Programação Evolutiva em Redes Prototípicas

Gabriel Sales
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
gabriel.sales@dcc.ufmg.br

Adriano Veloso
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
adrianov@dcc.ufmg.br

***Abstract*—Este trabalho apresenta a aplicação da Programação Evolutiva para otimizar pesos de Redes Prototípicas utilizando uma arquitetura adaptada da AlexNet. O objetivo é explorar a combinação de algoritmos evolutivos com redes neurais para melhorar a eficiência em tarefas de classificação de imagens em cenários de few-shot learning. Foram utilizados os conjuntos de dados Oxford 102 Flowers e Omniglot para validar a abordagem. Os resultados demonstram que o algoritmo evolutivo proposto supera os métodos tradicionais de treinamento de redes neurais, alcançando maior precisão na classificação de imagens com poucos exemplos de treinamento. Este desempenho superior é atribuído à capacidade do algoritmo evolutivo de evitar mínimos locais e explorar melhor o espaço de busca de soluções.**

***Index Terms*—Programação Evolutiva, Redes Prototípicas, Meta-Aprendizado, Few-Shot Learning, AlexNet, Classificação de Imagens.**

I. Introdução

Nos últimos anos, o campo da inteligência artificial tem testemunhado avanços notáveis, especialmente no que diz respeito às Redes Neurais Artificiais. Estas redes, inspiradas no funcionamento do cérebro humano, têm se mostrado uma ferramenta poderosa para resolver uma variedade de problemas complexos, desde reconhecimento de padrões até a tomada de decisões autônomas [1]. No entanto, o desenvolvimento de redes neurais eficientes e eficazes continua sendo um desafio significativo devido à complexidade das tarefas que elas enfrentam e à necessidade de otimizar sua arquitetura e parâmetros [2].

Dentro do âmbito das redes neurais, elas são projetadas para abordar problemas nos quais os seres humanos demonstram excelência, mas que são notoriamente difíceis de serem resolvidos por meio de abordagens programáticas convencionais. Durante o treinamento desses modelos, fazemos uso de funções de perda e otimizadores; contudo, esse processo não garante que a arquitetura da rede alcance um desempenho ótimo, visto que está sujeita a convergir para mínimos locais ou pontos de sela [3].

Além disso, esses modelos exigem uma vasta quantidade de exemplos de treino, em contraste com os seres humanos, que podem aprender novas tarefas com relativamente poucos exemplos. O conceito de meta-aprendizado almeja diminuir o número de exemplos necessários para aprender uma nova tarefa, treinando o modelo para várias tarefas diferentes [4].

Nesse contexto, este projeto de pesquisa tem como objetivo investigar novas formas de fazer um modelo aprender a aprender. O algoritmo proposto será constituido da utilização da Programação Evolutiva para otimizar os pesos de uma Rede Prototípica [5] que utilizará uma AlexNet [15] adaptada como esqueleto.

A Programação Evolutiva é uma metodologia que ganhou destaque na otimização de sistemas complexos, fundamentando-se em princípios inspirados na evolução biológica, tais como seleção natural, reprodução e mutação, para buscar soluções eficientes em espaços de busca de alta dimensionalidade [6]. A combinação da algoritmos evolutivos com redes neurais pode aprimorar o treinamento e a otimização delas, tornando-as mais generalistas e eficientes [7].

O algoritmo proposto demonstrou performance superior em comparação a cenários equivalentes, como o treinamento de múltiplos modelos e o treinamento de um único modelo utilizando o mesmo recurso computacional. Esses resultados foram obtidos utilizando as bases de dados Oxford 102 Flower [8] e Omniglot [9], demonstrando a eficácia da abordagem proposta em diferentes contextos de classificação de imagens.

II. Referencial Teórico

A. Gradiente Descendente

O Método do Gradiente Descendente é uma estratégia popular utilizada para minimizar uma função objetivo $J(\theta)$ que depende dos parâmetros θ de um modelo, onde θ pertence ao conjunto $\Re^d$. Esse método funciona atualizando os parâmetros na direção oposta ao gradiente da função objetivo $\nabla_\theta J(\theta)$. A taxa de aprendizado η determina a magnitude das atualizações dos parâmetros para aproximar de um mínimo local [10].

O algoritmo tem diversas variantes, entre as mais comuns estão: Gradiente Descendente em Lote; Gradiente Descendente Estocástico e o Gradiente Descendente em Mini Lotes.

1) Gradiente Descendente em Lote: O Gradiente Descendente em Lote (Batch Gradient Descent, BGD) é a variante original e calcula o gradiente da função objetivo para todo o conjunto de treino:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

Observe que todo o conjunto de dados é alocado na memória, portanto este método não é aconselhado quando o conjunto de treino não cabe na memória. Além disso, o tempo gasto por cada atualização de parâmetros é maior, o que pode

fazer com que o modelo demore mais tempo para atingir um mínimo local [10].

2) Gradiente Descendente Estocástico: O Gradiente Descendente Estocástico (SGD) é uma variante do algoritmo de otimização do Gradiente Descendente que acelera o treinamento de modelos de aprendizado de máquina.

Em vez de calcular o gradiente da função de perda usando todos os dados a cada iteração, o SGD realiza atualizações de parâmetros com base em amostras individuais ($x^{(i)}$ e $y^{(i)}$), selecionadas aleatoriamente, tornando o processo menos computacionalmente intensivo e permitindo uma convergência mais rápida em muitos casos.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Contudo, com as atualizações individuais o modelo é atualizado com uma variância alta, o que dificulta a convergência para um mínimo local, em contrapartida isso permite que o modelo explore outros vales e consiga melhorar sua solução [10].

3) Gradiente Descendente em Mini Lotes: O Gradiente Descendente em Mini Lotes utiliza um número intermediário, n, de exemplos para calcular o gradiente da função de perda:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Sua principal vantagem reside na combinação dos benefícios do Gradiente Descendente Estocástico e do Gradiente Descendente em Lotes. Ao dividir o conjunto de dados em pequenos lotes, ele oferece uma abordagem eficaz para otimização que combina eficiência computacional e convergência estável. Isso permite que os algoritmos de aprendizado de máquina atualizem os parâmetros do modelo de forma mais frequente e rápida em comparação com o Gradiente Descendente em Lotes, acelerando o processo de treinamento. Além disso, a variabilidade introduzida pelos mini lotes ajuda a evitar mínimos locais indesejados, melhorando a capacidade do algoritmo de escapar de ótimos locais subótimos [10].

B. Adam

O otimizador Adam (Adaptive Moment Estimation) é um algoritmo de otimização popular que combina os conceitos do SGD com momentos adaptativos. Isso permite que o otimizador ajuste a taxa de aprendizado de forma adaptativa para cada parâmetro, acelerando o treinamento no início e desacelerando à medida que se aproxima do ótimo, resultando em uma convergência mais eficiente e estável [11] e [10].

O Adam mantém duas estimativas móveis: o primeiro momento, média dos gradientes, e o segundo momento, média dos gradientes ao quadrado, das variáveis do modelo:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Como essas estimativas são nulas no começo do algoritmo, seus valores são enviesados para o 0. Os autores propuseram remover esse viés através da seguinte modificação:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Utilizando esses valores, os parâmetros do modelos são atualizados da seguinte forma:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

C. Meta-Aprendizado

O conceito de Meta-Aprendizado ainda não é acordado entre os acadêmicos, contudo eles concordam que este aprendizado procura capacitar um modelo a aprender a aprender. Em vez de treinar um modelo para executar uma tarefa específica, o objetivo do meta aprendizado é treinar um modelo em uma variedade de tarefas de aprendizado, de modo que ele possa adaptar rapidamente seus parâmetros a novas tarefas com base em apenas em um número pequeno de exemplos de treinamento [12].

Para que isso ocorra, é necessário treinar o modelo de forma que após algumas etapas de gradiente, ou até mesmo apenas uma única etapa de gradiente, ele seja capaz de produzir resultados satisfatórios em uma nova tarefa. Isso pode ser compreendido do ponto de vista técnico como a construção de uma representação interna que seja aplicável a diversas tarefas. Essa representação interna, quando adequada, permite que simples ajustes nos parâmetros, como a modificação dos pesos da camada superior em um modelo feedforward, sejam suficientes para obter bons resultados [4].

O meta-aprendizado é particularmente relevante em cenários em que o acesso a grandes conjuntos de dados de treinamento é limitado, tornando-o uma técnica valiosa para o aprendizado de máquina em situações de poucos dados [13]. Um paradigma comum no meta-aprendizado é o N-way K-shot, onde o modelo é treinado em episódios. Cada episódio consiste em aprender a classificar N classes diferentes, com apenas K exemplos de cada classe. Esses episódios são divididos em dois conjuntos: um conjunto de suporte com poucos exemplos por classe, que é utilizado para o aprendizado rápido da tarefa, e um conjunto de consulta com exemplos adicionais que são utilizados para avaliar o desempenho do modelo naquela tarefa.

D. Redes Prototípicas

As Redes Prototípicas são uma arquitetura de redes neurais artificiais para few-shot learning [5]. Elas possuem um modelo esqueleto que processa as entradas e gera uma representação da entrada em um espaço latente. Para um episódio de k exemplos de n classes de suporte, a rede prototípica processa cada um desses exemplos e calcula um protótipo, um ponto âncora, para cada classe no espaço latente. Este modelo classifica os dados de consulta verificando qual âncora está mais próxima da representação desse dado no espaço latente.

Matematicamente, para cada classe c com k exemplos de suporte $\{(x_1^c, y_1^c), (x_2^c, y_2^c), \ldots, (x_k^c, y_k^c)\}$, onde $y_i^c = c$, a

rede prototípica calcula o protótipo $\mathbf{p}_c$ como a média das representações das entradas no espaço latente:

$$\mathbf{p}_c = \frac{1}{k} \sum_{i=1}^{k} f_\theta(x_i^c)$$

onde f_θ é a função de embedding parametrizada pela rede neural com parâmetros θ.

Para um dado de consulta x, a rede classifica x atribuindo-lhe o rótulo da classe cujo protótipo está mais próximo da representação de x no espaço latente:

$$\hat{y} = \arg\min_c \|f_\theta(x) - \mathbf{p}_c\|$$

onde $\|\cdot\|$ denota a norma euclidiana.

E. Programação Evolutiva

A Programação Evolutiva é uma técnica de otimização inspirada no processo de evolução biológica. O conceito central por trás dela é simular um processo evolutivo em um ambiente controlado, onde populações de soluções candidatas passam por uma sequência de gerações para encontrar a melhor solução possível para um problema específico [14].

Um algoritmo de programação evolutiva pode ser definido nas etapas a seguir [6] e [14]:

1) Inicialização da População: No início, uma população de soluções candidatas é gerada aleatoriamente. Cada solução representa uma possível resposta para o problema em questão.

2) Avaliação: As soluções candidatas são avaliadas quanto à sua adequação em relação ao objetivo. Isso é feito usando uma função de avaliação que atribui uma pontuação a cada solução com base em quão bem ela resolve o problema.

3) Seleção: As soluções mais adequadas, ou seja, aquelas com as melhores pontuações, são selecionadas para "reprodução". Soluções de alto desempenho têm uma maior probabilidade de serem escolhidas, mas uma variedade de soluções é mantida para manter a diversidade genética.

4) Recombinação: Pares de soluções selecionadas são combinados para criar novas soluções. Isso envolve a troca de informações genéticas entre as soluções país, criando uma descendência que herda características das soluções originais.

5) Mutação: Algumas das novas soluções geradas por recombinação sofrem mutações aleatórias, introduzindo variação genética na população.

6) Substituição: As novas soluções resultantes da recombinação e mutação substituem parte da população anterior. As soluções menos adequadas podem ser eliminadas.

7) Critério de Parada: O processo de avaliação, seleção e geração de descendentes é repetido por várias gerações até que um critério de parada seja atingido. Isso pode ser um número fixo de gerações, uma melhoria na qualidade da solução ou outras condições específicas.

III. Dados

A. Seleção das bases de dados

A seleção das bases de dados foi conduzida com base nos seguintes critérios: a presença de um grande número de classes e a existência de uma variedade de classes, abrangendo tanto aquelas que são semanticamente similares quanto aquelas que são distintas. Diante desses critérios, as bases escolhidas foram Oxford 102 Flowers [8] e Omniglot [9].

1) Oxford 102 Flowers: O conjunto de dados Oxford 102 Flowers é um benchmark para algoritmos de visão computacional, consistindo em imagens de alta resolução de 102 categorias de flores diversas [8]. A variedade de espécies, cores e formas presentes no conjunto de dados o torna ideal para testar a capacidade de generalização de modelos de classificação de imagens.

2) Omniglot: O Omniglot é um conjunto de dados desafiador para aprendizado de máquina e meta-aprendizado, consistindo em 1.623 caracteres de 50 alfabetos diferentes [9]. Sua diversidade e baixa resolução das imagens (28x28 pixels em escala de cinza) o tornam ideal para avaliar a capacidade de generalização e adaptação de modelos de reconhecimento de caracteres.

B. Pré-processamento dos dados

Os conjuntos de dados foram pré-processados para adequar-se ao formato de entrada de uma Rede Prototípica, que opera com episódios contendo k exemplos de n classes distintas, caracterizando um cenário de aprendizado few-shot. Cada episódio é dividido em duas partes: um conjunto de suporte, contendo k exemplos de cada uma das n classes para formar as representações prototípicas, e um conjunto de consulta, com novos exemplos a serem classificados com base nos protótipos aprendidos. Os dados foram divididos em conjuntos de treino (70%), validação (15%) e teste (15%), garantindo que cada conjunto contenha classes únicas.

IV. Rede Prototípica

A arquitetura escolhida para o esqueleto da Rede Prototípica [5] foi a AlexNet [15], uma arquitetura do tipo Rede Neural Convolucional (CNN) [16]. A AlexNet demonstrou um desempenho notável em diversas tarefas de visão computacional. Além disso, a sua arquitetura é relativamente compacta comparada a modelos mais recentes, e a torna uma escolha atraente para cenários com recursos computacionais limitados. Adicionalmente, o sucesso das CNNs em tarefas de transferência de aprendizado, onde características aprendidas em um grande conjunto de dados (como o ImageNet) são reutilizadas em novas tarefas [17], a torna uma base sólida para a construção da Rede Prototípica, especialmente em cenários com dados e recursos computacionais limitados.

V. Algoritmo Evolutivo

O algoritmo proposto, Algoritmo 1, percorre as seguintes fases, que são características de um algoritmo de programação evolutiva:

Algorithm 1 Algoritmo Evolutivo de Redes Prototípicas

Require: P: Tamanho da população
Require: G: Número de gerações
Require: T: Conjunto de dados de treinamento
Require: V: Conjunto de dados de validação
Inicializar uma população pop com P redes neurais com pesos aleatórios
for $g \leftarrow 1$ to G **do**
  for $p \in pop$ **do**
    Treina p usando T
    Avaliar p usando T e V
  end for
  Selecionar os melhores modelos para formar $parents$
  $new\_pop \leftarrow parents$
  while $|new\_pop| < P$ **do**
    Selecionar $p_1, p_2 \in parents$ com base na aptidão
    Realizar cruzamento entre p_1 e p_2 para gerar c
    Adicionar c a new_pop
  end while
  $pop \leftarrow new\_pop$
end for
return A melhor rede neural em pop baseada na aptidão

A. Inicialização

Uma população inicial de Redes Prototípicas, implementadas a partir de uma AlexNet adaptada como esqueleto, é inicializada aleatoriamente. A AlexNet tem suas camadas totalmente conectadas modificadas para produzir uma representação da entrada em um espaço latente. O tamanho da população é determinado por um hiperparâmetro.

B. Treinamento

A população de Redes Prototípicas é treinada no conjunto de dados de treino utilizando o otimizador Adam. A quantidade de épocas de treinamento é definida por um hiperparâmetro.

C. Avaliação

A população é avaliada de acordo com a seguinte função:

$$\text{Model}_{\text{Score}} = \alpha \cdot \text{Train}_{\text{Accuracy}} + \text{Validation}_{\text{Accuracy}}$$

onde a entrada consiste na acurácia do modelo no conjunto de treino e no conjunto de validação. Testes demonstraram que um valor adequado para α está entre $-0,1$ e $0,1$.

D. Seleção

Os modelos com melhor desempenho, representando a metade superior da população, são selecionados para a etapa de reprodução. A outra metade, com desempenho inferior, é removida da população, abrindo espaço para novas soluções na próxima geração.

E. Reprodução

Os modelos selecionados são reproduzidos de acordo com a seguinte fórmula e adicionados a população:

$$W_C = \frac{W_A + \beta W_B}{1 + \beta}$$

onde W_A e W_B são os pesos dos modelos pais, W_C é o peso do modelo filho, e β é um hiperparâmetro. Testes demonstraram que um bom valor para β é $0,025$.

F. Critério de Parada

As etapas de Treinamento, Avaliação, Seleção e Reprodução são repetidas por G gerações ou até que uma condição de parada seja satisfeita, sendo G mais um hiperparâmetro.

VI. Resultados

O algoritmo proposto foi avaliado em comparação a dois cenários de referência: treinamento paralelo de múltiplos modelos com seleção do melhor e treinamento de um único modelo utilizando recursos computacionais equivalentes aos do algoritmo proposto. Os resultados de desempenho (acurácia) nos conjuntos de dados Flowers-102 e Omniglot são apresentados nas Tabelas 1 e 2, respectivamente. Devido à alta demanda computacional dos cenários, apenas uma execução foi realizada para cada conjunto de dados, utilizando uma GPU Nvidia L4, com tempo de execução aproximado de 6 horas por conjunto de dados.

TABLE I
Desempenho dos cenários no Flowers-102

	Evolutivo	Multiplos Modelos	Único Modelo
5-way 5-shot	**0.5936**	0.5154	0.4860
5-way 3-shot	**0.5038**	0.4832	0.4390
5-way 1-shot	**0.4744**	0.4540	0.4024

TABLE II
Desempenho dos cenários no Omniglot

	Evolutivo	Multiplos Modelos	Único Modelo
5-way 5-shot	**0.5878**	0.4814	0.4154
5-way 3-shot	**0.6278**	0.4584	0.4212
5-way 1-shot	**0.4496**	0.4048	0.4140

Os resultados obtidos pelo algoritmo evolutivo foram consistentemente superiores aos demais cenários, demonstrando sua eficácia. O cenário com múltiplos modelos também apresentou desempenho superior ao cenário com um único modelo, confirmando a importância da diversidade na busca por soluções.

Uma possível justificativa para o desempenho superior do cenário evolutivo é que a combinação de modelos permite escapar de pontos de sela ou mínimos locais, ampliando o espaço de busca e possibilitando a descoberta de soluções mais otimizadas.

VII. Conclusões

Neste trabalho, investigamos a aplicação de um algoritmo de Programação Evolutiva para otimização dos pesos de Redes Prototípicas, utilizando a arquitetura AlexNet adaptada. Nossa abordagem foi avaliada em cenários de aprendizado few-shot com as bases de dados Oxford 102 Flowers e Omniglot, demonstrando desempenho superior quando comparada a métodos tradicionais de treinamento paralelo e treinamento único.

Os resultados evidenciam que a combinação de técnicas evolutivas com redes neurais pode efetivamente melhorar a capacidade de generalização e a eficiência de modelos de aprendizado, particularmente em cenários com dados limitados. A Programação Evolutiva se mostrou uma metodologia promissora para otimização de modelos complexos, oferecendo vantagens significativas na busca por soluções em espaços de alta dimensionalidade.

Futuras direções de pesquisa incluem a exploração de outras arquiteturas de redes neurais, o ajuste fino dos hiperparâmetros do algoritmo evolutivo e a aplicação desta abordagem em diferentes domínios de aprendizado, como processamento de linguagem natural e análise de séries temporais. Além disso, a investigação de técnicas híbridas que combinam Programação Evolutiva com outros métodos de otimização pode proporcionar avanços adicionais no campo do aprendizado de máquina.

Esta pesquisa contribui para o avanço do estado da arte em meta-aprendizado e otimização de redes neurais, oferecendo insights valiosos para o desenvolvimento de modelos mais eficientes e generalizáveis em uma variedade de contextos aplicados.

References

[1] Oludare Isaac Abiodun, Aman Jantan, A. E. O. K. V. D. N. A. M. H. A. (2018). State-of- the-art in artificial neural network applications: A survey. Heliyon.

[2] Mingxing Tan, Q. V. L. (2020). Efficientnet: Rethinking model scaling for convolutional neural networks.

[3] Yann N. Dauphin, Razvan Pascanu, C. G. K. C. S. G. Y. B. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.

[4] Chelsea Finn, Pieter Abbeel, S. L. (2017). Model-agnostic meta-learning for fast adapta- tion of deep networks.

[5] Jake Snell, Kevin Swersky, R. S. Z. (2017). Prototypical networks for few-shot learning.

[6] Mitchell, M. (1995). Genetic algorithms: An overview. Complexity.

[7] Jatinder N. D. Gupta, R. S. S. (1999). Comparing backpropagation with a genetic algorithm for neural network training. Omega.

[8] Nilsback, M.-E. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In Indian Conference on Computer Vision, Graphics and Image Processing.

[9] Brenden M. Lake, Ruslan Salakhutdinov, J. B. T. (2015). Human-level concept learning through probabilistic program induction. In Science, pages 1332–1338. 350 edition.

[10] Ruder, S. (2017). An overview of gradient descent optimization algorithms.

[11] Diederik P. Kingma, J. B. (2017). Adam: A method for stochastic optimization.

[12] Ricardo Vilalta, Y. D. (2002). A perspective view and survey of meta-learning. Artificial Intelligence Review.

[13] Lisha Chen, Sharu Theresa Jose, I. N. S. P. T. C. O. S. (2022). Learning with limited samples: Meta-learning and applications to communication systems.

[14] Kramer, O. (2017). Genetic Algorithm Essentials, volume 679. Studies in Computational Intelligence.

[15] Alex Krizhevsky, Ilya Sutskever, G. E. H. (2012). Imagenet classification with deep convolutional neural networks.

[16] Keiron O'Shea, R. N. (2015). An introduction to convolutional neural networks.

[17] Jason Yosinski, Jeff Clune, Y. B. H. L. (2014). How transferable are features in deep neural networks?