Universidade Federal do ABC

Centro de Matemática, Computação e Cognição

Treinamento e Ajuste Topológico de Redes Neurais Artificiais Através de Estratégias Neuroevolutivas

Santo André – SP

Novembro/2021

Lucas Fernandes Muniz

Treinamento e Ajuste Topológico de Redes Neurais Artificiais Através de Estratégias Neuroevolutivas

Projeto de graduação em computação apresentado como requisito para a obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal do ABC.

Orientador: Prof. Dr. Denis Gustavo Fantinato

Coorientadora: Profa. Dra. Carla Negri Lintzmayer

Santo André – SP

Novembro/2021

Resumo

As Redes Neurais Artificiais (AAN, do inglês *Artificial Neural Network*) vêm alcançando notável desempenho em diversas aplicações. No contexto do aprendizado supervisionado, tradicionalmente, o treinamento dessas redes é realizado através do algoritmo de retropropagação. No entanto, esse algoritmo possui algumas limitações, dentre as quais ressalta-se sua alta taxa de convergência para soluções locais e a desvantagem de assumir uma topologia fixa para a ANN. Nesse âmbito, surge como promissora alternativa a abordagem chamada Neuroevolução, que considera o treinamento das ANNs por meio de algoritmos genéticos, permitindo que não apenas os pesos das ANNs sejam ajustados, mas também sua topologia. Em especial, o método NEAT (do inglês *NeuroEvolution of Augmenting Topologies*) tem mostrado relevante desempenho em problemas de aprendizado por reforço, o que é geralmente atribuído à sua estratégia de crescimento incremental a partir de uma topologia mínima e o mecanismo de especiação. Entretanto, no contexto do aprendizado supervisionado, o método NEAT ainda carece de uma análise mais aprofundada. Nesse sentido, a proposta desse trabalho é realizar uma análise comparativa sobre o desempenho do algoritmo de retropropagação e o método NEAT. Nessa análise, serão consideradas métricas advindas da Teoria da Informação para aferir de forma mais completa a eficiência das ANNs em termos de qualidade da predição final da rede.

Palavras-Chave: Redes Neurais Artificiais; Neuroevolução; Algoritmos Genéticos.

Sumário

1 Introdução

Nos últimos anos, algoritmos de aprendizado de máquina, como as redes neurais artificiais (ANN, do inglês *Artificial Neural Network*), têm sido utilizados para resolver problemas do mundo real em diferentes áreas [3, 8, 10]. Um dos modelos de redes neurais artificiais mais úteis em diversas aplicações é o perceptron de múltiplas camadas (MLP, do inglês *Multilayer Perceptron*) [3]. No contexto do aprendizado supervisionado, o treinamento de uma rede neural MLP geralmente é baseado no algoritmo de retropropagação do erro (*backpropagation*), que oferece uma maneira eficiente de utilizar as derivadas em relação aos parâmetros da rede, mesmo em suas camadas intermediárias. Porém, o treinamento através do algoritmo de retropropagação possui certas limitações, como o problema da exploração ineficiente do espaço de busca, a perda da capacidade de generalização da rede quando a mesma é treinada excessivamente (fenômeno conhecido como *overfitting*) e a dificuldade em se definir uma topologia (e.g., número de camadas e de neurônios) adequada para a rede [3, 10].

Nesse sentido, uma promissora abordagem aplicada principalmente no contexto de aprendizado por reforço se mostra capaz de superar algumas limitações do algoritmo de retropropagação do erro: o uso da técnica de evolução artificial dos pesos de uma rede neural artificial utilizando o Algoritmo Genético (GA, do inglês *Genetic Algorithm*) [4], denominada *neuroevolução* [15, 16].

Os algoritmos genéticos são metaheurísticas populacionais utilizadas na solução de problemas de otimização complexos, que se baseiam nas propriedades de cruzamento genético e Teoria da Evolução [4]. Nesses algoritmos, os processos de cruzamento, mutação e seleção, denominados como operadores genéticos, são responsáveis por modificar os indivíduos (representações da solução) da população de soluções, com o objetivo de melhorá-los para que atinjam um certo objetivo. A qualidade de um indivíduo (eficiência da solução em resolver um certo problema) é medida por uma função *fitness*, que guia a busca executada pela metaheurística, sendo maximizada ou minimizada durante as buscas. As metaheurísticas possuem capacidade de obtenção de boas soluções pois permitem balancear o processo de exploração (diversificação) e explotação (intensificação) de soluções em um espaço de busca, a partir do uso de uma função *fitness* adequada [4].

Apesar da neuroevolução permitir uma exploração mais eficiente do espaço de busca e

também lidar mais diretamente com a capacidade de generalização da rede devido ao seu processo de ajustes de pesos das ANNs, o processo de ajuste da topologia da rede ainda carecia de maior atenção. Com o objetivo de suprir essa lacuna, foi proposto o algoritmo *NeuroEvolution of Augmenting Topologies* (NEAT) [15], que permite a adaptação de pesos de forma concomitante à busca por topologias para a rede. Sua abordagem de construção incremental da rede privilegia topologias mais simples, auxiliando, portanto, na busca por estruturas mais eficientes, capazes de conciliar melhor desempenho e menor complexidade computacional. Por outro lado, alguns estudos recentes [13, 14, 17] buscam definir a eficiência de uma ANN sob a ótica da Teoria da Informação [7], usando medidas como a Entropia de Shannon e a Informação Mútua. Basicamente, esses trabalhos consideram que as camadas da ANN devem buscar um equilíbrio entre compressão e predição, de forma que a ANN aprenda a representar de maneira eficiente as informações mais relevantes contidas nos dados (compressão), sem que haja distorção ou perda significativa na predição.

Tendo em vista um considerável incremento no espaço de busca do NEAT, que abrange não apenas os pesos mas as diferentes topologias de ANN, surge um questionamento: o algoritmo NEAT consegue de forma efetiva superar as limitações do treinamento por retro-propagação do erro e ainda encontrar uma topologia eficiente? Neste trabalho, tentamos responder este questionamento a partir do estudo das vantagens e desvantagens do uso do NEAT como um método de treinamento de redes neurais artificiais.

Este texto está organizado na seguinte forma. A Seção 2 discute as justificativas para a execução deste projeto. Na Seção 3, os principais objetivos do projeto são apresentados. A Seção 4 contém a fundamentação teórica dos algoritmos e métodos utilizados, e também explica a interpretação das medidas de eficiência utilizadas. A Seção 5 explica detalhadamente como os algoritmos foram implementados. A Seção 6 exibe os resultados obtidos nas simulações com NEAT e o *backpropagation*. Na Seção 7, uma análise sobre os resultados das simulações é apresentada. E, finalmente, a Seção 8 descreve as conclusões obtidas neste projeto e as possibilidades de trabalhos futuros.

2 Justificativa

Como exposto anteriormente, as redes neurais artificiais, principalmente as MLP, possuem grande utilidade na solução de diversos problemas do mundo real[3, 8, 10]. O estudo de novas formas de treinamento de uma rede MLP é uma tarefa importante, pois pode produzir novos métodos que superassem as limitações do algoritmo de retropropagação do erro e, além disso, sejam capazes de ajustar a topologia da rede de forma eficiente. Nesse âmbito, o método de neuroevolução NEAT surge como uma promissora ferramenta que supre esses dois pontos. Entretanto, apesar de ter exibido bom desempenho em problemas de aprendizagem por reforço, no contexto do aprendizado supervisionado o NEAT ainda carece de uma análise mais aprofundada sobre suas reais vantagens em relação ao algoritmo de retropropagação. Baseando-se nisso, este projeto visa adaptar o algoritmo NEAT para ser utilizado no treinamento de redes neurais MLP em problemas de aprendizado supervisionado. Além disso, visamos estudar as características dos métodos de treinamento por retropropagação e pelo NEAT a partir do uso de medidas da Teoria da Informação, que permitirão avaliar a eficiência das redes treinadas pela análise da eficiência de suas camadas.

A proposta desse trabalho é realizar um estudo comparativo aprofundado entre o algoritmo de retropropagação e o método NEAT no contexto de aprendizado supervisionado. Espera-se que o algoritmo de neuroevolução forneça uma maneira mais eficiente de explorar a superfície de erro, apresentando maior taxa de convergência para mínimos globais (uma das limitações do treinamento por retropropagação do erro). No entanto, por também realizar uma busca de uma topologia eficiente, o NEAT pode estar sujeito a uma perda de desempenho, devido ao crescente espaço de busca. A análise da eficiência das redes será executada a partir do uso de conceitos da Teoria da Informação, mais especificamente da informação mútua [7] e do plano de informação [14], pois fornecem uma maneira bastante rica para a avaliação da rede camada a camada. Para teste, uma seleção representativa de problemas de aprendizado supervisionado (classificação) foi obtida a partir de dados disponíveis em bases públicas[1].

[1] Como UCI e Kaggle.

3 Objetivos

A execução dessa pesquisa teve como base três objetivos. O primeiro deles foi a realização de um estudo teórico aprofundado sobre as MLPs e GAs, incluindo também o estudo das técnicas clássicas de treinamento das ANNs, como o algoritmo de retropropagação. Houve também o estudo do treinamento de redes por neuroevolução, com foco no funcionamento do método NEAT, que possui a capacidade de evoluir topologias ao longo das gerações. Essa fase envolveu o estudo de uma seleção de livros e artigos que tratam, de forma aprofundada e bastante atual, dos métodos utilizados, e contou com o acompanhamento do orientador e da coorientadora através de reuniões semanais para a discussão dos tópicos.

O segundo objetivo esteve ligado ao estudo dos conceitos de Teoria da Informação, cujo entendimento foi importante para a análise adequada da eficiência de toda a rede em termos da complexidade topológica e desempenho (visto que as medidas de erro tradicionais avaliam, de forma limitada, apenas a saída final da rede).

Sob uma perspectiva mais prática, o terceiro objetivo visou a implementação da rede neural MLP, e dos algoritmos GA e NEAT através da linguagem de programação *Julia*. A meta era que o aluno adquirisse experiência como projetista de redes neurais artificiais e de metaheurísticas, sendo tal experiência alcançada através da comparação do desempenho dos métodos estudados. Pretendeu-se atingir um ganho de desempenho em relação ao método de retropropagação utilizando o NEAT como método de treinamento.

4 Fundamentação Teórica

O presente projeto fez uso de três elementos principais, que são o perceptron de múltiplas camadas, os algoritmos genéticos e o método NEAT. Estes serão descritos na sequência.

4.1 O *Perceptron* de Múltiplas Camadas

Uma ANN é um modelo computacional que possui o objetivo de modelar uma função que mapeia vetores de entradas para certos vetores de saída. Ela é composta por um conjunto de $t \geq 3$ camadas, sendo uma camada de entrada, uma ou mais camadas intermediárias (ou ocultas) e uma camada de saída. A ℓ-ésima camada é formada por um conjunto com n_ℓ nós, denominados *neurônios artificiais*. Na abordagem totalmente conectada, cada neurônio de uma camada irá se conectar com cada neurônio da camada seguinte, sendo cada conexão ponderada por um peso. Formalmente, o j-ésimo neurônio da camada ℓ se conecta ao k-ésimo neurônio da camada $\ell + 1$ com uma conexão ponderada pelo valor $w_{jk}^{(\ell+1)}$, para todo $1 \leq j \leq n_\ell$, $1 \leq k \leq n_{\ell+1}$ e $1 \leq \ell < t$. Além disso, o j-ésimo neurônio da ℓ-ésima camada possui um valor $z_j^{(\ell)}$ associado, que indica a sua *saída* potencialmente não linear. Na camada de entrada, os neurônios apenas recebem o dado de entrada $\mathbf{x} \in \mathbb{R}^{n_1}$, ou seja, $\mathbf{z}^{(1)} = \mathbf{x}$. Essa informação é propagada camada a camada pela rede até gerar a saída da rede e, por esta propriedade, este tipo de rede é chamado de *feedforward*. A Figura 1 ilustra o modelo geral de uma ANN *feedforward* [10] em que o sinal de saída de uma camada é apenas transmitido para a próxima camada.

A propagação do dado de entrada em uma ANN, em que o sinal de uma camada é transmitido estritamente para a camada seguinte, pode ser descrita por uma série de transformações funcionais, em que a saída da ℓ-ésima camada, para $2 \leq \ell \leq t$ é dada por

$$\mathbf{z}^{(\ell)} = \mathbf{h}^{(\ell)} \left(\mathbf{W}^{(\ell)} \mathbf{z}^{(\ell-1)} \right) \tag{1}$$

sendo $\mathbf{z}^{(\ell)} = [z_1^{(\ell)} \ z_2^{(\ell)} \ \dots \ z_{n_\ell}^{(\ell)}]^T$ o vetor com as saídas dos neurônios da ℓ-ésima camada, $\mathbf{W}^{(\ell)}$ a matriz cujos elementos da j-ésima linha representam os pesos sinápticos do j-ésimo neurônio na ℓ-ésima camada, e $\mathbf{h}^{(\ell)}(\cdot) = [h_1^{(\ell)}(\cdot) \ h_2^{(\ell)}(\cdot) \ \dots \ h_{n_\ell}^{(\ell)}(\cdot)]^T$ um conjunto de funções univariadas chamadas de *funções de ativação*. As funções de ativação são aplicadas elemento

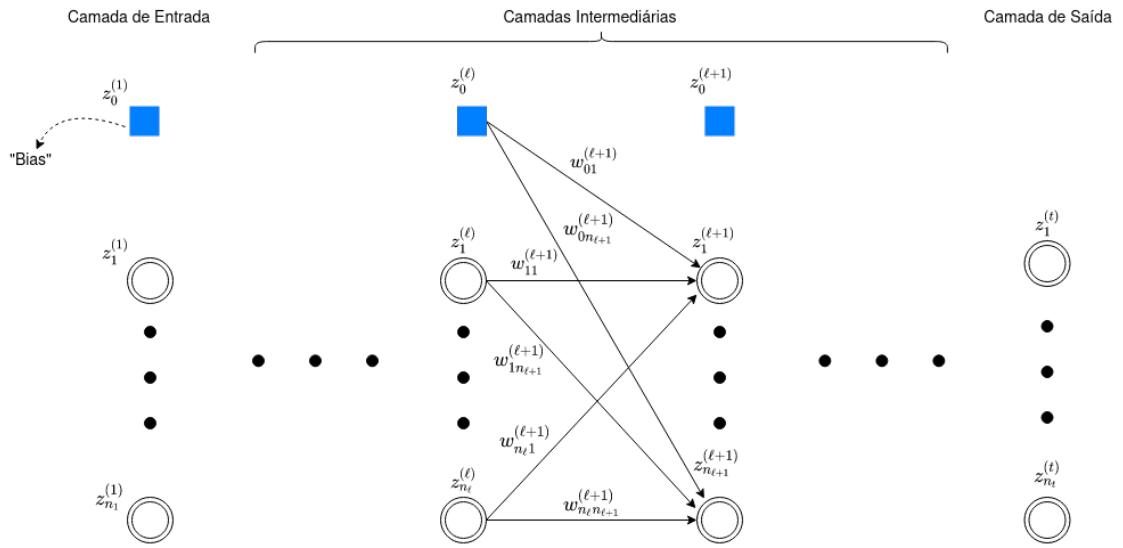


Figura 1: Modelo de uma rede neural feedforward com t camadas, com cada neurônio sendo representado por um círculo duplo. Os quadrados azuis representam o termo de bias, com $z_0^{(\ell)} = 1$ *para toda camada* ℓ*. Cada conexão, representada por uma seta, tem um peso associado.*

a elemento no vetor resultante do produto matricial $\mathbf{W}^{(\ell)}\mathbf{z}^{(\ell-1)}$ [3, 10]. Geralmente, um termo de *bias* é incluso na primeira coluna da matriz de pesos $\mathbf{W}^{(\ell)}$ e incrementa-se a $\mathbf{z}^{(\ell)}$ um elemento com valor unitário fixo na primeira posição do vetor (i.e., $z_0^{(\ell)} = 1$).

Em relação às funções de ativação $h(\cdot)$, geralmente são utilizadas a tangente hiperbólica e a *Rectified Linear Unit* (ReLU), que é dada por $h(x) = \max\{0, x\}$ [1, 9, 10, 18]. Em problemas de classificação, a camada de saída usualmente tem como função de ativação a softmax [9], enquanto que, em problemas de regressão, usa-se uma função linear para não restringir os valores de saída [9].

Nesse modelo de ANN, cada neurônio se comporta como um *perceptron* [3, 10], que é um modelo de discriminante linear, e por essa razão esse modelo é conhecido como um *perceptron multicamadas* ou *perceptron de múltiplas camadas* (MLP, do inglês *Multilayer Perceptron*). As redes MLP exibem um alto grau de *conectividade*, cuja intensidade é determinada e ajustada pelos pesos sinápticos da rede [10]. Assim, as redes MLP são diferenciáveis com respeito aos parâmetros da rede, e essa propriedade possui um papel central no treinamento da rede [3].

4.1.1 Treinamento e Retropropagação do Erro

No aprendizado supervisionado, a entrada de uma rede MLP é representada por um vetor $\mathbf{x}$ denominado vetor de entrada, enquanto que os valores desejados (ou de referência) para a

saída da rede são representados em um vetor $\mathbf{y}$, denominado *saída desejada*. Durante o treinamento, utilizamos um conjunto de treinamento com N amostras, cuja i-ésima amostra é uma tupla $(\mathbf{x}_i, \mathbf{y}_i)$, em que $\mathbf{x}_i$ é um vetor de entrada e $\mathbf{y}_i$ é o vetor de saída desejado. Para calcular a diferença entre o vetor de saída de uma rede neural e o vetor de valores desejados, utiliza-se uma função custo (ou *loss function*) $E(\boldsymbol{\theta})$, em que $\boldsymbol{\theta}$ é o conjunto de todos os parâmetros ajustáveis da ANN. Geralmente, o custo utilizado é o *Erro Quadrático Médio* ou a *Entropia Cruzada* [9, 10]. Durante o treinamento da rede, nosso objetivo é encontrar os parâmetros $\boldsymbol{\theta}$ que minimizam a função $E(\boldsymbol{\theta})$ escolhida.

Devido à característica não linear da MLP, haverá muitos pontos estacionários (pontos críticos) com mesmos ou distintos valores, bem como muitos pontos mínimos com mesmos ou distintos valores em $E(\boldsymbol{\theta})$. No entanto, o ponto ótimo é aquele associado ao *mínimo global* da função custo.

Como não há uma maneira analítica de obter a solução de $\nabla E(\boldsymbol{\theta}) = 0$ (devido ao comportamento não linear da rede), métodos iterativos são utilizados. O método mais simples de utilizar a informação do gradiente para encontrar pontos mínimos na função de erro é, a partir da inicialização aleatória dos parâmetros $\boldsymbol{\theta}^{(0)}$, escolher a atualização de parâmetro que executa pequenos passos na direção oposta (negativa) ao do gradiente, em que

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta \nabla E(\boldsymbol{\theta}^{(\tau)}) \,, \tag{2}$$

onde o parâmetro $\eta > 0$ é chamado de *taxa de aprendizagem* e τ é o passo da iteração. A ideia é que, a cada passo, os parâmetros se movam na direção da maior taxa de decrescimento da função de erro e, por isso, esse algoritmo é conhecido como *gradiente descendente*.

Entretanto, para o treinamento da MLP, é necessário o uso de um método eficiente para calcular o gradiente da função de erro $E(\boldsymbol{\theta})$ considerando os parâmetros contidos dentro das camadas intermediárias, e um dos métodos mais usados é o de retropropagação do erro, ou *error backpropagation* [3, 10]. Esse método utiliza um esquema de passagem de informação local em que, em um primeiro passo, o sinal de entrada é propagado até a saída da rede, gerando o erro ao final e, em um segundo passo, as derivadas do erro em relação aos pesos sinápticos são calculadas e retropropagadas para as camadas intermediárias através do uso da regra da cadeia (por isso é denominado *backpropagation*) [3].

4.1.2 Desvantagens do Algoritmo de Retropropagação do Erro

Apesar do *error backpropagation* ser um algoritmo computacionalmente eficiente, ele possui algumas desvantagens no treinamento. A mais direta delas está associada à alta frequência de convergência a mínimos locais. Apesar de existirem técnicas, como o uso do Jacobiano e da matriz Hessiana [10], que são capazes de aumentar a convergência para melhores mínimos, estas ainda se mostram insuficientes. Muitas vezes, utilizam-se redes com topologias mais sofisticadas para que seus mínimos locais atinjam bom desempenho, mas há uma evidente redução da eficiência da rede em termos de seu aumento na complexidade computacional [9].

Outra desvantagem é que, dependendo da função de ativação escolhida, as camadas finais possuem gradientes locais maiores do que as camadas iniciais, que possuem gradientes locais menores [9]. Dessa forma, diferentes camadas da rede possuem velocidade de aprendizado diferentes. Isso ocorre pois o valor do gradiente vai diminuindo conforme o erro é retropropagado na rede, sendo este efeito conhecido como *dissipação do gradiente* [10].

Um outro problema é a inicialização dos parâmetros, pois o valor mínimo encontrado pelo gradiente descendente dependerá da posição inicial dos parâmetros da rede na superfície de erro [3]. Como a função de erro geralmente possui vários mínimos (locais ou globais), para encontrar mínimos suficientemente bons pode ser necessário executar os algoritmos baseados no gradiente múltiplas vezes, utilizando diferentes pontos iniciais aleatórios para que os resultados sejam comparados em um conjunto independente de dados de validação.

Outra limitação do *error backpropagation* é o fato da minimização do erro pelo gradiente descendente não fornecer nenhuma informação sobre o treinamento excessivo da rede e a perda da sua capacidade de *generalização*. Uma rede neural possui o objetivo de modelar uma função que mapeia vetores de entradas para certos vetores de saída. Logo, uma rede terá boa capacidade de generalização se o seu mapeamento for correto para dados que não foram utilizados no processo de treinamento [10]. O algoritmo *error backpropagation* sempre irá diminuir o erro da rede no conjunto de treinamento, mas quando há um excesso de treinamento, é possível que a rede memorize o mapeamento, perdendo assim a capacidade de generalização, já que a rede não foi capaz de obter informações sobre a distribuição que estava contida nos exemplos de treinamento. Esse fenômeno é conhecido como *overfitting*. Para evitá-lo, é necessário utilizar um conjunto de validação para avaliar o erro da rede, sendo que esse conjunto possui dados que não foram utilizados no treinamento. O desempenho da

rede no conjunto de validação permite identificar quando a rede está perdendo sua capacidade de generalização e o momento em que o treinamento deve ser encerrado [3, 9, 10].

Por fim, mas não menos importante, temos a desvantagem do algoritmo de retropropagação assumir uma topologia fixa para a MLP e não oferecer informações relevantes para permitir seu correto ajuste. Essa etapa é geralmente tratada a partir da definição de um conjunto de topologias de MLP, seguidas de um teste exaustivo para identificar aquela capaz de apresentar o melhor desempenho e, de forma conveniente, a menor complexidade topológica.

Dessa forma, a neuroevolução emerge como uma poderosa ferramenta para tentar superar essas limitações observadas com o algoritmo de retropropagação [15, 16]. Um elemento chave nesse método é o uso do Algoritmo Genético, descrito a seguir.

4.2 Algoritmos Genéticos e a Neuroevolução

Desde sua criação em 1970, os Algoritmos Genéticos (GA, do inglês *Genetic Algorithm*) se tornaram uma das metaheurísticas populacionais mais usadas para a solução de problemas de otimização complexos [4]. Eles se baseiam nas propriedades definidas em cruzamento genético e na Teoria da Evolução, de forma que os operadores genéticos são responsáveis pelo cruzamento, mutação e seleção de indivíduos (soluções). A aplicação dos operadores sobre uma população de soluções é responsável por modificá-la, gerando soluções melhores.

A busca de soluções sobre o espaço de busca é guiada por uma função *fitness*, responsável por avaliar o desempenho de cada indivíduo, que é minimizada ou maximizada durante as gerações. Em cada geração, a partir de uma população de indivíduos, a função *fitness* de cada indivíduo é avaliada e os melhores são selecionados para o cruzamento. Após o cruzamento, o indivíduo passa pelo processo de mutação, em que parte dele pode ser modificada aleatoriamente. Por último, os indivíduos passam por um processo de seleção para que os indivíduos mais adaptados possam permanecer na próxima geração. Os operadores genéticos (etapas de cruzamento, mutação e seleção) são aplicados com o objetivo de evitar que a busca convirja para um ótimo local no espaço de busca [11]. O Algoritmo 1 ilustra um pseudocódigo com os principais passos do algoritmo genético.

Por não depender da informação do gradiente, os GAs podem apresentar maior frequência de convergência global, já que o GA apresenta a característica de explorar o espaço de busca e intensificar a busca em soluções promissoras, enquanto que as técnicas baseadas

Algoritmo 1: Pseudocódigo de um algoritmo genético, onde n_G representa o número de gerações (iterações) e t_p o tamanho da população de soluções.

Entrada: t_p, n_G

```
P = GERAR_POPULAÇÃO(t_p)
para i = 1 até n_G faça
    F = CRUZAMENTO(P)
    P' = P ∪ F
    P' = MUTAÇÃO(P')
    P = SELEÇÃO(P')
devolve P
```

no gradiente otimizam a solução seguindo uma única direção. Isto motivou seu uso para o treinamento das ANNs, de forma que os indivíduos representassem redes neurais, dando origem à neuroevolução. Isto também permitiu que as desvantagens do efeito de *dissipação do gradiente* e da inicialização dos pesos da rede (graças à abordagem populacional) fossem sanadas diretamente. Além disso, como o ajuste é feito a partir da busca de diversas soluções que minimizam o erro do conjunto de dados e não apenas em uma única direção que minimiza o erro (como no algoritmo de retropropagação), a possibilidade de ocorrência de *overfitting* diminui.

Mesmo com essas vantagens, o problema do ajuste da topologia ainda persistia. Assim, surgiram as primeiras abordagens que combinavam o ajuste dos pesos da ANN e também de sua topologia [2]. Abordagens iniciais partiam de uma população inicial formada por topologias aleatórias [5, 6], porém as topologias presentes na população possuíam parâmetros (neurônios e conexões) desnecessários, além do fato de que topologias grandes aumentavam o número de parâmetros a serem buscados, de forma que o tamanho das topologias permitidas acabava sendo limitado [15]. Outra abordagem adotada inicialmente foi considerar que sub-redes representassem unidades funcionais que poderiam ser recombinadas durante o cruzamento, porém diferentes topologias que não fossem baseadas nas mesmas sub-redes não podiam ser recombinadas [12, 15]. O *NeuroEvolution of Augmenting Topologies*, descrito a seguir, foi desenvolvido com o objetivo de resolver esses problemas.

4.3 NEAT

O *NeuroEvolution of Augmenting Topologies* (NEAT) é um algoritmo de neuroevolução, geralmente utilizado em tarefas de aprendizado por reforço [15, 16]. Dado um determinado problema a ser resolvido com o uso de uma rede neural, esse algoritmo busca uma rede para esse problema evoluindo sua estrutura (topologia) junto com os pesos, a partir do uso do GA. Durante a busca, o desempenho das redes presentes na população é calculado avaliando o desempenho da rede no problema a ser resolvido, de forma que o algoritmo tenta buscar uma rede que possua uma topologia mínima e que resolva o problema de forma mais eficiente possível. O NEAT evolui as redes neurais otimizando e complexificando-as simultaneamente, semelhante ao processo de evolução biológica, sendo que esta capacidade do NEAT de buscar topologias é uma característica que o difere dos métodos de neuroevolução tradicionais, que costumam utilizar topologias fixas.

Para efetuar de forma eficiente a busca, o NEAT minimiza a dimensionalidade do espaço de busca enviesando-a em direção ao espaço de busca mínimo, partindo de uma população uniforme inicial de topologias com nenhum neurônio intermediário (as entradas são conectadas diretamente com a saída). Novas estruturas são adicionadas gradualmente pelas mutações estruturais, e as estruturas que sobrevivem são as que se mostraram úteis através da avaliação do seu valor de *fitness*. Assim, as estruturas que ocorrem no NEAT são justificadas, pois uma estrutura apenas permanecerá na população caso ela produza soluções melhores. Como a população parte de uma estrutura mínima, a dimensionalidade do espaço de busca também é minimizada, o que garante melhor desempenho na execução da busca [15].

4.3.1 Codificação Genética

A codificação genética do NEAT é direta, ou seja, cada neurônio e conexão da rede é especificado no genoma (indivíduo). Os genomas são representações lineares da conectividade da rede, sendo formados por uma lista de *genes de conexão*, que especifica a conexão entre dois *genes de neurônio*, e uma lista de genes de neurônios que estão presentes na lista de genes conexão. Um gene de neurônio é especificado por um número de identificação do gene, a localização do neurônio na rede e o seu tipo (entrada, intermediário ou saída). Um gene de conexão especifica os dois neurônios que ele conecta, o peso de conexão, se a conexão é

expressada (bit de habilitação) e um *número de inovação*, que marca o momento em que a conexão foi criada (*origem histórica*) e permite detectar genes correspondentes. Note que os indivíduos do NEAT nem sempre serão redes totalmente conectadas e ele permite conexão entre camadas não consecutivas. A Figura 2 mostra um exemplo de codificação genética do NEAT e a sua interpretação como uma rede neural (fenótipo) [15].

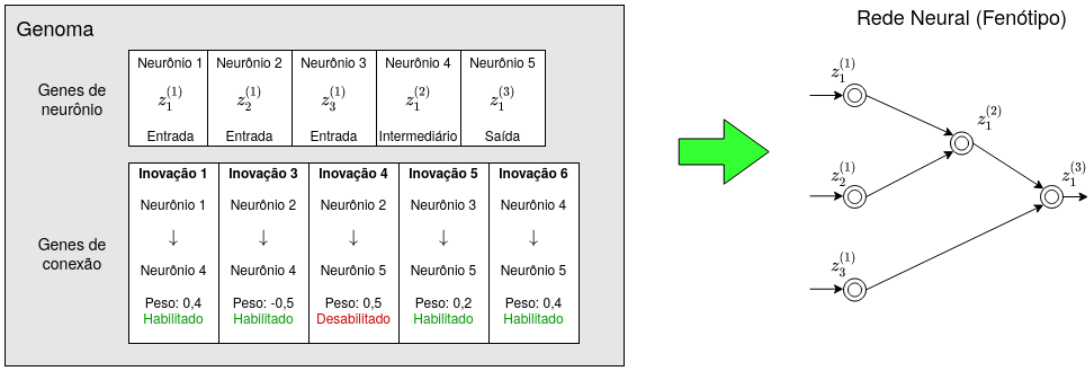


Figura 2: Exemplo de uma codificação genética utilizada pelo NEAT.

4.3.2 Mutação

A mutação no NEAT pode mudar ambos os pesos e a estrutura da rede. A mutação de peso pode ser uma perturbação adicionada ou não a cada geração. Já a mutação estrutural pode ocorrer de duas formas: a mutação de *adição de conexão* e a mutação de *adição de neurônio*, sendo ambas mutações que expandem o tamanho do genoma adicionando genes. Na mutação de adição de conexão, um gene de conexão é adicionado conectando dois neurônios que não estavam conectados. Na mutação de adição de neurônio, uma conexão é separada e um novo neurônio é alocado no lugar da conexão. Formalmente, se uma antiga conexão conectava os neurônios i e j, então ela é desabilitada e duas novas conexões são adicionadas ao genoma, sendo que uma conexão sai de i e vai para o novo nó possuindo peso 1, e a outra conexão sai do novo nó e atinge j possuindo o peso da conexão antiga. Esse método de adição de nó tem por objetivo minimizar o efeito da mutação. Em geral, quando novas estruturas são adicionadas, o desempenho da rede neural tende a cair pois a nova estrutura foi gerada aleatoriamente e não teve tempo de ser otimizada. Mas, devido à especiação do NEAT, a rede neural terá tempo de otimizar e utilizar a nova estrutura adicionada na mutação, pois ela competirá com indivíduos semelhantes e não será perdida caso alguma rede de outra espécie seja melhor. As Figuras 3 e 4 ilustram as mutações de adição de neurônio e conexão, respectivamente. Neste

projeto, para garantir que a população seja formada apenas por redes MLP, não foi permitida a formação de conexão no mesmo neurônio (recursão) e nem a formação de conexões que vão para um neurônio localizado na mesma camada ou em uma camada anterior, de modo que os neurônios de uma determinada camada só se conectam com os neurônios da próxima camada.

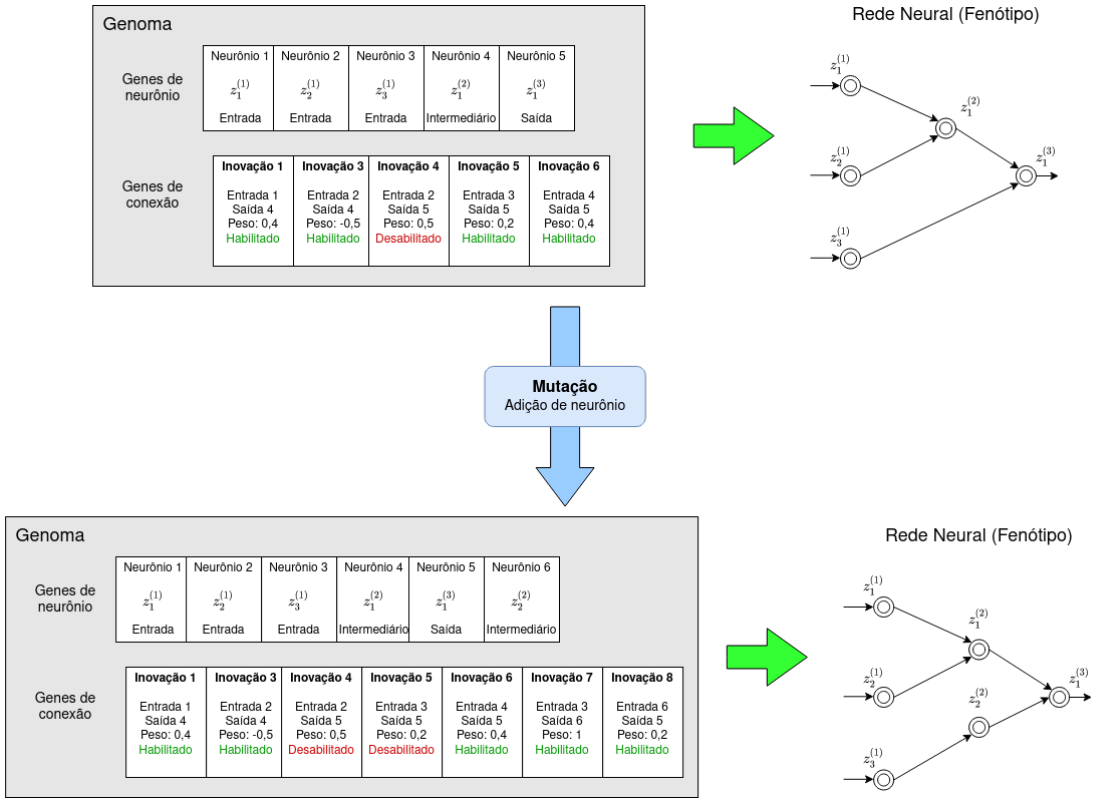


Figura 3: Aplicação de uma mutação de adição de neurônio no indivíduo da Figura 2.

Como os pesos e topologias crescem sem restrições, é possível gerar redes iguais com diferentes representações genéticas, o que dificulta o cruzamento entre os genes que representam a mesma estrutura (genes correspondentes) em duas soluções diferentes. Porém, no NEAT este problema é resolvido pelo número de inovação, que permite identificar os genes correspondentes entre indivíduos, imitando o conceito de genes homólogos na biologia [15]. Dois genes com a mesma origem histórica representam a mesma estrutura (embora possivelmente possuam pesos diferentes), pois derivam do mesmo gene ancestral em alguma geração passada. É importante notar que o número de inovação não leva em consideração a posição da conexão na topologia da rede, pois ele apenas representa o momento em que a conexão (nova estrutura) foi adicionada. Logo, a conexão pode mudar sua posição na topologia, devido às outras mutações que ocorreram ao longo das gerações, mas o número de inovação não

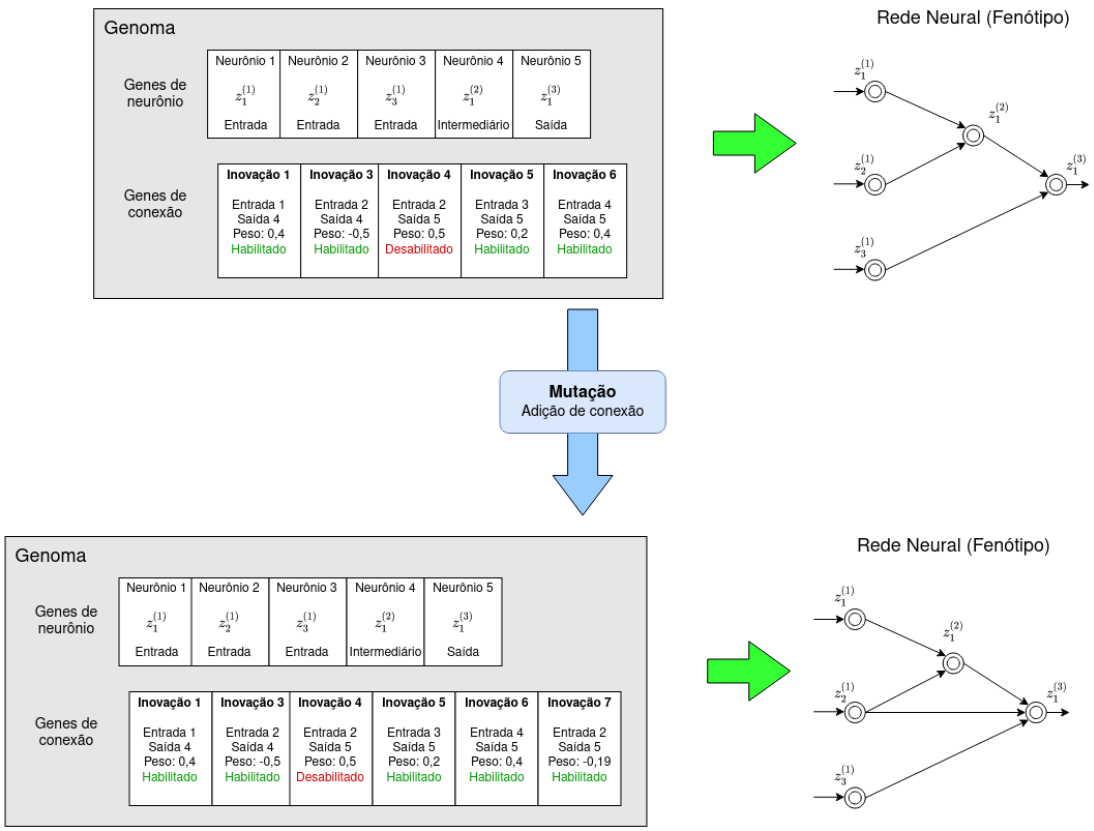


Figura 4: Aplicação de uma mutação de adição de conexão no indivíduo da Figura 2.

será modificado. Sempre que um novo gene aparece (através de uma mutação estrutural), o *número de inovação global* é incrementado e atribuído ao gene, sendo o número de inovação a representação da cronologia de aparição dos genes. Para manter a consistência e evitar um grande crescimento do número de inovação, mutações estruturais que geraram a mesma estrutura em uma mesma geração irão possuir a mesma marcação histórica (número de inovação).

4.3.3 Cruzamento

Para o cruzamento, dois indivíduos são escolhidos e os genes que possuem a mesma marcação histórica deles são alinhados (genes correspondentes), enquanto que os genes que não possuem correspondência são classificados como *disjuntos* ou *excessivos*, dependendo do local de sua ocorrência no intervalo do número de inovação dos genes correspondentes. Os genes disjuntos são os genes que possuem número de inovação menor que o maior número de inovação do conjunto de genes correspondentes, enquanto que os genes excessivos possuem possuem número de inovação maior que o maior número de inovação dos genes correspondentes.

Na composição do filho, genes são escolhidos aleatoriamente de um dos pais no conjunto de genes correspondentes, enquanto que os genes disjuntos e excessivos do genitor de maior desempenho, avaliado por uma função *fitness*, são passados para o filho. Na Figura 5 é possível ver um esquema do cruzamento de dois indivíduos (pai 1 e pai 2) no NEAT.

4.3.4 Especiação

A adição de novos genes e o cruzamento de diferentes estruturas permite a produção de uma população topologicamente diversificada. Porém, tal população não consegue manter inovações topológicas, pois estruturas menores otimizam mais rápido do que as maiores e adições estruturais inicialmente diminuem o desempenho de uma ANN, diminuindo as chances de sobrevivência das inovações, mesmo que sejam cruciais para o desempenho da rede.

A solução encontrada pelo NEAT foi a proteção das inovações através da *especiação* da população. No NEAT, cada espécie terá um processo de evolução independente, regido pelos operadores genéticos do GA. A estratégia de especiação permite que os indivíduos compitam com outros indivíduos do mesmo "nicho", ou seja, com topologias parecidas, em vez de uma disputa com toda a população. Desta forma, inovações topológicas são protegidas em um novo nicho, que oferece tempo para a otimização da estrutura através da competição dentro do seu nicho. A população é dividida em espécies, sendo que redes que possuem topologias similares estarão na mesma espécie. A definição de uma espécie para um indivíduo é dada a partir da distância de compatibilidade entre um indivíduo e o representante de uma dada espécie. A distância de compatibilidade é definida pela história evolucionária dos genomas (indivíduos), que é obtida pelo número de genes de mesma origem histórica nos genomas. Se dois genomas possuem uma grande quantidade de genes disjuntos ou excessivos, então eles compartilham pouca história evolucionária e vice-versa. Formalmente, a distância de compatibilidade δ entre duas redes neurais i e j no NEAT é dada pela combinação linear do número de genes excessivos entre i e j, denotada E_{ij}, número de genes disjuntos entre i e j, denotado D_{ij}, e a média da diferença dos pesos dos genes correspondentes (incluindo os genes desabilitados) entre i e j, denotado $\overline{W}_{ij}$, da seguinte forma:

$$\delta(i,j) = \frac{c_1 E_{ij}}{N} + \frac{c_2 D_{ij}}{N} + c_3 \overline{W}_{ij}\,, \tag{3}$$

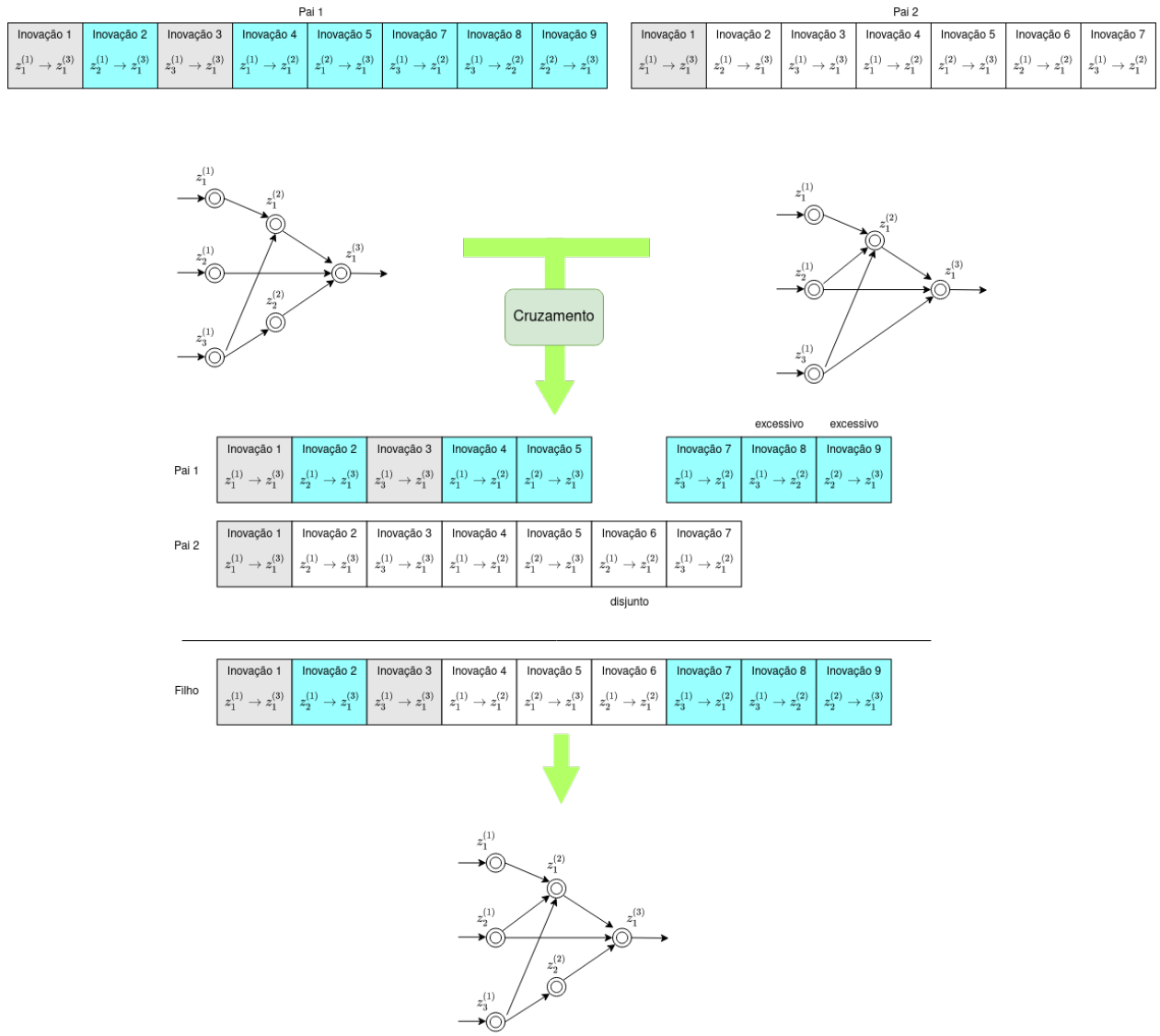


Figura 5: Cruzamento de dois indivíduos no NEAT. Os genes cinzas estão desabilitados, os genes azuis pertencem ao pai 1 e os brancos ao pai 2. Considerando que neste caso o pai 1 e o pai 2 possuem o mesmo desempenho, os genes disjuntos e excessivos são passados aleatoriamente para o filho.

onde os coeficientes c_1, c_2 e c_3 são utilizados para ajustar a importância de cada fator, e N é o número de genes do maior genoma. A medida δ permite a especiação a partir de um limiar δ_t. Durante as gerações, uma lista ordenada de espécies é mantida, sendo cada espécie representada por um genoma aleatório da espécie na geração anterior. A cada geração, genomas são sequencialmente alocados na primeira espécie em que a distância de compatibilidade entre o genoma e o representante da espécie é menor que o limiar δ_t. Caso um genoma não seja compatível com nenhuma espécie, uma nova espécie será gerada contendo esse genoma como representante.

O mecanismo de reprodução do NEAT utiliza o compartilhamento explícito de genes, tal que indivíduos de uma mesma espécie compartilham o *fitness* do seu nicho. Dessa forma, mesmo que uma espécie possua um desempenho muito bom, ela não dominará a população pois não poderá se tornar muito grande. O *fitness* ajustado f'_i de um indivíduo i é calculado de acordo com sua distância δ para cada indivíduo j da população, calculada da seguinte forma:

$$f'_i = \frac{f_i}{\sum_{j=1}^{n} \mathrm{sh}(\delta(i,j))}\,, \tag{4}$$

em que sh é a função de compartilhamento, que é 0 quando $\delta(i,j) > \delta_t$, e é 1 caso contrário. Dessa forma, $\sum_{j=1}^{n} \mathrm{sh}(\delta(i,j))$ representa o número de indivíduos que pertencem à mesma espécie de i. Cada espécie irá gerar um determinado número de prole em proporção à soma dos pesos ajustados f'_i de seus indivíduos membros. As espécies reproduzem eliminando os membros de menor desempenho e gerando a prole com os indivíduos restantes da espécie, sendo que toda a população é substituída pela prole gerada por cada espécie.

4.4 Medidas de Informação aplicadas às Redes Neurais Artificiais

A fim de estudar como a informação da entrada e da saída de uma rede neural é transformada pelo modelo, neste projeto foram usados alguns conceitos oriundos da Teoria da Informação, como a informação mútua e o plano de informação. Na Teoria da Informação, intuitivamente, a informação mútua entre duas variáveis aleatórias X e Z, representada por $I(X,Z)$, busca medir a quantidade de informação que a variável X traz para Z. Matematicamente, seja $p(X,Z)$ a distribuição conjunta e $p(X)$ e $p(Z)$ as distribuições marginais associadas às

variáveis aleatórias X e Z. A informação mútua é dada por

$$I(X, Z) = \int_x \int_y p(x, z) \log \frac{p(x, z)}{p(x)p(z)} dx \, dz \, . \tag{5}$$

Com base nela, é possível estabelecer uma medida de eficiência das camadas de uma rede neural.

Vamos associar à variável X a distribuição dos dados de entrada $\mathbf{x}$ da MLP, à Y a distribuição dos valores desejados de saída $\mathbf{y}$ e à Z a distribuição dos sinais de saída $\mathbf{z}^{(\ell)}$ de uma dada camada ℓ da rede. Interpretando uma rede neural como uma cadeia de Markov de representações sucessivas da camada de entrada, é possível visualizar como a informação da entrada e da saída é distribuída em cada camada da rede utilizando o plano de informação, que é o gráfico da informação mútua entre uma camada e a variável de entrada X e a de valores desejados Y. Assim, para uma rede com t camadas, podemos representar o plano de informação da rede utilizando $t - 1$ pontos monotônicos conectados [14], sendo que para uma camada representada pela variável Z, a posição da camada no plano de informação será dada pelas coordenadas $(I(X, Z), I(Z, Y))$. O valor $I(X, Z)$ diz respeito a compressão eficiente da informação de entrada, enquanto que $I(Z, Y)$ refere-se à qualidade da saída da camada (predição). Dessa forma, uma camada será considerada eficiente se conseguir eliminar o ruído dos dados de entrada e capturar apenas a informação útil para a predição final, ou seja, se possuir baixo valor de $I(X, Z)$ e alto valor de $I(Z, Y)$. A Figura 6 ilustra o plano de informação de uma rede neural com quatro camadas, sendo uma de entrada, duas intermediárias e uma de saída. Observe que, no plano de informação, a única camada não representada é a de entrada. Para facilitar a visualização, as camadas intermediárias são simbolizadas por um círculo, enquanto que a camada de saída é simbolizada por uma estrela.

Ravid Shwartz-Ziv e Naftali Tishby [14] mostraram que o treinamento de uma MLP pela otimização do gradiente descendente possui duas fases distintas. Na primeira e mais curta fase, as camadas aumentam a informação sobre os rótulos desejados, ou seja, ocorre um aumento do termo $I(Z, Y)$. Já na segunda e mais longa fase, as camadas diminuem a informação sobre as entradas, ou seja, ocorre uma diminuição do termo $I(X, Z)$. Dessa forma, o treinamento de uma rede tende a deslocar os termos associados às camadas da rede para o canto superior esquerdo do plano de informação. Ao comparar o plano de informação

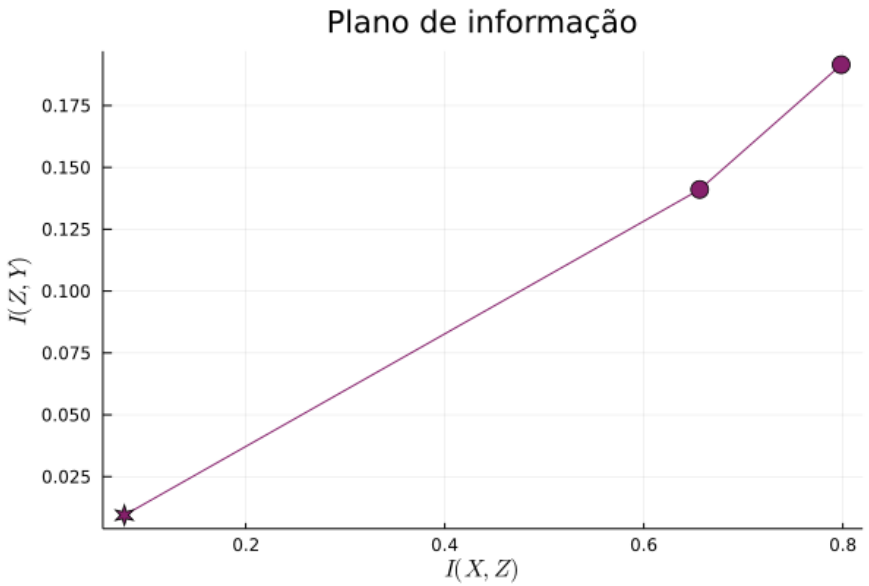


Figura 6: Representação de uma rede neural com quatro camadas no plano de informação, onde os círculos representam as camadas intermediárias e a estrela representa a camada de saída.

das redes treinadas pelo *backpropagation* e pelo NEAT, poderemos avaliar se o treinamento pelo NEAT possui ou não o mesmo comportamento do treinamento pelo *backpropagation*, permitindo-nos observar as possíveis vantagens ou limitações do NEAT.

5 Implementação

Nesta seção, serão descritos os detalhes de implementação dos algoritmos deste projeto[2]. Todos os algoritmos deste projeto foram implementados na linguagem de programação Julia[3], que é adequada para computação científica pois possui alto desempenho computacional.

5.1 Conjunto de Dados

Para este projeto, foram selecionados quatro conjuntos de dados de problemas de classificação para o treinamento das MLPs utilizando *backpropagation* e o NEAT. Para o treinamento, 80% dos dados foram separados aleatoriamente como conjunto de treinamento e 20% como conjunto de teste. O primeiro conjunto de dados ("Drug Classification") possui 200 amostras e é referente a um problema de classificação de medicamentos[4]. Cada amostra representa 5 características de um paciente e a rede precisa predizer o medicamento adequado ao paciente, entre 5 medicamentos disponíveis (5 classes). O segundo conjunto de dados ("Income Classification"), que possui 3000 amostras, é referente ao problema de classificação da renda anual de uma pessoa[5]. Cada amostra possui 13 características de uma pessoa, sendo que a rede deve classificar se a renda anual dessa pessoa é maior que $50,000 (2 classes). O terceiro conjunto de dados ("Wine Quality"), que possui 6497 amostras, é referente a um problema de classificação da qualidade de vinhos[6]. Cada amostra possui 11 características de um vinho e a rede deve predizer uma nota de 0 a 10 para a qualidade do vinho (11 classes). O quarto conjunto de dados ("Drug Consumption"), referente a um problema de classificação do perfil de consumo de bebida alcoólica de uma pessoa[7], possui 1885 amostras. Cada amostra possui 12 características de uma pessoa e cada pessoa é classificada em 7 perfis distintos de consumo (7 classes).

[2] O código fonte está disponível em: `https://github.com/Lucas-Muniz/MNEAT.git`

[3] Página oficial: `https://julialang.org/`

[4] Disponível em: `https://www.kaggle.com/prathamtripathi/drug-classification`

[5] Disponível no arquivo "`income.csv`", no endereço: `https://www.kaggle.com/ahmettezcantekin/beginner-datasets`

[6] Disponível em: `https://archive.ics.uci.edu/ml/datasets/wine+quality`

[7] Disponível em: `https://archive.ics.uci.edu/ml/datasets/Drug+consumption+%28quantified%29`

5.2 MLP e *Backpropagation*

Neste projeto, foram utilizadas as implementações de MLP e do algoritmo *backpropagation* do pacote Flux[8], que fornece implementações práticas para o uso e treinamento de MLPs. As redes MLP foram treinadas utilizando o algoritmo de gradiente descendente com taxa de aprendizagem igual a $\eta = 0{,}2$, sendo que os pesos foram ajustados utilizando o algoritmo *backpropagation*. Dado que as redes serão utilizadas para resolver problemas de classificação, escolheu-se a Entropia Cruzada como função de custo, já que tal função apresenta bom desempenho para problemas de classificação [3]. Para facilitar e aumentar a qualidade do treinamento, os conjuntos de dados passaram por duas etapas de pré-processamento: balanceamento de classes e normalização. Como alguns conjuntos de dados possuem uma quantidade desbalanceada de amostras para cada classe, foi necessário balancear a quantidade de amostras de cada classe no conjunto de treinamento a partir da adição de cópias das amostras que pertenciam às classes desbalanceadas. O objetivo do balanceamento de classes foi permitir com que as redes conseguissem aprender os padrões de cada classe de forma consistente e uniforme (isto é, com igual probabilidade), garantindo, assim, um treinamento melhor, sem privilegiar qualquer classe. Além disso, as amostras também foram normalizadas utilizando a função de normalização do pacote Flux, que transforma os dados a fim de que sua distribuição possua média igual a 0 e desvio padrão igual a 1. Ambos os processos de pré-processamento anteriormente citados foram utilizados tanto no treinamento por *backpropagation* quanto no treinamento utilizando o NEAT implementado. O NEAT modificado que foi implementado neste trabalho é descrito na próxima subseção.

5.3 MNEAT

Neste projeto, foi implementada uma versão modificada do NEAT original [15], sendo esta versão chamada de *Modified NeuroEvolution of Augmenting Topologies* (MNEAT). A implementação original do NEAT permite uma grande liberdade de possíveis topologias, porém, como neste projeto o foco é a busca por uma MLP, que possui uma topologia mais restrita, uma codificação mais compacta foi criada. Dado que em uma MLP os neurônios de uma camada estão totalmente conectados com os neurônios da próxima camada, é possível criar

[8]Documentação do pacote: `https://fluxml.ai/Flux.jl`

uma codificação onde todos os pesos dos neurônios conectados a um mesmo neurônio A são armazenados no gene de neurônio de A. Assim, as conexões da rede não precisam de um gene de conexão, tornando a codificação genética mais compacta. A codificação genética proposta utiliza genes de camada e genes de neurônio. Os genes de camada especificam a características de uma camada da MLP e os genes de neurônio especificam as características de um neurônio da MLP. O gene de camada é composto por um número de identificação, o tipo da camada (intermediária ou saída), a função de ativação dos neurônios da camada e um vetor de genes de neurônio. O gene de neurônio é composto por um número de identificação, o número de inovação, a especificação do tipo de neurônio (intermediário ou saída), a localização do neurônio na rede (posição da camada e do neurônio na camada), o número de pesos conectados ao neurônio, um vetor de pesos, o peso de *bias* e o bit de habilitação. Neste projeto, todos os neurônios das camadas intermediárias possuem a tangente hiperbólica (tanh) como função de ativação, enquanto que os neurônios de saída possuem a função softmax, já que as redes irão resolver problemas de classificação. Além disso, os genes de neurônio apenas codificam neurônios que possuem outros neurônios conectados a eles, logo, os neurônios da camada de entrada não precisam ser representados no genoma do indivíduo. Dessa forma, o genoma de um indivíduo é formado por uma lista de genes de camada, sendo que cada um desses genes possui uma lista de genes de neurônio. A Figura 7 ilustra a codificação genética proposta e como ela é interpretada como uma MLP (fenótipo). Note que um neurônio só será expresso se o seu bit de habilitação estiver habilitado (valor verdadeiro), sendo uma variação em relação ao NEAT original, já que um neurônio é desabilitado e não apenas uma conexão. Caso o bit de habilitação esteja desabilitado, o neurônio e os pesos de conexão associados a ele não serão expressos, como mostrado na Figura 7.

Neste projeto, o MNEAT foi implementado contendo três tipos de mutação: mutação de pesos, mutação estrutural e mutação de bit de habilitação. A mutação de pesos, assim como no NEAT original, adiciona uma perturbação em alguns pesos da rede aleatoriamente. Neste projeto, quando um gene de neurônio sofre uma mutação de pesos, todos os pesos do vetor de pesos conectados ao neurônio sofrerão uma perturbação. A perturbação executada nos pesos conectados ao k-ésimo neurônio da camada ℓ, para todo $1 \leq j \leq n_\ell$, será dada por

$$w_{jk}^{(\ell)} = w_{jk}^{(\ell)} + \xi\kappa_p, \tag{6}$$

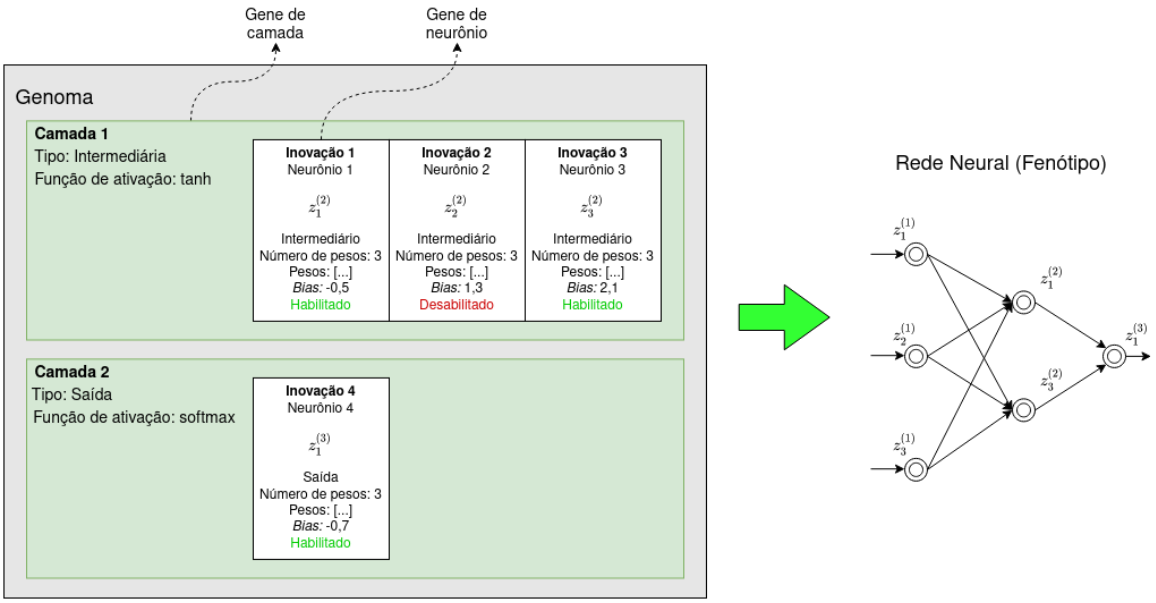


Figura 7: Codificação genética implementada no NEAT neste projeto.

onde ξ é um número aleatório gerado a partir de uma distribuição normal com média igual 0 e desvio padrão igual a 1, e κ_p é um parâmetro que define o valor da maior mutação possível a ser aplicada no peso. Já a perturbação adicionada ao termo *bias* do k-ésimo neurônio da camada ℓ é dada por

$$w_{0k}^{(\ell)} = w_{0k}^{(\ell)} + \xi\kappa_b, \tag{7}$$

onde κ_b é um parâmetro que define o valor da maior mutação possível a ser aplicada no termo *bias*. A mutação estrutural pode ser de dois tipos: de *adição de neurônio* ou de *adição de camada*. A mutação de adição de neurônio adiciona um gene de neurônio ao genoma em uma camada aleatória e o vetor de pesos do novo gene é gerado aleatoriamente. Para manter a conectividade da MLP, todos neurônios da camada seguinte recebem um peso (gerado aleatoriamente) associado ao novo neurônio, em seus respectivos vetores de pesos. Observe que a camada de saída mantém um número constante de neurônios, por isso não sofre nenhuma mutação de adição de neurônios. A mutação de adição de camada faz a adição de uma camada de neurônios entre duas camadas intermediárias consecutivas selecionadas aleatoriamente ou entre a última intermediária e a de saída, criando um novo gene de camada. A quantidade de neurônios da nova camada será igual ao piso do valor médio do número de neurônios das duas camadas consecutivas selecionadas. O objetivo é que a nova camada possua um número de neurônios semelhante ao das camadas vizinhas. Já a mutação de

bit de habilitação selecionará um único neurônio aleatório de uma camada intermediária e inverterá o seu bit de habilitação, caso a camada possua mais de um neurônio. Ao aplicar o operador de mutação em um indivíduo, a probabilidade de ocorrência de uma mutação estrutural é dada pelo parâmetro p_{me}. Caso uma mutação estrutural ocorra, a probabilidade de uma mutação de adição de neurônio é dada por p_{mn} e a probabilidade de uma mutação de adição de camada é dada por p_{mc}. Caso não ocorra uma mutação estrutural, a probabilidade de uma mutação de bit de habilitação é dada por p_{mh}. E caso uma mutação de bit de habilitação não ocorra, será aplicada uma mutação de pesos, que selecionará aleatoriamente um neurônio de cada camada e adicionará uma perturbação no vetor de pesos do neurônio com uma probabilidade p_{mpp} e uma perturbação no termo *bias* com uma probabilidade p_{mpb}. O Algoritmo 2 ilustra o pseudocódigo do operador de mutação de pesos.

Algoritmo 2: Pseudocódigo do operador de mutação de pesos de um indivíduo i com probabilidade p_{mpp} de alterar os pesos do vetor de pesos e probabilidade p_{mpb} de alterar o *bias*.

```
Entrada: i, p_mpp, p_mpb
camadas = CONTAR_CAMADAS(i)
para c = 0 até camadas − 1 faça
    n = SELECIONAR_NEURONIO_ALEATORIO(i, c)
    tam = COMPRIMENTO(n.pesos)
    para p = 0 até tam − 1 faça
        se GERAR_NUMERO_ALEATORIO() ≤ p_mpp então
            n.pesos[p] = ADICIONAR_PERTURBAÇÃO(n.pesos[p], p_mpp)
    se GERAR_NUMERO_ALEATORIO() ≤ p_mpb então
        n.bias = ADICIONAR_PERTURBAÇÃO(n.bias, p_mpb)
    i = ATUALIZAR_INDIVIDUO(i, c, n)
devolve i
```

O cruzamento implementado no projeto segue a mesma ideia do NEAT original. Dados dois indivíduos, os genes de neurônio correspondentes (mesmo número de inovação) serão alinhados e, para cada número de inovação, um gene correspondente será escolhido aleatoriamente para formar o novo individuo. O filho também irá herdar os genes disjuntos e excessivos do pai que possui o maior desempenho, e caso os pais possuam o mesmo desempenho, os genes disjuntos e excessivos de ambos os pais serão herdados. Note que, caso um gene de neurônio seja incluído em uma rede em que o neurônio possui menos pesos de conexão, então os pesos excessivos serão eliminados do vetor de pesos do neurônio. Caso ele

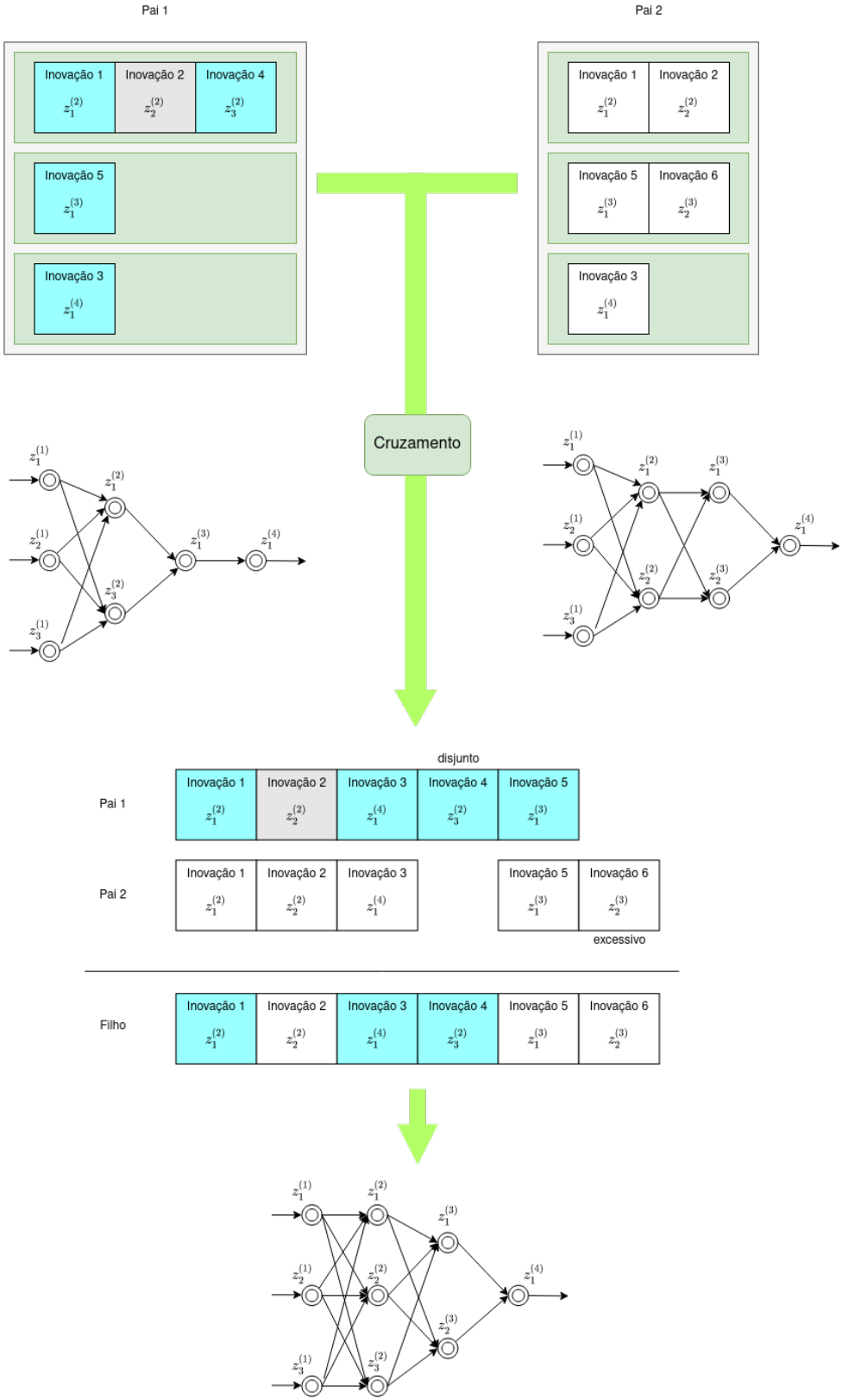


Figura 8: Cruzamento de dois indivíduos no MNEAT. Os genes cinzas estão desabilitados, os genes azuis pertencem ao pai 1 e os brancos ao pai 2. Considerando que neste caso o pai 1 e o pai 2 possuem o mesmo desempenho, os genes disjuntos e excessivos são passados para o filho, enquanto que os genes correspondentes são escolhidos aleatoriamente.

seja adicionado a uma rede com mais pesos de conexão, então novos pesos aleatórios serão adicionados ao vetor de pesos. A Figura 8 ilustra o processo de cruzamento utilizando a codificação genética do MNEAT.

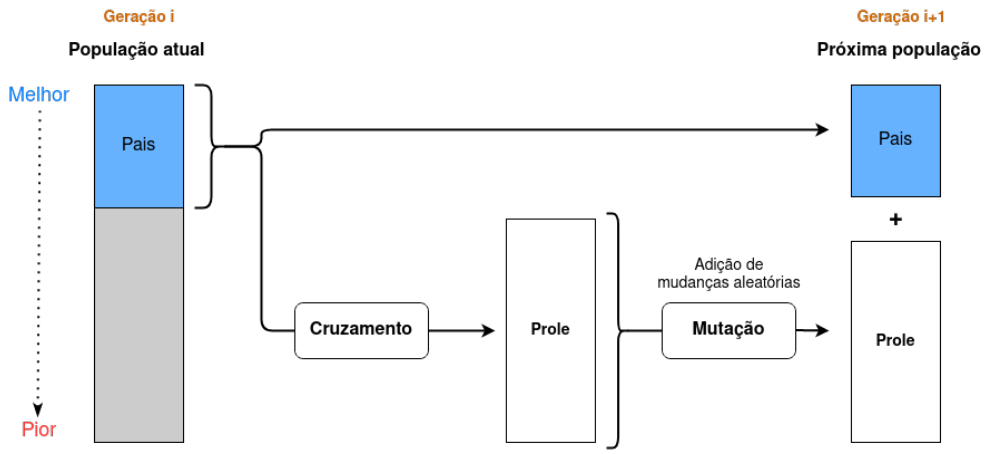


Figura 9: Processo de aplicação dos operadores genéticos do GA em uma espécie do MNEAT.

O processo de especiação implementado também segue a ideia do NEAT original, sendo que a Figura 9 ilustra o processo de evolução que ocorre dentro de uma espécie, inspirado na evolução feita pelo GA. Inicialmente, uma população com n_p indivíduos é criada, sendo que todos os indivíduos possuem a mesma topologia inicial e, por conseguinte, todos pertencem à mesma espécie. A topologia inicial adotada é uma rede MLP com uma camada intermediária composta por apenas um neurônio, sendo que os pesos de conexão são gerados aleatoriamente. Observe que não há uma implementação concreta do operador de seleção, já que o cruzamento gera exatamente a quantidade de filhos necessária para a formação da população da espécie na próxima geração.

A cada geração, ranqueamos os indivíduos do melhor ao pior, utilizando a função *fitness*. Sendo $t_{et}(i)$ a taxa de erro no conjunto de treinamento de um indivíduo i, a função *fitness* f foi definida como

$$f(i) = \frac{2}{t_{et}(i) + 1} - 1\,, \tag{8}$$

sendo que, ao longo das gerações, $f(\cdot)$ será maximizada, sendo que $f(\cdot)$ terá seu maior valor quando $t_{et} = 0$, enquanto que terá seu menor valor quando $t_{et} = 0$. Note que f é normalizada, ou seja, $0 \leq f(i) \leq 1$. A Figura 10 ilustra o comportamento de $f(\cdot)$ em função de t_{et}. Para gerar a prole da próxima geração, uma porcentagem p_{pe} dos melhores indivíduos da espécie é selecionada como o grupo de pais. Esse grupo passará pelo processo de cruzamento, que

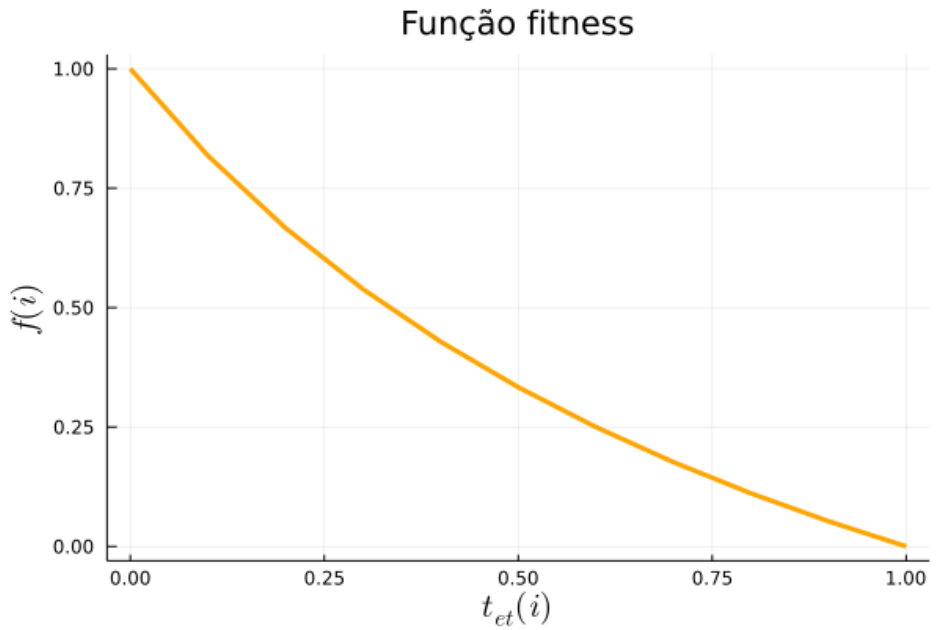


Figura 10: Comportamento da função de fitness de um indivíduo i em relação a $t_{et}(i)$.

selecionará dois pais aleatórios para gerar uma nova rede, formando assim os indivíduos filhos (prole). Todos as redes da prole passarão pelo processo de mutação, que alterará os pesos da rede ou adicionará uma nova estrutura (novo neurônio ou nova camada). Após o processo de mutação, a população da espécie na próxima geração será formada pela junção do grupo de pais e do grupo de proles. A cada nova geração, um indivíduo aleatório da espécie da geração anterior é escolhido como o representante da espécie. Assim, antes de formar a população da próxima geração, todos os indivíduos têm a sua distância em relação ao representante de todas as outras espécies medida. O indivíduo será adicionado à primeira espécie que possuir uma distância compatível, ou seja, abaixo do limiar δ_t. Caso algum indivíduo passe do limiar δ_t de separação de todos os representantes, uma nova espécie é criada, sendo esse indivíduo o seu representante.

Para definir se um determinado indivíduo pertence a uma espécie, dados dois indivíduos i e j, a distância entre eles foi definida como

$$\delta(i,j) = c_1 E_{ij} + c_2 D_{ij} + c_3 \overline{W}_{ij}\,, \tag{9}$$

sendo E_{ij} o número de genes excessivos entre i e j, D_{ij} o número de genes disjuntos entre i e j e $\overline{W}_{ij}$ a média da diferença dos pesos dos genes correspondentes entre i e j (incluindo os genes desabilitados e considerando o menor vetor de pesos dos genes de neurônio avaliados).

Os coeficientes foram definidos como $c_1 = 1$, $c_2 = 1$ e $c_3 = 0,4$, enquanto que o limiar de separação de espécie utilizado foi $\delta_t = 3$. Escolheu-se usar esta função de distância pois ela gerou os melhores resultados durante os testes de execução.

Durante os testes iniciais, a simulação do MNEAT utilizando um número ilimitado de espécies por geração gerou resultados insatisfatórios, porém, ao limitar o número de espécies total por geração, os resultados obtidos melhoraram consideravelmente. Dessa forma, neste projeto, o MNEAT foi implementado com restrição na quantidade de espécies por geração. Uma possível explicação da vantagem da limitação de espécies é que uma grande diversidade de topologias permite explorar melhor o espaço de busca, mas diminui a velocidade de otimização dos pesos da topologia de uma espécie, já que a otimização dos pesos depende da quantidade de indivíduos que possuem uma topologia similar dentro de uma espécie. Portanto, para que os pesos de uma topologia sejam otimizados eficientemente, é necessário que uma espécie tenha uma quantidade de indivíduos razoável em relação à quantidade total de indivíduos na população. Dessa forma, a limitação de espécies diminui a diversidade de topologias, mas permite uma melhor otimização dos pesos da rede, já que as espécies são maiores. Para efetuar a seleção das espécies, houve a criação de uma política de exclusão, que excluirá a espécie mais velha que não possui o melhor indivíduo sempre que uma nova espécie for criada. Além dessa política de exclusão, também foi implementada uma condição de exclusão de espécies caso ocorra estagnação do processo de busca. Se o melhor indivíduo encontrado não mudar após 100 gerações, então as duas espécies com o melhor desempenho (melhor *fitness* médio) serão mantidas e as outras espécies serão eliminadas. Além disso, caso uma espécie, que não tenha o melhor indivíduo da geração, não melhore seu *fitness* médio após 30 gerações, a espécie será removida da população. Após a remoção de uma espécie, a população será completada produzindo novos filhos das espécies remanescentes na população.

Para definir o tamanho da prole gerada por uma espécie na próxima geração, será utilizado o *fitness* médio dos indivíduos de uma espécie, sendo que no MNEAT não utilizamos a função *fitness sharing* (sh). Seja $\overline{f}_i$ o *fitness* médio dos indivíduos de uma espécie i, n_{te} o número total de espécies de uma geração e n_p o número total de indivíduos na população. A quantidade q_i de indivíduos da espécie i na próxima geração será

$$q_i = \left\lfloor \alpha_i \frac{\overline{f}_i n_p}{\sum_{j=1}^{n_{te}} \overline{f}_j} \right\rfloor, \tag{10}$$

onde α_i é um parâmetro que bonificará ou penalizará uma espécie de acordo com a idade da espécie (número de gerações em que a espécie está presente na população). O parâmetro α_i será responsável por bonificar espécies recém criadas, enquanto que irá penalizar espécies mais velhas. Dessa forma, novas espécies conseguirão otimizar os seus indivíduos mais rapidamente, o que melhora a competição entre as espécies, já que espécies mais velhas já estão mais "otimizadas". O valor do parâmetro α_i é definido da seguinte forma:

$$\alpha_i = \begin{cases} 1{,}2 & \text{se } I_i < 10 \\ 1 & \text{se } 10 \leq I_i \leq 30 \\ 0{,}8 & \text{se } I_i > 30 \end{cases}, \tag{11}$$

onde I_i é número de gerações em que a espécie i está presente na população. Note que devido ao parâmetro α_i, pode ocorrer que $\sum_{j=1}^{n_{te}} q_j \neq n_p$. Caso o número de indivíduos gerados seja maior que o número da população, então alguns indivíduos (que não sejam representantes de uma espécie ou o melhor indivíduo) serão descartados aleatoriamente até atingir o número total da população. Caso o número de indivíduos gerados seja menor que o necessário, novos indivíduos de espécies aleatórias serão gerados até atingir o número total da população.

5.4 Medidas de Informação

Neste trabalho, a medida de informação mútua foi utilizada para medir a eficiência das camadas de uma MLP. Interpretando cada camada de uma rede com t camadas como uma variável Z_ℓ, para $2 \leq \ell \leq t$, a informação mútua entre as camadas e a entrada e a saída esperada será calculada.

Para calcular a informação mútua de uma camada, a saída da função de ativação dos neurônios (tangente hiperbólica) foi discretizada particionando-a em $n_{int} = 10$ intervalos igualmente espaçados entre -1 e 1. A discretização de uma camada ℓ foi feita a partir da utilização dos valores discretizados dos neurônios, sendo que todas as possíveis combinações de valores foram discretizadas por um histograma que contém uma quantidades de barras (igualmente espaçadas) igual a $n_{int} \cdot n_\ell$. Os valores discretizados da camada ℓ serão interpretados como uma variável aleatória Z_ℓ e, assim, poderemos calcular diretamente a distribuição conjunta da camada e dos padrões de entrada pertencentes a X, o que permite calcular

$p(Z_\ell, X)$. Considerando que a informação sobre os valores a serem preditos é modelada pela cadeia de Markov $Y \rightarrow X \rightarrow Z_\ell$, então, para cada camada da rede, a probabilidade conjunta de Z_ℓ e Y é dada por $p(Z_\ell, Y) = \sum_x p(x, Y)p(Z_\ell|x)$, para $x \in X$ [14]. Utilizando essas probabilidades conjuntas, é possível calcular $I(X, Z_\ell)$ e $I(Z_\ell, Y)$ de cada camada, que são os termos da relação de compressão e da qualidade de predição da camada, respectivamente. A fim de normalizar o valor de $I(Z_\ell, Y)$, o valor calculado para o plano de informação foi $I(Z_\ell, Y)/I(X, Y)$.

6 Simulações e Resultados

Nesta seção iremos descrever as simulações do MNEAT e do *backpropagation* executadas nos quatro conjuntos de dados selecionados. Em todas as simulações utilizou-se os mesmos valores de parâmetros para o MNEAT, listados na Tabela 1, sendo que o MNEAT foi inicializado com uma população de 150 indivíduos e executadas apenas umas vez para cada conjunto de dados. Com o propósito de estudar mais profundamente o processo de treinamento executado pelo MNEAT, alguns gráficos contendo informações sobre a evolução do MNEAT foram gerados. Para a análise da evolução das soluções, imagens sobre a evolução do *fitness* do melhor indivíduo e do *fitness* médio da população foram geradas. Além disso, para complementar a análise da evolução das redes, imagens sobre a evolução da complexidade topológica e da acurácia (no conjunto de treinamento e de teste) do melhor indivíduo também foram geradas. Neste trabalho, a complexidade topológica de uma rede é dada pelo número de pesos de conexões (sem contar os pesos associados ao *bias*).

Para a análise da capacidade de generalização da rede obtida, utilizou-se imagens da distribuição de saída da melhor rede encontrada pelo MNEAT, nos conjuntos de treinamento e no teste. Já a eficiência das redes treinadas pelo *backpropagation* e pelo MNEAT foram comparadas a partir de imagens do plano de informação das redes treinadas por ambos os métodos, sendo que o plano de informação foi medido utilizando o conjunto de treinamento. O plano de informação das redes treinadas pelo *backpropagation* foi obtido a partir do valor médio da posição das camadas de 10 redes treinadas por 5000 épocas, sendo a posição no plano plotada a cada 250 épocas, enquanto que, para a geração do plano de informação das redes treinadas pelo MNEAT, plotou-se a posição da melhor rede de cada geração. A descrição da arquitetura das redes treinadas pelo *backpropagation* e pelo MNEAT se encontram nas próximas seções. Os modelos testados no *backpropagation* possuíam apenas uma camada intermediária, sendo que esta camada possuía um número de neurônio igual a 1, 1,5 ou 2 vezes o número de neurônios da camada de entrada do problema. Por fim, a taxa de erro das redes treinadas por ambos os algoritmos de treinamento é mostrada na Tabela 2. Para o *backpropagation*, foi calculada a taxa de erro média de 10 redes treinadas por 5000 épocas, enquanto que a taxa de erro do MNEAT foi definida pela taxa de erro do melhor indivíduo obtido após 5000 gerações.

Tabela 1: Parâmetros do MNEAT.

Parâmetro	Valor
n_p	150
n_{te}	3
p_{me}	0,15
p_{mn}	0,7
p_{mc}	0,2
p_{mh}	0,15

Parâmetro	Valor
p_{mpp}	0,9
p_{mpb}	0,5
p_{pe}	0,2
κ_p	0,5
κ_b	0,5

Tabela 2: Taxa de erro das redes treinadas nos quatro conjunto de dados. A coluna "BP" representa a taxa de erro das redes treinadas pelo backpropagation, no conjunto de treinamento e de teste.

	Conjunto 1		Conjunto 2		Conjunto 3		Conjunto 4	
	BP	MNEAT	BP	MNEAT	BP	MNEAT	BP	MNEAT
Treinamento	0	0,0282	0,1368	0,2314	0,6924	0,6013	0,494	0,6271
Teste	0	0,15	0,2125	0,27	0,8071	0,8231	0,8093	0,9178

6.1 Conjunto de Dados 1

No conjunto de dados 1, durante os testes com o treinamento por *backpropagation*, a menor rede com melhor desempenho encontrada experimentalmente foi uma rede de 3 camadas, em que cada camada possui 5 neurônios. No treinamento utilizando o MNEAT, a melhor rede encontrada também possui 3 camadas, sendo que as camadas de entrada e saída possuem 5 neurônios cada e a intermediária possui 6 neurônios. Na Tabela 2, podemos observar que o desempenho da rede treinada pelo MNEAT é inferior ao da rede treinada pelo *backpropagation*. Além disso, a rede treinada pelo MNEAT é ligeiramente maior (um neurônio a mais).

As Figuras 11, 12a e 12b ilustram a evolução do *fitness*, da complexidade topológica e da acurácia da população do MNEAT, respectivamente, ao longo do treinamento no conjunto de dados 1. Note que, pela Figura 11, inicialmente, o *fitness* do melhor indivíduo cresce rapidamente, se estabiliza por algumas gerações, aumenta novamente e se estabiliza mais uma vez. É possível observar nas Figuras 12a e 12b que esses aumentos coincidem com o aumento da complexidade topológica e da acurácia do melhor indivíduo, logo, é possível notar que o MNEAT escolhe uma rede maior quando esse incremento melhora o desempenho do indivíduo. Quando um incremento topológico não gera redes melhores, o MNEAT passa a otimizar os pesos da rede. Pela Figura 12b, é possível notar que mesmo apenas utilizando a taxa de erro sobre o conjunto de treinamento, o MNEAT consegue buscar redes com boa

capacidade de generalização, já que a acurácia no conjunto de teste tende a aumentar junto com a do conjunto de treinamento.

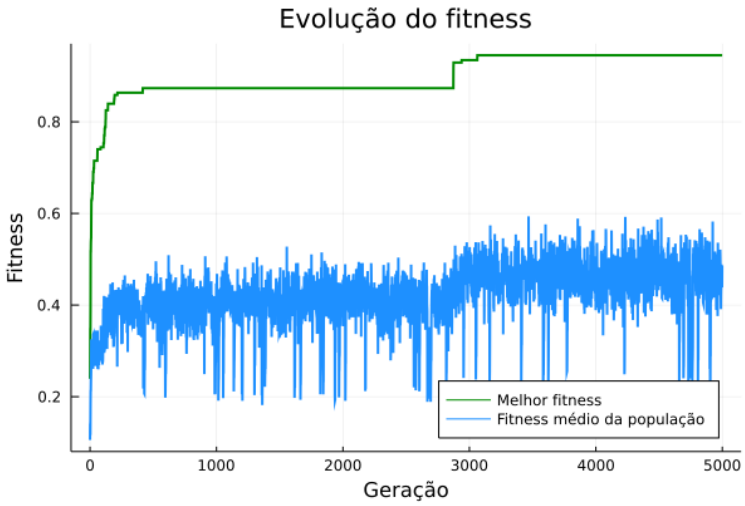
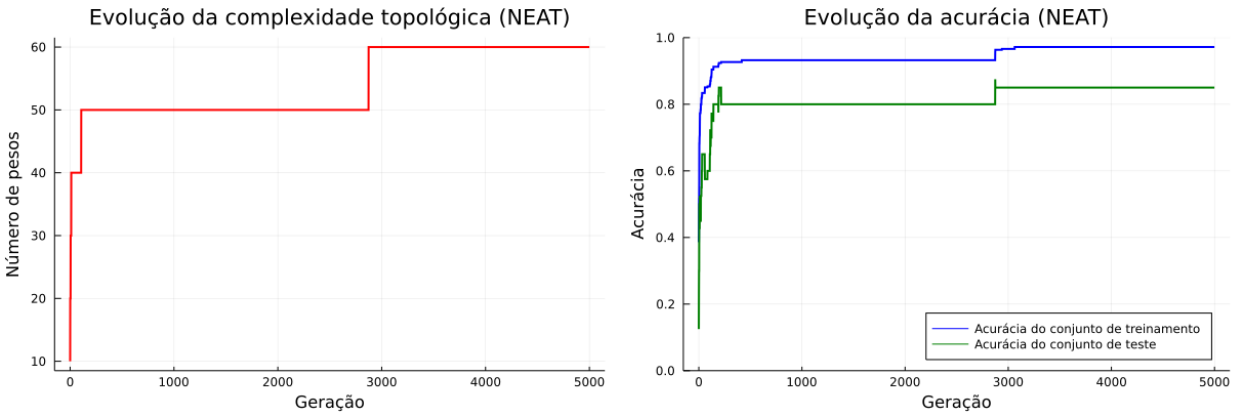


Figura 11: Evolução do valor da função fitness no conjunto de dados 1.

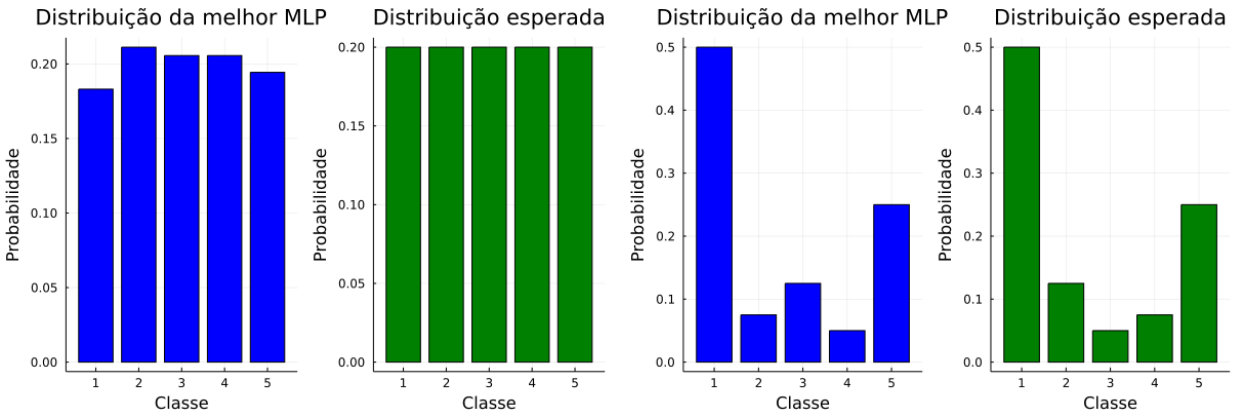


(a) Complexidade topológica. *(b) Acurácia.*

Figura 12: Evoluções no conjunto de dados 1.

A Figura 13 mostra a distribuição de saída da melhor rede encontrada pelo MNEAT, ilustrando a boa capacidade de generalização da rede MLP encontrada. Na Figura 13a é possível notar que a distribuição da rede se assemelha à distribuição esperada no conjunto de treinamento. Já na Figura 13b é possível observar que a distribuição da classificação da rede também se assemelha à distribuição esperada no conjunto de teste, sendo possível avaliar que parte dos erros de classificação da rede ocorre para amostras da classe 3, que é a classe que possui a menor quantidade de amostras no conjunto de dados 1.

As Figuras 14a e 14b mostram o plano de informação da rede treinada pelo *backpropaga-*



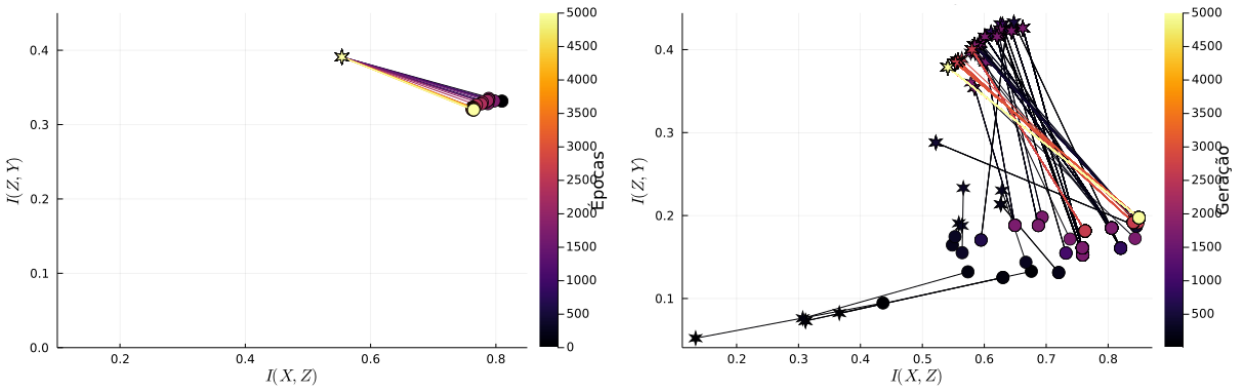
(a) No conjunto de treinamento. *(b) No conjunto de teste.*

Figura 13: Distribuição de saída da melhor rede encontrada pelo MNEAT no conjunto de dados 1.

tion e pelo MNEAT, respectivamente. Pela Figura 14a, podemos observar que a informação sobre os valores desejados aumenta rapidamente já nas primeiras épocas, mas a informação sobre a entrada diminui lentamente no decorrer do treinamento. Já pela Figura 14b, a informação sobre os valores desejados e sobre a entrada aumenta lentamente. Utilizando as figuras do plano de informação, podemos comparar a eficiência das redes treinadas pelo *backpropagation* e MNEAT, respectivamente. A partir das figuras, vemos que ambos os métodos de treinamento geraram redes com valores de $I(Z, Y)$ na camada de saída semelhantes, porém, o treinamento pelo *backpropagation* atingiu esse valor mais rapidamente que o MNEAT, que gradualmente atingiu esse valor. No entanto, os valores de $I(X, Z)$ da camada intermediária da rede treinada pelo *backpropagation* foi um pouco menor que o do MNEAT, mostrando que o *backpropagation* gerou uma rede com capacidade de compressão melhor. Mesmo com essas diferenças, é possível notar que o plano de informação das redes treinadas por ambos os métodos possuem valores semelhantes.

6.2 Conjunto de Dados 2

No conjunto de dados 2, treinando a rede pelo algoritmo *backpropagation*, a menor rede encontrada experimentalmente com um bom desempenho foi uma rede de 3 camadas, em que a camada de entrada possui 13 neurônios, a camada intermediária possui 18 neurônios e a camada de saída possui 2 neurônios. Ao executar o treinamento utilizando o MNEAT, obtivemos uma rede de 3 camadas, sendo que a camada de entrada possui 13 neurônios,



(a) Do treinamento pelo backpropagation. *(b) Do treinamento pelo MNEAT.*

Figura 14: Evolução do plano de informação no conjunto de dados 1.

a intermediária possui 9 neurônios e a de saída possui 2 neurônios. Na Tabela 2 podemos observar que o desempenho da rede treinada pelo MNEAT é inferior ao da rede treinada pelo *backpropagation*, porém, essa rede é menor que a rede encontrada experimentalmente no treinamento pelo *backpropagation*.

As Figuras 15, 16a e 16b ilustram a evolução do *fitness*, da complexidade topológica e da acurácia da população do MNEAT, respectivamente, ao longo do treinamento no conjunto de dados 2. Note que, pela Figura 11, o *fitness* do melhor indivíduo cresce rapidamente e depois se estabiliza pelo resto das gerações. É possível observar nas Figuras 12a e 12b que esse aumento coincide com o aumento da complexidade topológica e a acurácia do melhor indivíduo, logo, é possível notar que o MNEAT escolhe uma rede com complexidade adequada e depois passa a otimizar os pesos dessa rede. Pela Figura 12b, é possível notar que o MNEAT consegue buscar redes com boa capacidade de generalização, porém, durante a fase de otimização dos pesos, a acurácia sobre o conjunto de teste diminui e depois se estabiliza.

A capacidade de generalização da rede MLP encontrada é ilustrada na Figura 17, que mostra a distribuição de saída da melhor rede encontrada pelo MNEAT. Na Figura 17a é possível notar que a distribuição da rede se assemelha à distribuição esperada no conjunto de treinamento, mas ainda comete erros ao classificar amostras da classe 0. Já na Figura 17b é possível observar que a distribuição da classificação da rede se assemelha à distribuição esperada no conjunto de teste, sendo possível observar que os erros de classificação da rede ocorrem para amostras da classe 0, que gerou mais erros no conjunto de treinamento.

As Figuras 18a e 18b mostram o plano de informação da rede treinada pelo *backpropaga-*

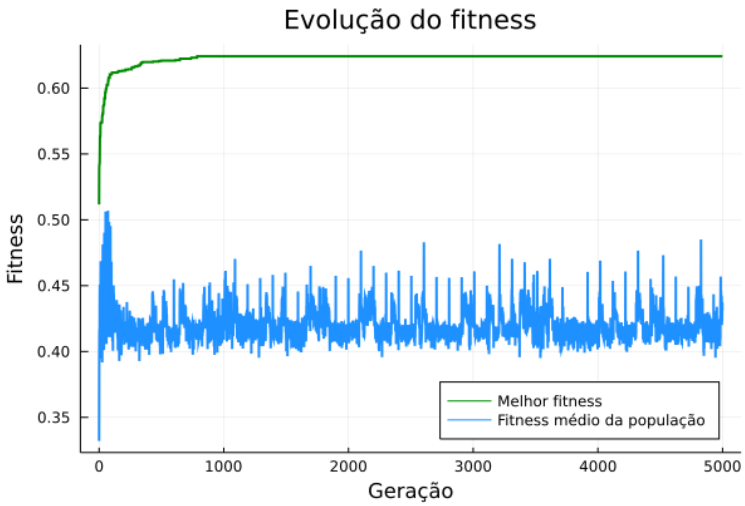
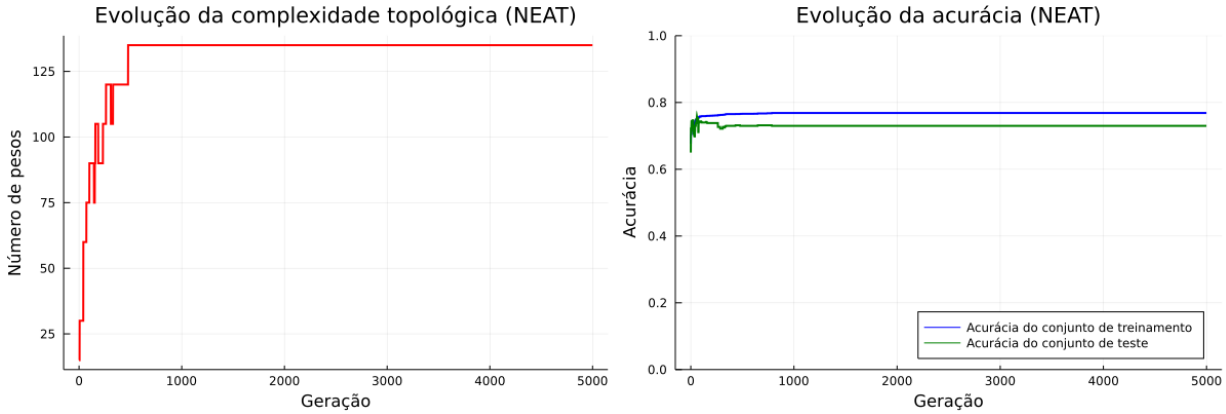


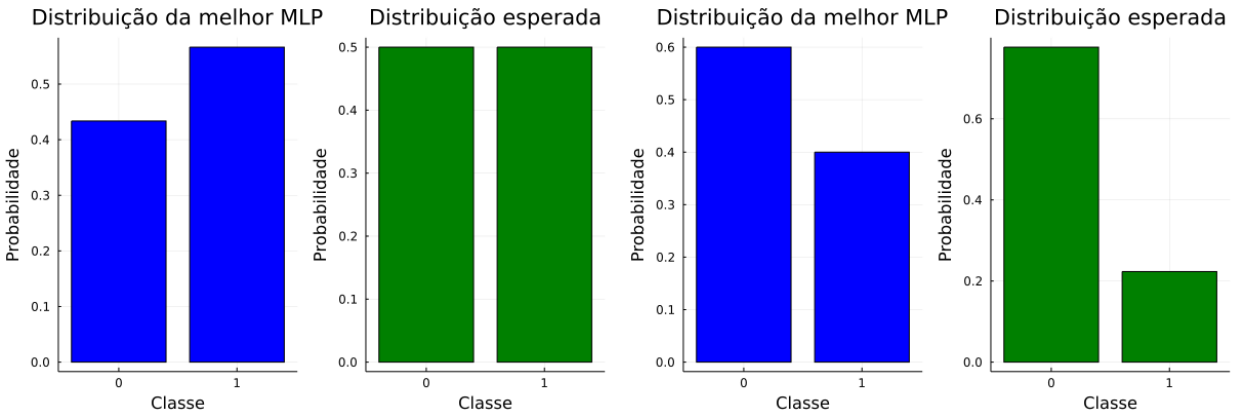
Figura 15: Evolução do valor da função fitness no conjunto de dados 2.



(a) Complexidade topológica. *(b) Acurácia.*

Figura 16: Evoluções no conjunto de dados 2.

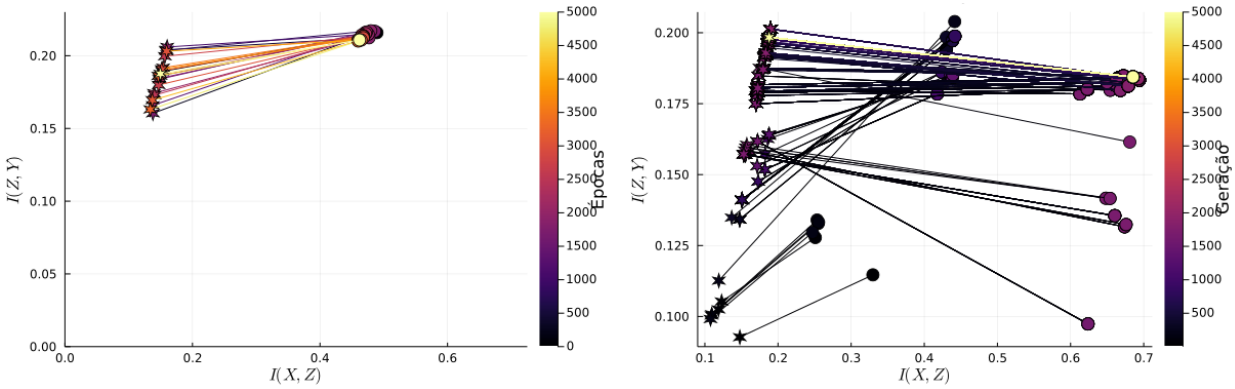
tion e pelo MNEAT, respectivamente. Pela Figura 18a, podemos observar que a informação sobre os valores desejados aumenta rapidamente, sendo que a informação sobre a entrada diminui ligeiramente na camada intermediária no decorrer das épocas. Também é possível notar que a informação sobre os valores desejados na camada de saída acaba diminuindo. Tal comportamento pode ter ocorrido devido ao treino excessivo da rede. Observe que, neste projeto, não foi feito nenhum teste em dados de validação para evitar o sobretreino da MLP. Já pela Figura 18b, é possível notar que, diferentemente do *backpropagation*, a informação sobre os valores desejados e sobre a entrada aumenta lentamente ao longo das gerações. Utilizando as Figuras 18a e 18b, podemos ver que ambos os métodos de treinamento geraram redes com valores de $I(Z,Y)$ semelhantes na camada intermediária e de saída. Note que o treinamento por *backpropagation* no plano de informação evoluiu mais rapidamente que



(a) No conjunto de treinamento. *(b) No conjunto de teste.*

Figura 17: Distribuição de saída da melhor rede encontrada pelo MNEAT no conjunto de dados 2.

o MNEAT. Também é possível observar que os valores de $I(X, Z)$ da camada intermediária da rede treinada pelo *backpropagation* foi aproximadamente metade do valor da camada intermediária da rede do MNEAT, mostrando que o *backpropagation* gerou uma rede com capacidade de compressão bem superior à rede treinada pelo MNEAT.



(a) Do treinamento pelo backpropagation. *(b) Do treinamento pelo MNEAT.*

Figura 18: Evolução do plano de informação no conjunto de dados 2.

6.3 Conjunto de Dados 3

No conjunto de dados 3, após a testagem experimental de alguns modelos, a melhor rede encontrada no treinamento pelo algoritmo *backpropagation* foi uma rede com 3 camadas, sendo que as camadas de entrada e de saída possuem 11 neurônios cada e a camada intermediária possui 16 neurônios. Ao treinar utilizando o MNEAT, obtivemos uma rede de 3 camadas, sendo que as camadas de entrada e de saída possuem 11 neurônios cada e a intermediária

possui 4 neurônios. Na Tabela 2 podemos observar que o desempenho da rede treinada pelo MNEAT tem uma capacidade de generalização um pouco inferior à capacidade de generalização da rede treinada por retropropagação, mas, assim como no conjunto de dados anterior, a rede encontrada pelo MNEAT é bem menor que a rede treinada por retropropagação. Mesmo assim, a taxa de erro no conjunto de treinamento da rede treinada pelo MNEAT foi melhor que a rede treinada pelo *backpropagation*.

As Figuras 19, 20a e 20b ilustram a evolução do *fitness*, da complexidade topológica e da acurácia da população do MNEAT, respectivamente, ao longo do treinamento no conjunto de dados 3, respectivamente. A Figura 15 mostra que, assim como no conjunto de dados 1, o *fitness* do melhor indivíduo cresce rapidamente e depois se estabiliza, até ter um novo aumento seguido por uma estabilização. É possível notar nas Figuras 20a e 20b que esses aumentos coincidem com os aumentos da complexidade topológica e da acurácia do melhor indivíduo. A Figura 20b mostra que, neste conjunto de dados, o MNEAT não foi capaz de encontrar uma rede com uma boa capacidade de generalização, como nos conjuntos anteriores.

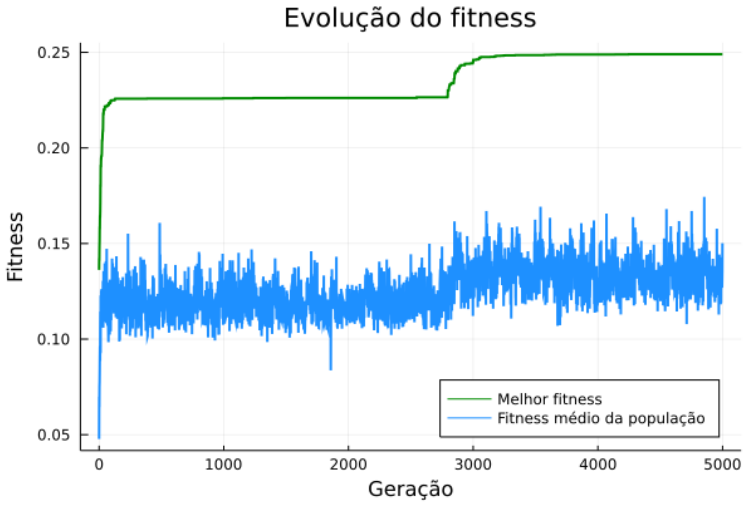
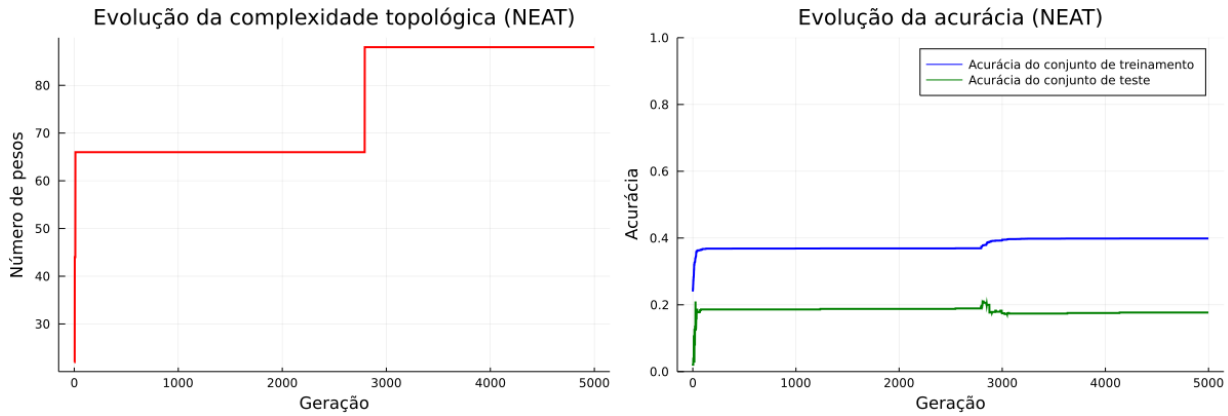


Figura 19: Evolução do valor da função fitness no conjunto de dados 3.

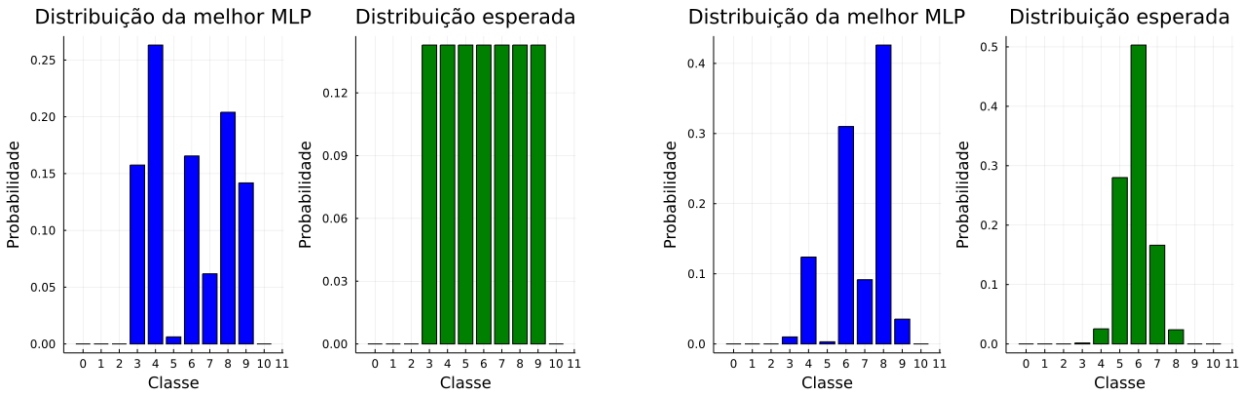
A capacidade de generalização da rede MLP encontrada é ilustrada mais detalhadamente na Figura 21. Na Figura 21a é possível notar que a distribuição da rede não se assemelha à distribuição esperada no conjunto de treinamento, mostrando que a rede tem bastante dificuldade para aprender a distribuição dos dados. Na Figura 21b é possível observar que essa dificuldade da rede permanece no conjunto de teste. Essas informações da distribuição mostram que, provavelmente, a rede está tendo dificuldade para adaptar os seus pesos devido



(a) Complexidade topológica. *(b) Acurácia.*

Figura 20: Evoluções no conjunto de dados 3.

à distribuição das classes no conjunto de dados. Estes resultados também podem indicar que é necessário treinar a rede por mais gerações.

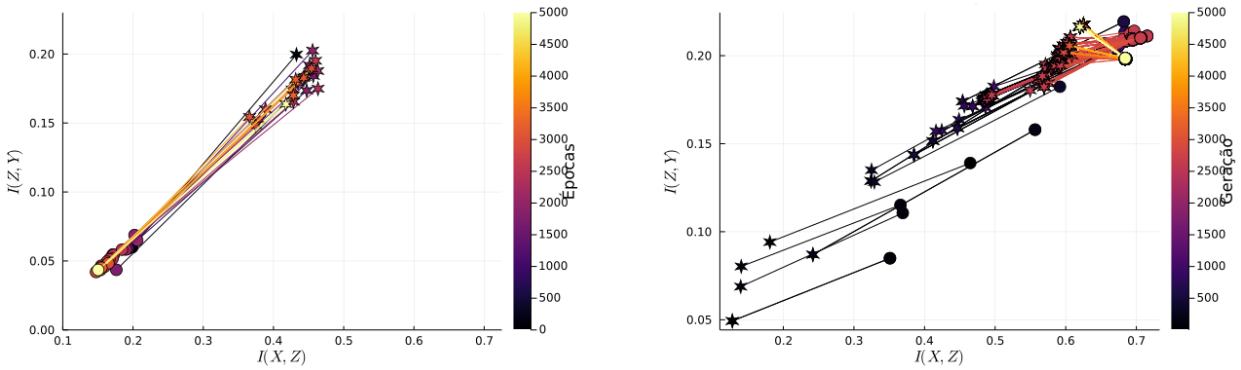


(a) No conjunto de treinamento. *(b) No conjunto de teste.*

Figura 21: Distribuição de saída da melhor rede encontrada pelo MNEAT no conjunto de dados 3.

Já as Figura 22a e 22b mostram o plano de informação da rede treinada pelo *backpropagation* e pelo MNEAT, respectivamente. Pela Figura 22a, podemos observar que a informação sobre os valores desejados e sobre a entrada diminui ao longo das épocas. Tal comportamento mostra que a rede está tendo dificuldade para aumentar a sua qualidade de predição, porém o treinamento consegue gerar camadas com boa capacidade de compressão. Já pela Figura 22b, nota-se que a informação sobre os valores desejados e sobre a entrada aumenta de forma constante e lenta, sendo que as camadas da rede possuem uma grande quantidade de informação dos valores de entrada. Pelas Figuras 22a e 22b, observamos que ambos os métodos de treinamento geraram redes com valores de $I(Z, Y)$ semelhantes na camada de

saída. Porém, os valores $I(X, Z)$ da rede treinada pelo *backpropagation* são bem menores que o valor da rede treinada pelo MNEAT. Logo, a rede treinada pelo *backpropagation* possui uma melhor capacidade de compressão da informação de entrada. Observe que o treinamento por *backpropagation* gerou pouca alteração do plano de informação, enquanto que o treinamento pelo MNEAT claramente aumentou a qualidade de predição da rede gradualmente ao longo das gerações.



(a) Do treinamento pelo backpropagation. *(b) Do treinamento pelo MNEAT.*

Figura 22: Evolução do plano de informação no conjunto de dados 3.

6.4 Conjunto de Dados 4

No conjunto de dados 4, a melhor rede encontrada experimentalmente no treinamento pelo algoritmo *backpropagation* possui 3 camadas, sendo que a camada de entrada possui 12 neurônios, a camada intermediária possui 14 neurônios e a camada de saída possui 7 neurônios. Ao treinar utilizando o MNEAT, obtivemos uma rede de 3 camadas, sendo que a camada de entrada possui 12 neurônios, a intermediária possui 8 neurônios e a de saída possui 7 neurônios. Pela Tabela 2 podemos observar que o desempenho da rede treinada pelo MNEAT, tanto no conjunto de treinamento quanto no conjunto de teste, foi inferior ao desempenho da rede treinada pelo *backpropagation*, mas a rede encontrada pelo MNEAT é menor que a rede treinada pelo *backpropagation*.

As Figuras 23, 24a e 24b ilustram a evolução do *fitness*, da complexidade topológica e da acurácia da população do MNEAT, respectivamente, ao longo do treinamento no conjunto de dados 4, respectivamente. A Figura 23 mostra que o *fitness* do melhor indivíduo cresce rapidamente e depois se estabiliza até o final do treinamento. É possível notar nas Figuras 24a

e 24b que esses aumentos do *fitness* coincidem com os aumentos da complexidade topológica e da acurácia do melhor indivíduo. Pela Figura 24b, vemos que o MNEAT não foi capaz de encontrar uma rede com boa capacidade de generalização, assim como no conjunto 3.

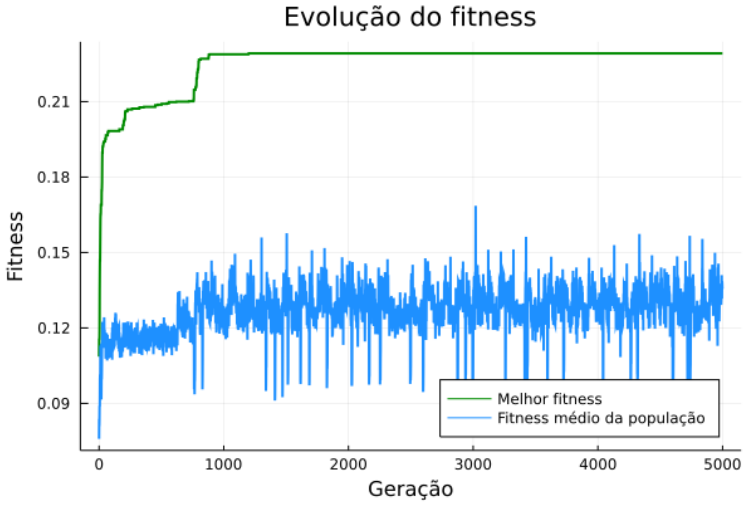
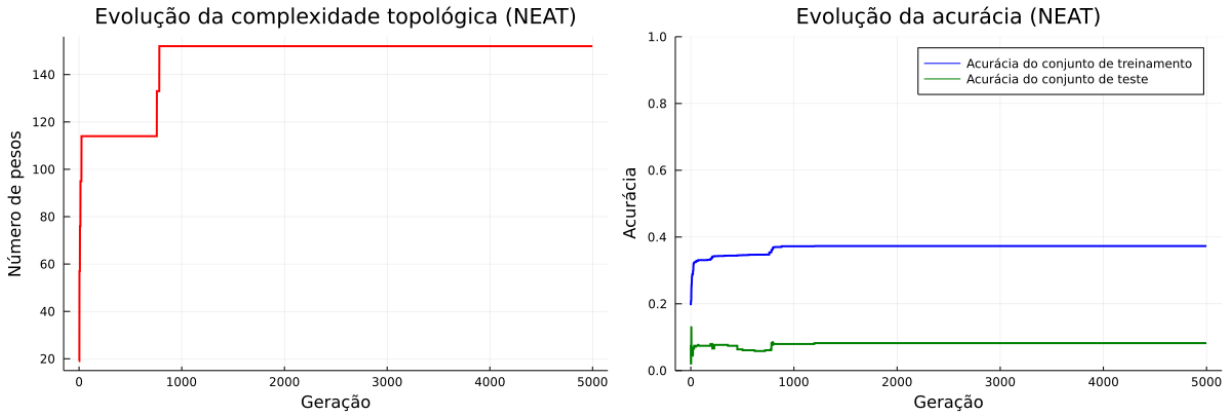


Figura 23: Evolução do valor da função fitness no conjunto de dados 4.

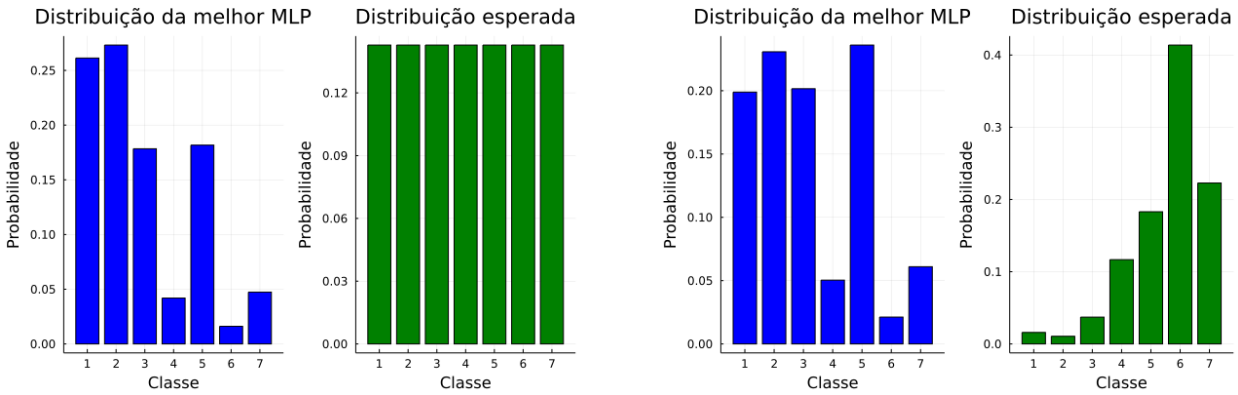


(a) Complexidade topológica. *(b) Acurácia.*

Figura 24: Evoluções no conjunto de dados 4.

A Figura 25 mostra a distribuição de saída da melhor rede encontrada pelo MNEAT. Nas Figuras 25a e 25b, é possível notar que a distribuição da rede não se assemelha à distribuição esperada no conjunto de treinamento nem no conjunto de teste, evidenciando a baixa capacidade de generalização. Estes resultados, assim como no conjunto de dados 3, indicam uma dificuldade do MNEAT em adaptar os pesos da rede, sendo que a rede possivelmente precisa de mais gerações para ser treinada adequadamente.

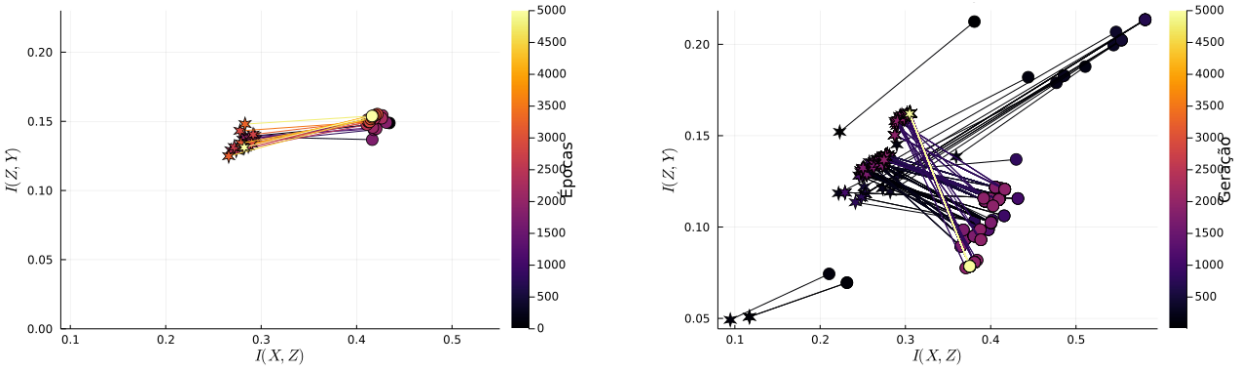
Finalmente, as Figuras 26a e 26b mostram o plano de informação da rede treinada pelo



(a) No conjunto de treinamento. *(b) No conjunto de teste.*

Figura 25: Distribuição de saída da melhor rede encontrada pelo MNEAT no conjunto de dados 4.

backpropagation e pelo MNEAT, respectivamente. Pela Figura 26a, é possível observar que, em um primeiro momento, a informação sobre os valores desejados aumenta, porém, nas épocas finais, este valor diminui na camada de saída. Este comportamento provavelmente indica a ocorrência de sobretreino da rede no final do treinamento. Além disso, é possível notar que houve poucas mudanças na capacidade de compressão das camadas, sendo que a informação sobre os dados de entrada manteve-se relativamente estável ao longo do treinamento. Já na Figura 26b, observa-se que a informação dos valores desejados e da entrada aumentam. Porém, a partir da metade do treinamento, a quantidade de informação da camada intermediária diminui, enquanto a quantidade de informação da camada de saída aumenta. Pelas Figuras 26a e 26b, vemos que ambos os métodos de treinamento conseguiram aumentar $I(Z, Y)$ ao longo do treinamento. Porém, neste conjunto de dados, o MNEAT apresentou melhor capacidade de treinar redes com baixos valores de $I(X, Z)$, diferentemente do *backpropagation*, que manteve a capacidade de compressão das camadas ao passar das épocas. Em relação à qualidade de predição, a rede do *backpropagation* possui uma camada intermediária com boa predição, enquanto que o MNEAT focou na qualidade da camada de saída.



(a) Do treinamento pelo backpropagation. *(b) Do treinamento pelo NEAT.*

Figura 26: Evolução do plano de informação no conjunto de dados 4.

7 Análise de Resultados

Após a realização das simulações utilizando o *backpropagation* e o MNEAT, foi possível identificar algumas características gerais do treinamento feito pelo MNEAT. No geral, o MNEAT foi capaz de achar redes MLP com topologia pequena e com capacidade de generalização similar às das redes treinadas pelo *backpropagation*, porém, as redes treinadas por este último possuíam menor taxa de erro, tanto no conjunto de treinamento quanto no conjunto de teste. Comparando o plano de informação das redes treinadas por ambos os métodos, foi possível observar que os valores da qualidade de predição da camada de saída são semelhantes, entretanto, a capacidade de compressão das redes treinadas pelo *backpropagation* é bem superior. Uma possível explicação para a baixa capacidade de compressão das redes treinadas pelo MNEAT é o fato de que, a cada geração, apenas alguns pesos da rede são alterados, e caso eles melhorem o desempenho do indivíduo eles permanecerão na população, sendo que o *backpropagation* consegue alterar todos os pesos da rede a cada época. Dessa forma, o *backpropagation* é mais eficiente em definir os pesos capazes de gerar representações eficientes dos dados de entrada. Observando a evolução do plano de informação de ambos os métodos, é possível avaliar que as redes treinadas pelo *backpropagation* possuem um movimento mais ordenado e com uma tendência mais clara, enquanto que no MNEAT o movimento é mais desordenado e aleatório. Todavia, o movimento desordenado do MNEAT é coerente com o seu modo de evolução das redes, já que os indivíduos sofrem modificações aleatórias e apenas as mais promissoras permanecem na população.

Também foi possível notar que o *backpropagation* exige menos épocas para obter uma

rede com boa qualidade de predição em relação ao número de gerações do MNEAT, ou seja, o MNEAT precisa de mais iterações. Uma característica interessante é que, durante o processo de otimização de pesos das redes do MNEAT, a acurácia do conjunto de teste sofreu pequenas quedas que logo se estabilizaram, indicando que o MNEAT é possivelmente resistente ao fenômeno de *overfitting*. Outra característica do MNEAT que foi verificada nas simulações é que o valor médio do *fitness* da população se mantém estável e menor que o *fitness* do melhor indivíduo durante o treinamento, indicando que o mecanismo de especiação do MNEAT consegue manter uma população diversa, mesmo com a adoção da limitação de espécies.

Uma limitação do algoritmo MNEAT implementado é a falta de um critério de parada significativo, que consiga interromper a evolução quando o MNEAT não conseguir evoluir mais as soluções. Até o momento, o treinamento irá parar após uma quantidade fixa de gerações ou quando um indivíduo com o maior *fitness* possível for encontrado, sendo que o último caso ocorre quando a taxa de erro no conjunto de treinamento for nula. Também é importante destacar que, neste trabalho, utilizou-se um único conjunto de parâmetros do NEAT para os quatro conjuntos de dados, sendo que MNEAT só foi executado uma vez em cada conjunto de dados. Há a possibilidade de melhorar o desempenho do MNEAT utilizando um conjunto de parâmetros específico para cada problema de classificação, adaptando adequadamente o MNEAT para cada classificação. Dado que o desempenho de predição das redes treinadas pelo MNEAT foi parecido ao das redes treinadas por *backpropagation*, então o MNEAT pode ser uma técnica útil para definir, de forma automática, uma boa topologia para um determinado problema de classificação. Posteriormente, o desempenho da rede poderia ser refinado treinando-a pelo *backpropagation*.

8 Conclusão e Trabalhos Futuros

Este trabalho teve como proposta o estudo de um método de treinamento de MLP baseado no algoritmo de neuroevolução NEAT, no contexto de aprendizado supervisionado. O principal objetivo era investigar as vantagens do NEAT modificado (MNEAT) em relação ao método *backpropagation*, sendo que a eficiência das redes treinadas por ambos os métodos seria avaliada por medidas advindas da Teoria da Informação.

Pelas simulações executadas, observou-se que o MNEAT é capaz de treinar redes com capacidade de generalização similar às das redes treinadas pelo *backpropagation*, além de não sofrer o efeito de *overfitting* e geralmente encontrar redes com topologia menores que as redes encontradas experimentalmente no método *backpropagation*. Contudo, as redes treinadas por *backpropagation* possuem desempenho melhor tanto no conjunto de treinamento quanto no teste. Essa tendência foi confirmada pelo plano de informação, que mostrou que, embora ambos os métodos treinem redes com qualidade de predição semelhantes, o método do *backpropagation* consegue gerar redes com melhores representações dos dados de entrada em cada camada. As principais vantagens do MNEAT, em relação ao método *backpropagation*, são o fato de não sofrer *overfitting*, a característica de não ser afetado pela dissipação do gradiente (pois o treinamento não utiliza informação do gradiente) e a capacidade de gerar topologias pequenas. Entretanto, a sua desvantagem é não possuir um processo de ajuste dos pesos que consiga adaptar todos os pesos da rede de forma coerente e eficiente. Portanto, pelas simulações apresentadas, o MNEAT se destacou em gerar redes com boas topologias, sendo que o *backpropagation* se destacou no treinamento dos pesos da rede.

Dados os resultados obtidos neste projeto, uma possibilidade de trabalho futuro seria a investigação de uma abordagem de treinamento que combine o MNEAT e o algoritmo *backpropagation*. Essa abordagem poderia ser baseada em duas fases de treinamento: na primeira fase o MNEAT seria utilizado para a definição automática de um modelo adequado ao problema, enquanto que na segunda fase o *backpropagation* seria utilizada para o aumento de desempenho da rede obtida pelo MNEAT.

Visto que o critério de minimização de erro não garante a escolha de uma topologia eficiente, outra possibilidade de trabalho futuro seria a incorporação de alguma medida de avaliação que fosse capaz de medir a eficiência da rede. Uma medida promissora é o *Informa-*

tion Bottleneck (IB) [17], que fornece uma maneira estatística de avaliar como a rede obtém informação relevante dos dados de entrada. Além disso, o desempenho do MNEAT poderia ser melhorado pela modificação dos seus operadores genéticos, modificando-os para enviesar a busca para o máximo global do espaço de soluções. Isso poderia ser feito pela utilização de operadores genéticos inspirados nos operadores do BRKGA, que utilizam o cruzamento uniforme parametrizado e a separação entre um conjunto de indivíduos elite e não elite [11] como técnicas para enviesar a busca executada pela metaheurística.

Referências

[1] A. Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[4] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information sciences*, 237:82–117, 2013.

[5] J. Branke. Evolutionary algorithms for neural network design and training. 1995.

[6] H. Braun and J. Weisbrod. Evolving neural feedforward networks. In *Artificial Neural Nets and Genetic Algorithms*, pages 25–32. Springer, 1993.

[7] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[8] R. O. Duda and P. E. Hart. *Pattern classification*. John Wiley & Sons, 2006.

[9] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[10] S. Haykin. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall, 2009. ISBN 9780131471399.

[11] R. Martí, P. M. Pardalos, and M. G. C. Resende, editors. *Handbook of Heuristics*. Springer International Publishing, 2018.

[12] J. C. F. Pujol and R. Poli. Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence*, 8(1):73–84, 1998.

[13] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.

[14] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[15] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Technical Report AI-01-290, Department of Computer Sciences, The University of Texas at Austin, 2001.

[16] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[17] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

[18] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.