído pela He supera a Glorot confirma a relevância da pesquisa quando ponderado a escolhao ótima dos hiperparâmetros de uma Rede Neural durante o processo NeuroEvolutivo.

Embora toda pesquisa tenha sido feita sob grande contribuição do código disponível em (Bohrer et al., 2020), este precisou ser atualizado e corrigido em múltiplas partes para funcionamento. O código implementado inicialmente por Bohrer et al. (2020) permitia apenas o uso de CPUs, o que era um empecilho considerável supondo a necessidade de realizar experimentos com um número elevado de populações e gerações. Por conseguinte, uma atualização foi feita para que os experimentos com o CoDeepNEAT pudessem ser

realizados em múltiplas GPUs paralelamente. Entretanto, tal implementação exigiu a reformulação de grande parte da implementação do CoDeepNEAT. O código original instaciava, montava, treinava, calculava a aptidão e realizava o *crossover* entre os indivíduos em uma única porção de memória—-o que é natural, uma vez que o mesmo foi criado para contemplar experimentos com uma GPU de 32 GB e com um número reduzidos de hiperparâmetros evolutivos (Bohrer et al., 2020). Neste cenário, todos os indivíduos podem estar carregados em memória; populações de Redes Neurais montadas, populações de *blueprints* e populações de módulos. Para realizar a pesquisa idealizada foi otimizada toda a parte de cálculo de aptidão entre indivíduos para que apenas um indivíduo por vez ocupasse a memória da GPU[1]. Cada Rede Neural montada é, então, salva em disco, como arquivo .hdf5, e calculada sua aptidão de forma individual. Terminado o *loop* de iteração para a aptidão, os melhores indivíduos de cada população eram selecionados e a pasta contendo todos os arquivos .hdf5 era excluída, e apenas o melhor indivíduo da geraçao atual era salvo em área separada para análise e compração posterior. Isto possiblitou o uso mais efetivo das GPUs nos experimentos de médio e longo prazo.

É importante abordar as limitações associadas ao uso exclusivo das bases de dados de visão computacional MNIST e CIFAR-10. É necessário reconhecer que essas bases de dados são relativamente simples e de entrada pequena, o que pode limitar a generalização das conclusões para outros domínios e tarefas mais complexas. Portanto, pesquisas futuras devem considerar a aplicação dos métodos e técnicas empregados neste estudo em bases de dados mais desafiadoras e variadas, incluindo aquelas voltadas para outras áreas além da visão computacional, como processamento de linguagem natural e reconhecimento de voz. Ao abordar diferentes tipos de tarefas e domínios, será possível obter uma compreensão mais abrangente da eficácia e aplicabilidade da Computação NeuroEvolutiva e da influência das funções de inicialização em diversos contextos. Essa expansão do escopo do estudo permitirá não apenas validar os resultados obtidos nesta pesquisa, mas também identificar novas oportunidades e desafios na área de NeuroEvolução Profunda.

Sumarizando as considerações realizadas, o trabalho realizado traz contribuições para a área de NeuroEvolução. Esse trabalho de dissertação: (*i*) melhora os conhecimentos acerca das funções de inicialização de pesos He, Glorot e Random quando utilizados em NeuroEvolução; (*ii*) caracteriza a importância de conexões residuais em algoritmos NeuroEvolutivos; (*iii*) promove a criação de um protocolo experimental para analisar o impacto de funções de inicialização de pesos em cenários diferenciados de poder computacional disponível; e, (*iv*) disponibiliza código fonte para reprodução dos resultados obtidos.

Como resultado deste trabalho de dissertação, alguns trabalhos foram produzidos e publicados:

1. (Evangelista e Giusti, 2021): Resumo expandido em língua inglesa intitulado "*Short-*

[1]O código deste trabalho está disponível em *github.com/lucasgabrielce/keras-codeepneat*

term effects of weight initialization functions in Deep NeuroEvolution", publicado no *EvoApplications 24th European Conference on the Applications of Evolutionary and bio-inspired Computation* (EvoApp 2021). Esse artigo descreve os primeiros experimentos realizados inicialmente no âmbito da configuração de curto prazo, sem *data augmentation*.

2. (Evangelista e Giusti, 2022): Artigo completo em língua inglesa intitulado "*Short-and-Long-Term Impact of Initialization Functions in NeuroEvolution*", publicado no *Intelligent Systems: 11th Brazilian Conference* (BRACIS 2022). Este artigo apresenta os resultados obtidos empiricamente dentro dos experimentos de curto, médio e longo prazo.

8.2 Trabalhos Futuros

Nesta seção são apresentadas algumas sugestões de trabalhos futuros. Tais sugestões de trabalhos visam aumentar o embasamento estatístico e investigar caminhos em aberto a partir do trabalho realizado.

Considerando a importância da residualidade no projeto de construção de uma rede neural, parece haver evidências que favorecem argumentos para uma possível investigação sobre essa maior capacidade das inicializações de He para facilitar o desenvolvimento de arquiteturas residuais eficientes em processos de evolução de curto prazo e médio prazo. Entretanto o recurso computacional para realização dos experimentos dificulta a reprodução múltipla do mesmo. Uma nova pesquisa em posse de um poder computacional maior abre um novo leque de possibilidades que permite, inclusive, embasar as argumentações com outras variáveis estatísticas para além da média de acurácia e parâmetros treináveis, indicações binárias da presença de residualidade em uma rede e valores nautrais para a quantidade de FLOPs utilizados.

A pesquisa acerca de funções de inicializações em NeuroEvolução ainda é escassa. Isto indica que esta área possui inúmeras possibilidades de vasto potencial de exploração. Isto sugere ao menos uma linha clara de trabalho: o desenvolvimento de funções de inicialização de pesos próprias para algoritmos NeuroEvolutivos. O trabalho de Lyu et al. (2020) aprofunda técnicas de herança de pesos entre pais e filhos durante a etapa de *crossover*, mas não inidica um caminho para a criação própria de inicialização para algoritmos NeuroEvolutivos. Ainda sobre o trabalho de Lyu et al. (2020), a possibilidade de evolução entre indivíduos, seguindo os princípios Lamarckianos, como proposto pelos mesmos, supõe um caminho suplementar ao trabalho desenvolvido nesta dissertação de Mestrado. Entre uma geração e outra, após a existência de uma nova população de *blueprints* e *módulos* e as novas Redes Neurais serem montadas, as mesmas, embora herdarem diretamente estruturas toplógicas evoluídas previamente, não herdam os pesos

adquiridos após a curta etapa de treino. O trabalho de Lyu et al. (2020) indica uma possibilidade de trabalho a ser explorada.

O problema do *cold start* para o CoDeepNEAT, isto é, o início aleatório para as populações de *blueprints* e *módulos*, também pode ser considerado uma outra fronte de pesquisa. Garantir que o CoDeepNEAT inicie com estruturas sub-ótimas de Redes Neurais (relação entre camadas de convolução e *batch normalization*, quantidade de parâmetros, profunidade etc.) diminui o espaço de busca inicial. Esta dissertação de mestrado seguiu o roteiro da inicialização aleatória por ser o protocolo utilizado no artigo original do CoDeepNEAT, mas isso não impede trabalhos futuros que insiram conhecimento humano no processo evolutivo.

Como última sugestão, uma vez que os trabalhos desta dissertação tiveram como foco as bases de dados MNIST e CIFAR-100, o desempenho do CoDeepNEAT, com as funções de inicialização já citadas, com outras bases de dados pode ser explorado. Bases conhecidas como CIFAR-100 e Fashion-MNIST, utilizadas como *bechmark* para modelos de Visão Computacional, mas também bases de dados tabulares, ou outros campos gerais de aplicações como linguagem natural, séries temporais e reconhecimento de fala, fornecem ideias novas de pesquisas a serem feitas.

Referências Bibliográficas

Aaltonen, T., Adelman, J., Akimoto, T., Albrow, M., Gonzalez, B. A., Amerio, S., Amidei, D., Anastassov, A., Annovi, A., Antos, J., et al. (2009). Measurement of the top-quark mass with dilepton events selected using neuroevolution at cdf. *Physical review letters*, 102(15):152001. Citado nas páginas 16 e 23.

Achim, S. (2020). Coding artificial intelligence snake using neat. Citado na página 23.

Adaloglou, N. (2020). Intuitive explanation of skip connections in deep learning. *https://theaisummer.com/*. Citado na página 36.

Agudo, M. G. (2021). The starting point of evolution strategies: the ee-(1+1) algorithm. *Medium*. Citado na página 9.

Bertoin, D., Bolte, J., Gerchinovitz, S., e Pauwels, E. (2021). Numerical influence of relu'(0) on backpropagation. *Advances in Neural Information Processing Systems*, 34:468–479. Citado na página 48.

Bird, A. (2007). Perceptions of epigenetics. *Nature*, 447(7143):396. Citado na página 11.

Bling, S. (2015). Mari/o - machine learning for video games. Citado na página 23.

Bohrer, J. d. S., Grisci, B. I., e Dorn, M. (2020). Neuroevolution of neural network architectures using codeepneat and keras. *arXiv preprint arXiv:2002.04634*. Citado nas páginas xv, 28, 29, 30, 51, 54, 59, 65, 66 e 67.

Box, G. E. (1957). Evolutionary operation: A method for increasing industrial productivity. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 6(2):81–101. Citado na página 15.

Bremermann, H. J. et al. (1962). Optimization through evolution and recombination. *Self-organizing systems*, 93:106. Citado na página 15.

Chung, J., Gulcehre, C., Cho, K., e Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. Citado na página 13.

Collins, J., Sohl-Dickstein, J., e Sussillo, D. (2016). Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*. Citado na página 13.

Desell, T. (2017). Large scale evolution of convolutional neural networks using volunteer computing. Em *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, páginas 127–128. Citado na página 11.

Desell, T. (2018). Accelerating the evolution of convolutional neural networks with node-level mutations and epigenetic weight initialization. Em *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, páginas 157–158. Citado nas páginas 11, 13 e 66.

DeVore, R., Hanin, B., e Petrova, G. (2021). Neural network approximation. *Acta Numerica*, 30:327–444. Citado na página 48.

Dogan, S., Barua, P. D., Kutlu, H., Baygin, M., Fujita, H., Tuncer, T., e Acharya, U. R. (2022). Automated accurate fire detection system using ensemble pretrained residual network. *Expert Systems with Applications*, página 117407. Citado na página 36.

Droste, S., Jansen, T., e Wegener, I. (2002). On the analysis of the (1+ 1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81. Citado na página 9.

Drozdzal, M., Vorontsov, E., Chartrand, G., Kadoury, S., e Pal, C. (2016). The importance of skip connections in biomedical image segmentation. Em *Deep learning and data labeling for medical applications*, páginas 179–187. Springer. Citado na página 35.

Dutta, P. e Mahanand, B. (2022). Affordable energy-intensive routing using metaheuristics. Em *Cognitive Big Data Intelligence with a Metaheuristic Approach*, páginas 193–210. Elsevier. Citado na página 9.

Eggers, F. (2020). Using neat to play snake. Citado nas páginas 23 e 24.

Eiben, A. E. e Smith, J. (2015). From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482. Citado na página 16.

Evangelista, L. G. C. e Giusti, R. (2021). Short-term effects of weight initialization functions in deep neuroevolution. *EvoApplications 24th European Conference on the Applications of Evolutionary and bio-inspired Computation*. Citado na página 67.

Evangelista, L. G. C. e Giusti, R. (2022). Short-and-long-term impact of initialization functions in neuroevolution. *Intelligent Systems: 11th Brazilian Conference, BRACIS 2022*, páginas 298–312. Citado na página 68.

Fogel, D. B. (1998). *Artificial intelligence through simulated evolution*. Wiley-IEEE Press. Citado na página 16.

Fogel, D. B. (2006). *Evolutionary computation: toward a new philosophy of machine intelligence*. John Wiley & Sons. Citado na página 16.

Fogel, L. J. (1962). Autonomous automata. *Industrial research*, 4:14–19. Citado na página 16.

Friedberg, R. M. (1958). A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13. Citado na página 15.

Friedberg, R. M., Dunham, B., e North, J. H. (1959). A learning machine: Part ii. *IBM Journal of Research and Development*, 3(3):282–287. Citado na página 15.

Glasmachers, T. (2020). Global convergence of the (1+ 1) evolution strategy to a critical point. *Evolutionary computation*, 28(1):27–53. Citado na página 9.

Glorot, X. e Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Em *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, páginas 249–256. JMLR Workshop and Conference Proceedings. Citado nas páginas 2, 13, 39, 40, 42, 43, 44 e 47.

Goodfellow, I. J., Bengio, Y., e Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. http://www.deeplearningbook.org. Citado nas páginas 2, 33, 34 e 40.

Greffenstette, J. J. e Baker, J. E. (1989). How genetic algorithms work: A critical look at implicit parallelism. Em *Proceedings of the 3rd International Conference on Genetic algorithms*, páginas 20–27. Citado na página 17.

He, K. e Sun, J. (2015). Convolutional neural networks at constrained time cost. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 5353–5360. Citado na página 35.

He, K., Zhang, X., Ren, S., e Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Em *Proceedings of the IEEE international conference on computer vision*, páginas 1026–1034. Citado nas páginas 2, 13, 39, 43 e 44.

He, K., Zhang, X., Ren, S., e Sun, J. (2016). Deep residual learning for image recognition. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 770–778. Citado nas páginas xv, 35 e 36.

Heinson, A. (2010). Observation of single top quark production at the tevatron collider. *Modern Physics Letters A*, 25(05):309–339. Citado na página 23.

Hirschbühl, D. (2008). Measurement of single top production at the tevatron. Relatório técnico, Fermi National Accelerator Lab.(FNAL), Batavia, IL (United States). Citado na página 23.

Hlav, E. (2022). Xavier glorot initialization in neural networks — math proof. Citado na página 40.

Hochreiter, S. e Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. Citado na página 13.

Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314. Citado na página 16.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press. Citado na página 16.

Hoorali, F., Khosravi, H., e Moradi, B. (2022). Irunet for medical image segmentation. *Expert Systems with Applications*, 191:116399. Citado na página 36.

Jagtap, A. D., Shin, Y., Kawaguchi, K., e Karniadakis, G. E. (2022). Deep kronecker neural networks: A general framework for neural networks with adaptive activation functions. *Neurocomputing*, 468:165–180. Citado na página 46.

Katanforoosh, K. e Kunin, D. (2019). Initializing neural networks. Citado na página 2.

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., e Shah, M. (2022). Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41. Citado na página 35.

Kicinger, R., Arciszewski, T., e De Jong, K. (2005). Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & structures*, 83(23-24):1943–1978. Citado na página 16.

Koutnik, J., Gomez, F., e Schmidhuber, J. (2010). Evolving neural networks in compressed weight space. Em *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, páginas 619–626. Citado nas páginas xv, 5, 6, 7 e 8.

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112. Citado na página 16.

Krizhevsky, A., Sutskever, I., e Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90. Citado na página 47.

Kumar, S. K. (2017). On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*. Citado nas páginas 2, 44, 61 e 66.

Leite, M. L. e Costa, F. F. (2017). Epigenômica, epigenética e câncer. *Revista Pan-Amazônica de Saúde*, 8(4):23–25. Citado na página 11.

Lennie, P. (2003). The cost of cortical computation. *Current biology*, 13(6):493–497. Citado na página 48.

Li, H., Xu, Z., Taylor, G., Studer, C., e Goldstein, T. (2018). Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31. Citado nas páginas xv, 36 e 37.

Luke, S. e Panait, L. (2006). A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344. Citado na página 16.

Lyu, Z., ElSaid, A., Karns, J., Mkaouer, M., e Desell, T. (2020). An experimental study of weight initialization and weight inheritance effects on neuroevolution. *arXiv preprint arXiv:2009.09644*. Citado nas páginas 11, 13, 40, 68 e 69.

Ma, Y. e Xie, Y. (2022). Evolutionary neural networks for deep learning: a review. *Int. J. of Machine Learning and Cybernetics*. Citado na página 2.

Mann, H. B. e Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, páginas 50–60. Citado na página 58.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2019). Evolving deep neural networks. Em *Artificial intelligence in the age of neural networks and brain computing*, páginas 293–312. Elsevier. Citado nas páginas xv, 1, 23, 24, 25, 26, 36, 51, 56 e 66.

Mishkin, D. e Matas, J. (2015). All you need is a good init. *arXiv preprint arXiv:1511.06422*. Citado na página 39.

Moscato, P. et al. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989. Citado na página 8.

Nair, V. e Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. Em *Icml*. Citado na página 48.

Neri, F. e Cotta, C. (2012). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14. Citado na página 9.

Nwankpa, C., Ijomah, W., Gachagan, A., e Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*. Citado na página 46.

Okada, H., Wada, T., Yamashita, A., e Matsuse, T. (2012). Interval-valued evolution strategy for evolving neural networks with interval weights and biases. Em *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*, páginas 2056–2060. IEEE. Citado nas páginas xv, 10, 11 e 12.

Ororbia II, A. G., Mikolov, T., e Reitter, D. (2017). Learning simpler language models with the differential state framework. *Neural computation*, 29(12):3327–3352. Citado na página 13.

Papavasileiou, E., Cornelis, J., e Jansen, B. (2020). A systematic literature review of the successors of 'neuroevolution of augmenting topologies'. *Evolutionary Computation*, páginas 1–73. Citado nas páginas 15, 25 e 26.

Rada, R. (2008). Expert systems and evolutionary computing for financial investing: A review. *Expert systems with applications*, 34(4):2232–2240. Citado na página 16.

Ramachandran, P., Zoph, B., e Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*. Citado na página 46.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., e Kurakin, A. (2017). Large-scale evolution of image classifiers. Em *International Conference on Machine Learning*, páginas 2902–2911. PMLR. Citado na página 11.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Library Translation 1122*. Citado na página 16.

Ronneberger, O., Fischer, P., e Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. Em *International Conference on Medical image computing and computer-assisted intervention*, páginas 234–241. Springer. Citado na página 35.

Saxe, A. M., McClelland, J. L., e Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*. Citado na página 39.

Schmidt-Hieber, J. (2020). Nonparametric regression using deep neural networks with relu activation function. *The Annals of Statistics*, 48(4):1875–1897. Citado na página 48.

Schwefel, H. (1968). Projekt mhd-staustrahlrohr: Experimentelle optimierung einer zweiphasendüse, teil i. Relatório técnico, Technischer Bericht 11.034/68, 35, AEG Forschungsinstitut, Berlin, Germany. Citado na página 16.

Schwefel, H.-P. (1965). Kybernetische evolution als strategie der experimentellen forschung in der stromungstechnik. *Diploma thesis, Technical Univ. of Berlin*. Citado na página 16.

Schwefel, H.-P. (1977). Evolutionsstrategien für die numerische optimierung. Em *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, páginas 123–176. Springer. Citado na página 17.

Shafiq, M. e Gu, Z. (2022). Deep residual learning for image recognition: a survey. *Applied Sciences*, 12(18):8972. Citado na página 35.

Sibi, P., Jones, S. A., e Siddarth, P. (2013). Analysis of different activation functions using back propagation neural networks. *Journal of theoretical and applied information technology*, 47(3):1264–1268. Citado na página 46.

Srivastava, R. K., Greff, K., e Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*. Citado na página 35.

Stanley, K. (2016). I'm ken stanley, artificial intelligence professor who breeds artificial brains that control robots and video game agents, and inventor of the neat algorithm – ama! Citado na página 23.

Stanley, K. O., Clune, J., Lehman, J., e Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35. Citado na página 18.

Stanley, K. O. e Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127. Citado nas páginas 11, 18, 19, 20 e 22.

Sussillo, D. e Abbott, L. (2014). Random walk initialization for training very deep feedforward networks. *arXiv preprint arXiv:1412.6558*. Citado na página 39.

Togelius, J., Gomez, F., e Schmidhuber, J. (2008). Learning what to ignore: Memetic climbing in topology and weight space. Em *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, páginas 3274–3281. IEEE. Citado nas páginas 8 e 10.

Whitley, L. D. et al. (1989). The genitor algorithm and selection pressure: why rank-based allocation of reproductive trials is best. Em *Icga*, volume 89, páginas 116–123. Fairfax, VA. Citado na página 17.

Wu, S., Zhong, S., e Liu, Y. (2018). Deep residual learning for image steganalysis. *Multimedia tools and applications*, 77(9):10437–10453. Citado nas páginas 33 e 34.

Zhang, K., Zuo, W., Chen, Y., Meng, D., e Zhang, L. (2017). Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing*, 26(7):3142–3155. Citado na página 35.

Zhou, G.-B., Wu, J., Zhang, C.-L., e Zhou, Z.-H. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234. Citado na página 13.

Zhou, X., Li, X., Hu, K., Zhang, Y., Chen, Z., e Gao, X. (2021). Erv-net: An efficient 3d residual neural network for brain tumor segmentation. *Expert Systems with Applications*, 170:114566. Citado na página 36.

Apêndice A

Melhores Redes nos experimentos de Longo Prazo

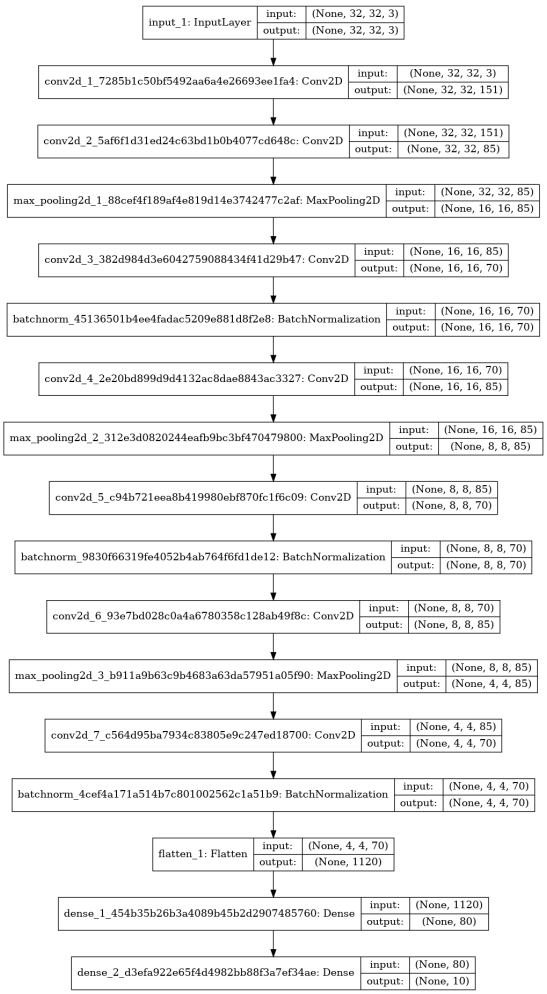

Figura 8.1: Melhor Rede encontrada pela inicialização Glorot ao término do processo evolutivo de Longo Prazo, construída através do framework Keras.

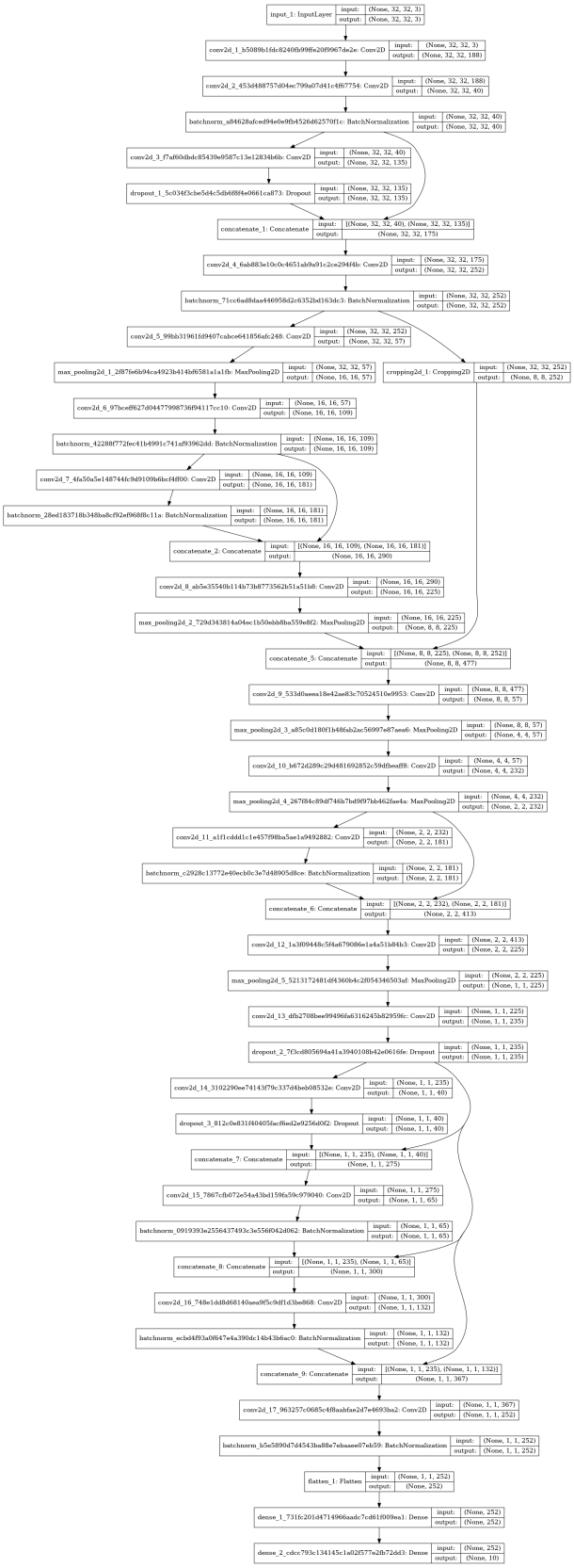

Figura 8.2: Melhor Rede encontrada pela inicialização He ao término do processo evolutivo de Longo Prazo, construída através do framework Keras.

Apêndice B

Experimentos de médio prazo

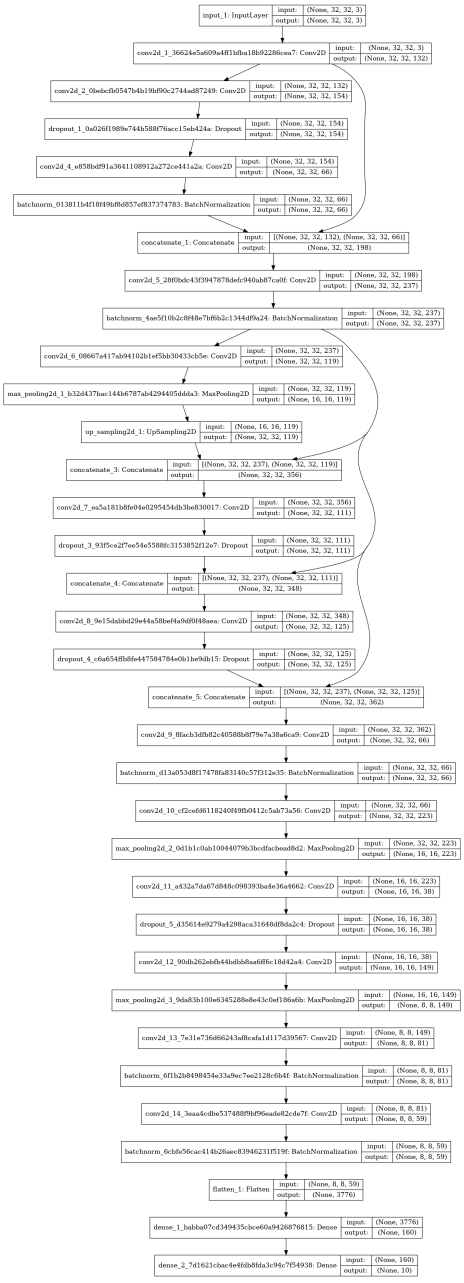

Figura 8.3: Melhor Rede encontrada pela inicialização He Uniform ao término do processo evolutivo de Médio Prazo, construída através do framework Keras.

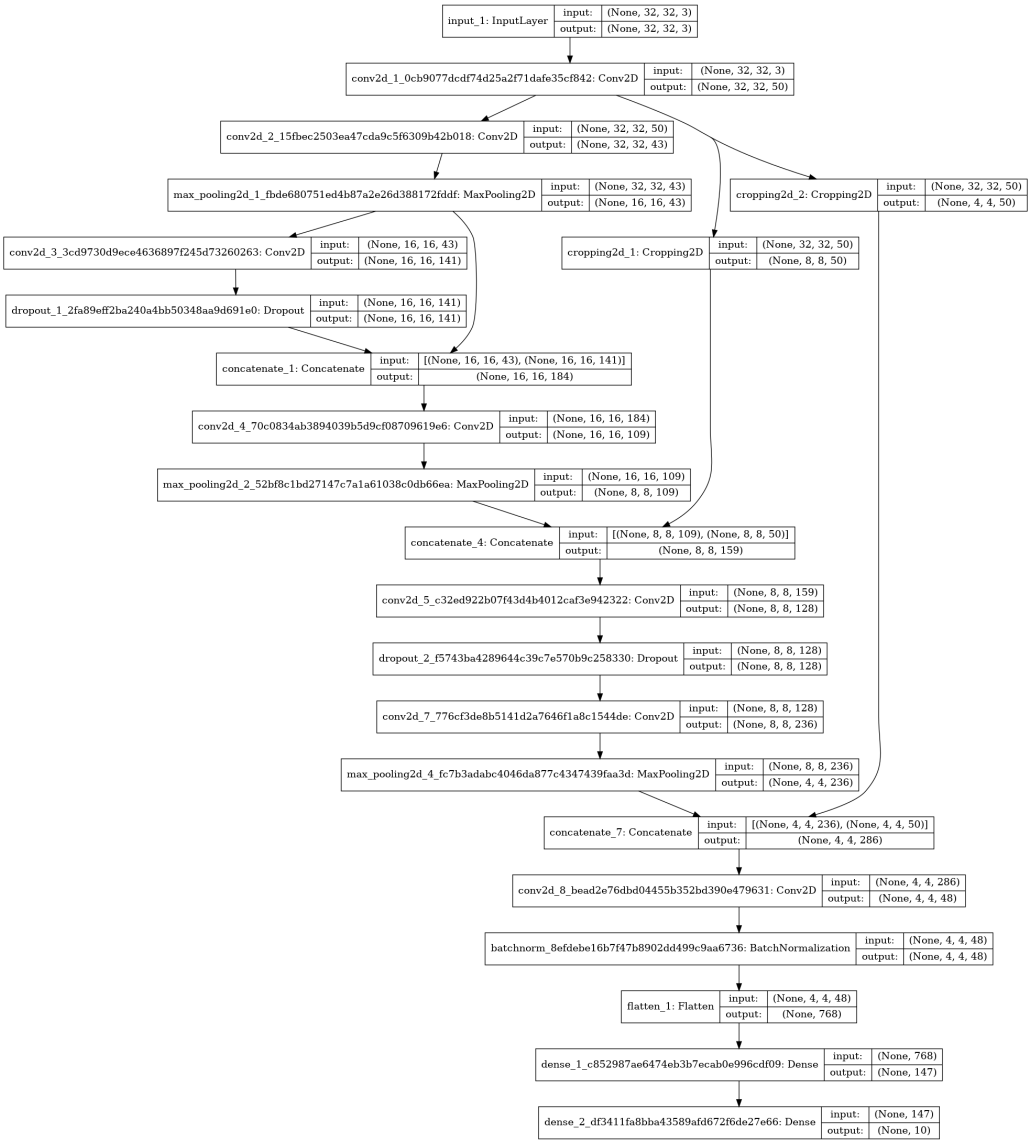

Figura 8.4: Melhor Rede encontrada pela inicialização He Normal ao término do processo evolutivo deMédio Prazo, construída através do framework Keras.

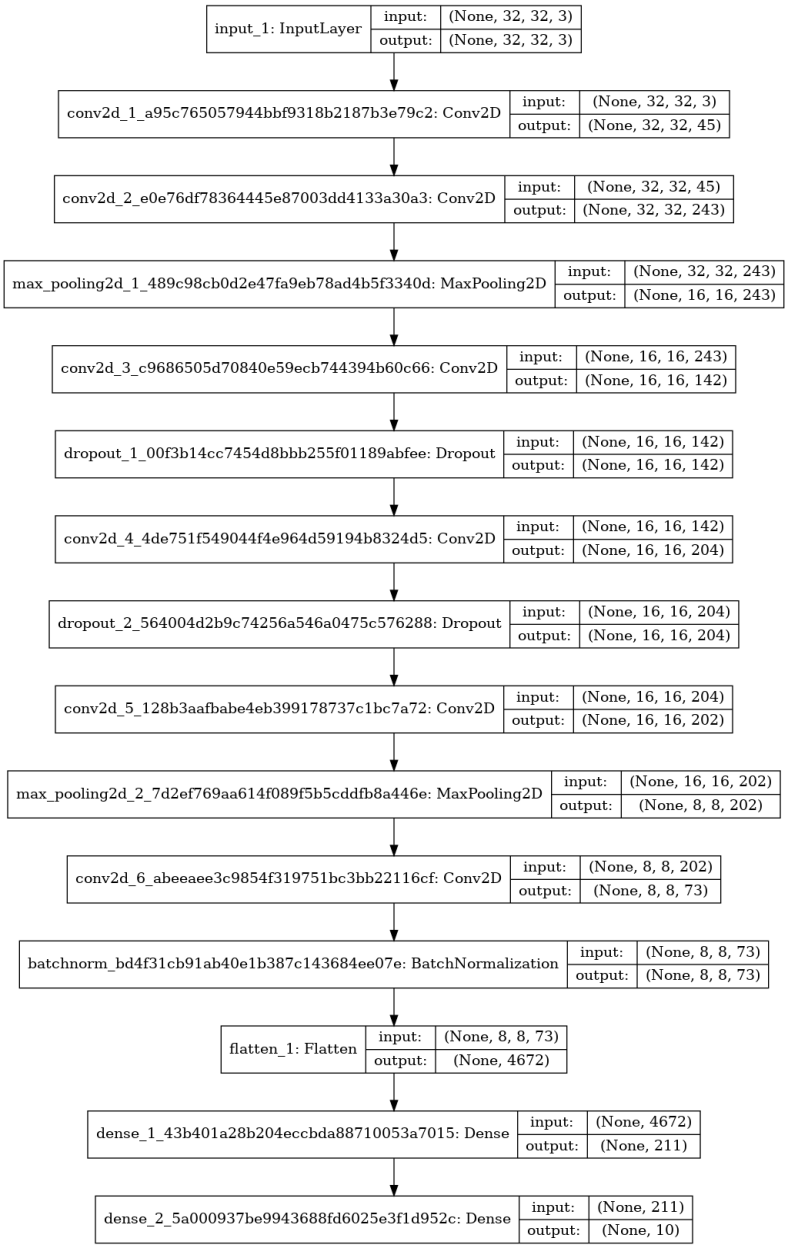

Figura 8.5: Melhor Rede encontrada pela inicialização Glorot Uniform ao término do processo evolutivo de Médio Prazo, construída através do framework Keras.

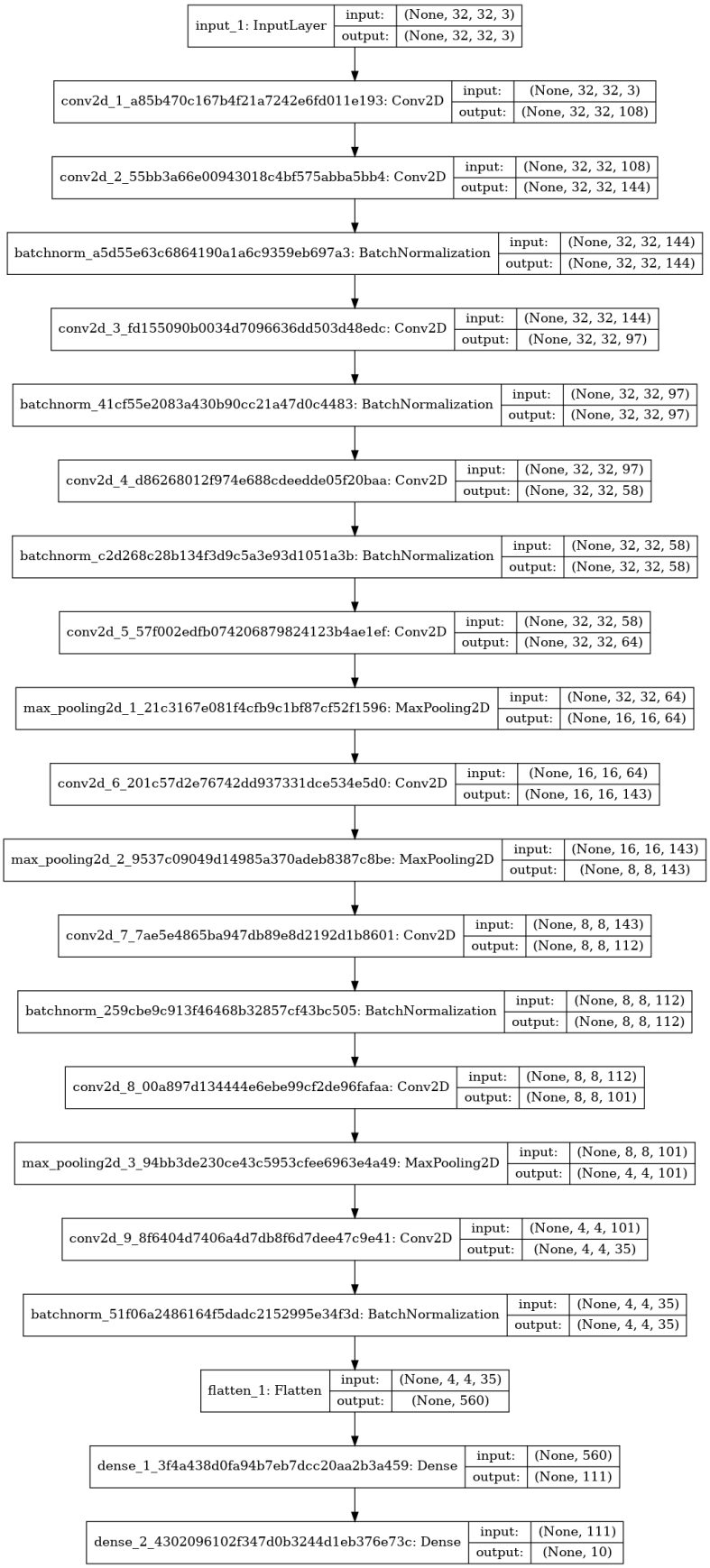

Figura 8.6: Melhor Rede encontrada pela inicialização Glorot Normal ao término do processo evolutivo de Médio Prazo, construída através do framework Keras.

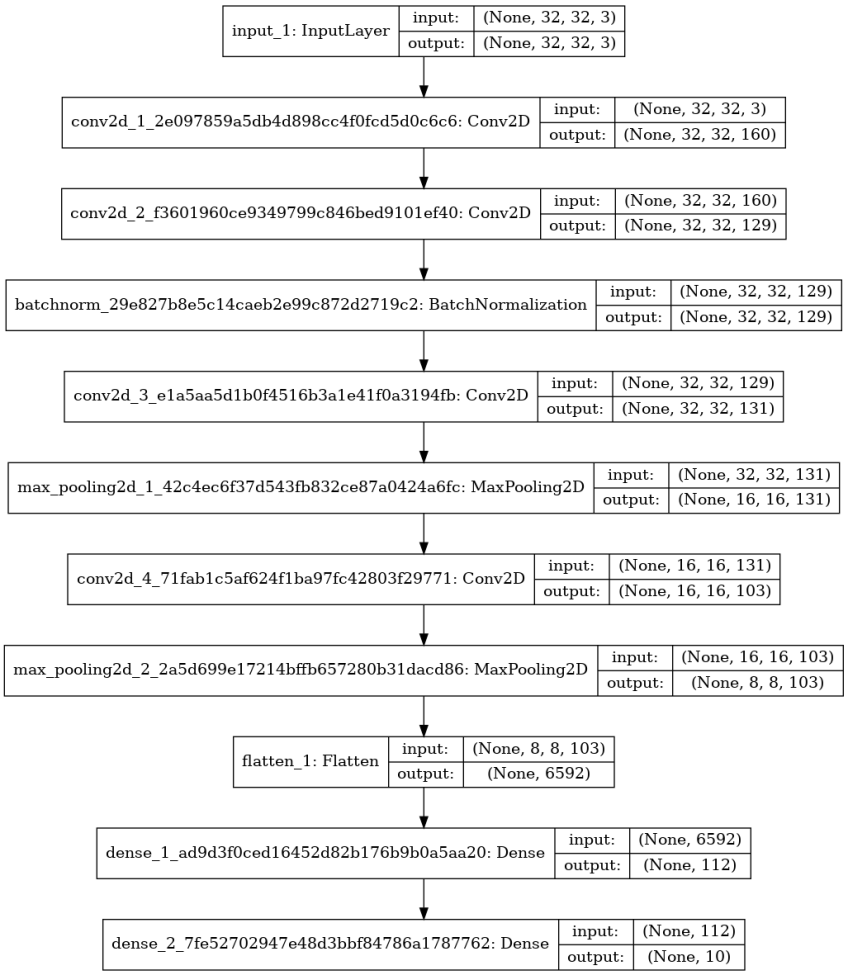

Figura 8.7: Melhor Rede encontrada pela inicialização Random Uniform ao término do processo evolutivo de Médio Prazo, construída através do framework Keras.

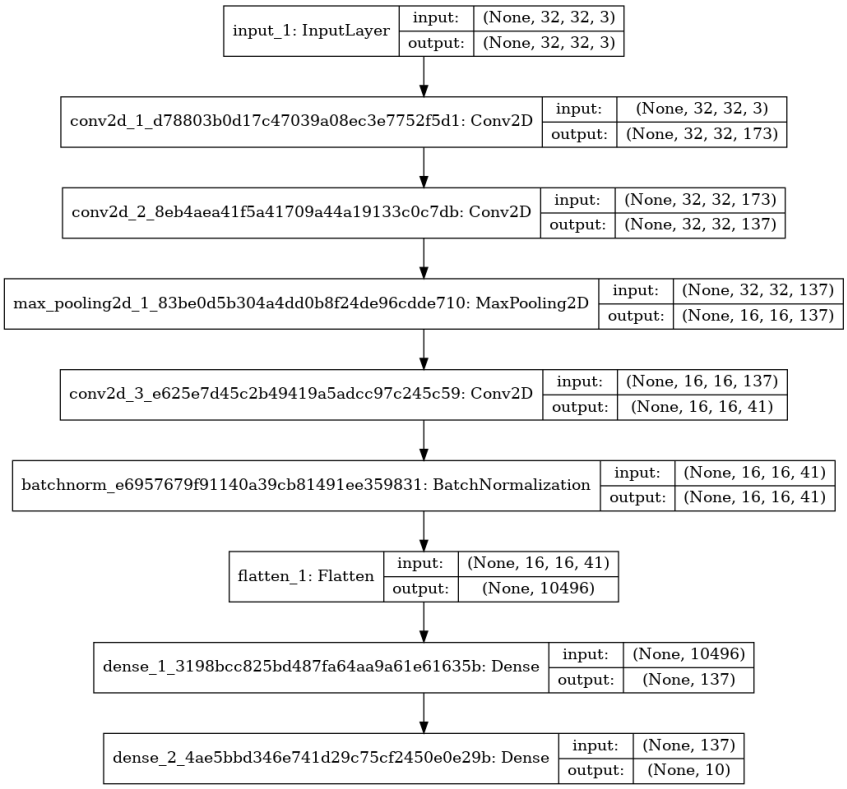

Figura 8.8: Melhor Rede encontrada pela inicialização Random Normal ao término do processo evolutivo de Médio Prazo, construída através do framework Keras.

Apêndice C

Exemplo de cálculo da função de aptidão (*fitness*) no CoDeepNEAT

Considerando os seguintes parâmetros evolutivos:

- Uma população de 5 módulos: M1, M2, M3, M4, M5 e M7;
- Uma população de 5 *blueprints*: B1, B2, B3, B4 e B5; e,
- uma população de 5 RN montadas: R1, R2, R3, R4 e R5.

As 5 RN que comporam a População P' foram montadas obedecendo às seguintes arquiteturas e obtiveram os respectivos valores de acurácia:

- RA: contruída em cima do *blueprint* B1, utilizou 2 módulos M2 e 1 módulo M5. Obteve $0,7$ de acurácia;
- RB: contruída em cima do *blueprint* B2, utilizou 1 módulo M1. Obteve $0,1$ de acurácia;
- RC: contruída em cima do *blueprint* B3, utilizou 1 módulo M4. Obteve $0,3$ de acurácia;
- RD: contruída em cima do *blueprint* B2, utilizou 1 módulo M3. Obteve $0,25$ de acurácia; e,
- RE: contruída em cima do *blueprint* B1, utilizou 2 módulos M1 e 1 módulo M2. Obteve $0,8$ de acurácia.

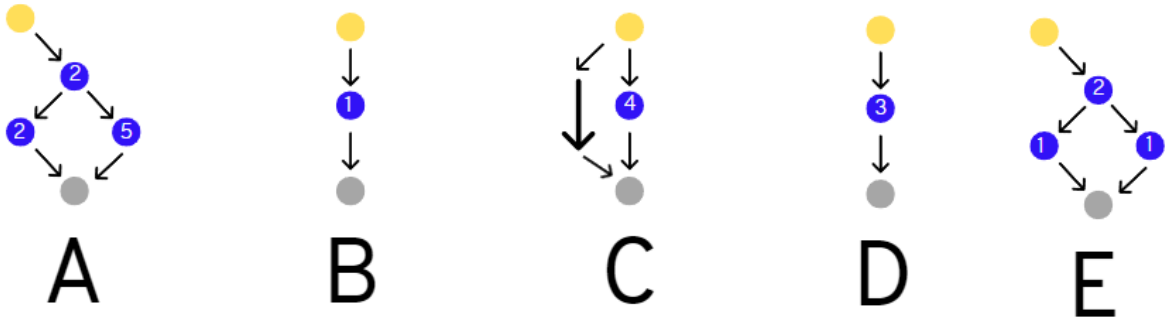

Figura 8.9: Exemplo de RNs montadas com uma população de módulos e *blueprints*.

Os vetores de de performance, $V_{performanceM}$ e $V_{performanceB}$, são inicializados com 7 e 3 células vazias, respectivamente:

$$V_{performanceM} = \_, \_, \_, \_, \_, \_, \_$$
$$V_{performanceB} = \_, \_, \_$$

Iterando sequencialmente por cada RN montada, começando por RA e terminando em RE, temos:

RA - 2 M2, 1 M5, B1. Acurácia $0,7$.

$$V_{performanceM} = \_, 2 \times (0.7), \_, \_, 0.7, \_, \_$$
$$V_{performanceM} = \_, 1.14, \_, \_, 0,7, \_, \_$$

$$V_{performanceB} = 0.7, \_, \_$$

RB - 1 M1, B2. Acurácia $0,1$.

$$V_{performanceM} = 0.1, 1.4, \_, \_, 0.7, \_, \_$$

$$V_{performanceB} = 0.7, 0.1, \_$$

RC - 1 M4, B3. Acurácia $0,3$.

$$V_{performanceM} = 0.1, 1.4, \_, 0.3, 0.7, \_, \_$$

$$V_{performanceB} = 0.7, 0.1, 0.3$$

RD - 1 M3, B2. Acurácia $0,25$.

$$V_{performanceM} = 0.1, 1.4, 0.25, 0.3, 0.7, \_, \_$$

$$V_{performanceB} = 0.7, (0.1 + 0.25), 0.3$$
$$V_{performanceB} = 0.7, 0.35, 0.3$$

RE - 2 M1, 1 M2, B1. Acurácia $0,8$.

$$V_{performanceM} = 0.1 + 2 \times (0.8), 1.4 + 0.8, 0.25, 0.3, 0.7, \_, \_$$
$$V_{performanceM} = 1.7, 2.2, 0.25, 0.3, 0.7, \_, \_$$

$$V_{performanceB} = 0.7 + 0.8, 0.35, 0.3$$
$$V_{performanceB} = 1.15, 0.35, 0.3$$

Ao final do término do cálculo da aptidão, teríamos os vetores $V_{performanceM}$ e $V_{performanceB}$ com os valores:

$$V_{performanceM} = 1.7, 2.2, 0.25, 0.3, 0.7, 0.0, 0.0$$

$$V_{performanceB} = 1.15, 0.35, 0.3,$$

onde os módulos M1 e M2, e os *blueprints* B1 e B2 obtiveram os maiores resultados, em suas respectivas populações.