Utilização da Programação Evolutiva em Redes Neurais para Meta-Aprendizado

Pesquisa Científica - Projeto Orientado a Computação II

Aluno: Gabriel M. M. Sales[1], Orientador: Adriano A. Veloso[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

{gabriel.sales, adrianov}@dcc.ufmg.br

1. Introdução

Nos últimos anos, o campo da inteligência artificial tem testemunhado avanços notáveis, especialmente no que diz respeito às Redes Neurais Artificiais (RNAs). As RNAs, 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 [Oludare Isaac Abiodun 2018]. No entanto, o desenvolvimento de RNAs 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.

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.

Além disso, estes modelos necessitam de uma grande quantidade de exemplos de treino, enquanto nós, seres humanos, não precisamos de muitos exemplos para aprender uma nova tarefa. O conceito de meta-aprendizado almeja diminuir o número de exemplos necessários para aprender uma nova tarefa, treinando o modelo para várias tarefas diferentes [Chelsea Finn 2017].

Nesse contexto, este projeto de pesquisa tem como objetivo investigar novas formas de Meta-Aprendizado (MA). A estratégia de aprendizado a ser explorada será a utilização da programação evolutiva para otimizar os pesos de uma rede neural.

A programação evolutiva (PE) 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 [Mitchell 1995]. A combinação da PE com RNAs pode aprimorar o treinamento e a otimização de RNAs, tornando-as mais generalistas e eficientes [Jatinder N. D. Gupta 1999].

No âmbito da disciplina Projeto Orientado em Computação I (POC I), foram selecionadas as bases de dados Omniglot [Brenden M. Lake 2015] e Oxford 102 Flower [Nilsback and Zisserman 2008] para o treinamento e avaliação de modelos propostos.

A escolha das referidas bases se deu em função da presença de um elevado número de classes, bem como da existência de uma ampla variedade de classes, englobando tanto aquelas que apresentam similaridade semântica quanto aquelas que são distintas.

Adicionalmente, explorou-se a viabilidade da PE para otimizar pesos de RNAs no contexto do MA. Foi implementado parcialmente um algoritmo de MA com PE que será aprimorado nesta disciplina de Projeto Orientado em Computação II (POC II). Investigar-se-á se a utilização da técnica de Layer-wise Relevance Propagation [Grégoire Montavon 2019] para realizar a reprodução dos modelos resultará em modelos com melhor performance em múltiplas tarefas. Até o final da disciplina, também será realizada uma comparação meticulosa com abordagens existentes para otimização de RNAs através de testes de hipóteses.

2. Referencial Teórico

2.1. 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 [Ruder 2017].

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

2.1.1. 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:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{1}$$

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 [Ruder 2017].

2.1.2. 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.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2}$$

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 [Ruder 2017].

2.1.3. 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:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{3}$$

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 [Ruder 2017].

2.2. 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 [Diederik P. Kingma 2017] e [Ruder 2017].

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:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{4}$$

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:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{5}$$

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

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{6}$$

2.3. Meta-Aprendizado

O conceito de Meta-Aprendizado (MA) ainda não é acordado entre os acadêmicos, contudo eles concordam que o MA 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 [Ricardo Vilalta 2002].

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 [Chelsea Finn 2017].

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 [Lisha Chen 2022].

2.4. Programação Evolutiva

A Programação Evolutiva (PE) é uma técnica de otimização inspirada no processo de evolução biológica. O conceito central por trás da PE 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 [Kramer 2017].

Um algoritmo de programação evolutiva pode ser definido nas etapas a seguir [Mitchell 1995] e [Kramer 2017]:

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 (Crossover): 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.

2.5. Layer-wise Relevance Propagation

A Layer-wise Relevance Propagation (LRP) é uma técnica utilizada para interpretar as decisões de uma rede neural, atribuindo importância a cada entrada do modelo com relação à saída final. Desenvolvida como uma abordagem de explicabilidade em redes neurais, a LRP busca decompor a saída da rede em contribuições de cada entrada, permitindo entender quais características ou entradas foram mais relevantes para uma decisão específica [Grégoire Montavon 2019].

A abordagem da LRP é baseada no princípio de conservação de relevância, que estabelece que a soma das importâncias atribuídas a todas as entradas deve ser igual à saída da rede. Dessa forma, a relevância é propagada camada por camada, da saída para a entrada, atribuindo valores proporcionais à influência de cada neurônio em relação à saída final.

A relevância, $R_i^{(l)}$, de um neurônio i de uma camada l, pode ser calculada com a seguinte fórmula:

$$R_i^{(l)} = \sum_j \frac{z_{ij} R_j^{(l+1)}}{\sum_{i'} z_{i'j}} \tag{7}$$

em que z_{ij} é igual a $x_i^{(l)} w_{ij}^{(l,l+1)}$.

A LRP é valiosa não apenas para interpretar as decisões da rede, mas também para identificar potenciais fontes de viés e entender como diferentes entradas contribuem para o resultado final. Essa técnica tem sido aplicada em diversas áreas, desde visão computacional até processamento de linguagem natural, proporcionando insights valiosos sobre o funcionamento interno de modelos complexos de aprendizado profundo.

3. Metodologia

Este projeto se divide em quatro etapas: ”Implementação do Layer-wise Relevance Propagation”, ”Formulação das Hipóteses”, ”Execução dos Modelos” e ”Analise dos Resulta-

dos".

3.1. Implementação do Layer-wise Relevance Propagation

No contexto da etapa de reprodução de dois modelos de RNAs no algoritmo de PE, diversas metodologias podem ser utilizadas para a combinação dos pesos dos modelos. A seguir, são apresentadas algumas abordagens triviais:

$$W_C = \frac{W_A + W_B}{2}$$

$$W_C = \frac{W_A S_A + W_B S_B}{S_A + S_B} \tag{8}$$

$$W_C = \frac{W_A |W_A| + W_B |W_B|}{|W_A| + |W_B|}$$

onde W_A e W_B são os pesos dos modelos pais, W_C é o peso do modelo filho, S_A e S_B são os scores dos modelos pais. Lembrando que o score de um modelo foi definido da seguinte forma em POC I:

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

onde α é um hiperparâmetro.

Uma forma alternativa de combinar os pesos de dois modelos seria utilizar a relevância da LRP da seguinte forma:

$$W_C = \frac{W_A R_A + W_B R_B}{R_A + R_B}$$

onde R_A e R_B são as relevâncias dos modelos pais. Esta alternativa tem o potencial de gerar modelos que preservam a estrutura dos pesos dos modelos pais que apresentam maior ativação.

Nesta etapa, será realizada a implementação do algoritmo de LRP e subsequentemente implementação da combinação de modelos utilizando a relevância como peso.

3.2. Formulação das Hipóteses

Nesta etapa, serão formulados os testes de hipóteses necessários para verificar se o algoritmo proposto apresenta algum ganho em desempenho no contexto de MA. Um exemplo de hipótese seria: sob as mesmas condições computacionais, o modelo proposto treinado por uma quantidade específica de tempo apresenta desempenho superior a um modelo convencional treinado pelo mesmo período de tempo.

Adicionalmente à formulação dos testes, será necessário analisar sua viabilidade, considerando os recursos computacionais e o tempo disponível. Um modelo base de RNA com menor número de parâmetros pode ser treinado e testado mais vezes em um

período de tempo fixo, o que resultaria em um intervalo de confiança menor por existirem mais amostras. No entanto, a diferença de desempenho entre o algoritmo proposto e o convencional pode ser menor, exigindo mais treinos e testes. Pontanto, nesta etapa também se fará necessario ponderar sobre a escolha do modelo base.

3.3. Execução dos Modelos

Nesta etapa, a implementação do algoritmo MA é posta em prática, levando em consideração a influência das condições computacionais. Durante esta etapa, é possível adaptar o algoritmo de MA e realizar ajustes nos hiperparâmetros tanto da PE quanto da RNA com o objetivo de otimizar o desempenho.

Durante essa fase, os modelos serão treinados nas bases de treino e, subsequentemente, serão executados em um conjunto de teste para avaliar sua capacidade de generalização e seu desempenho em cenários do mundo real. Para fins de comparação e validação dos resultados obtidos com a nova metodologia, também serão realizados testes com abordagens convencionais de otimização de pesos de RNAs.

3.4. Analise dos Resultados

Comparar o desempenho das RNAs geradas com abordagens convencionais de treinamento em novas tarefas. Discutir as vantagens e desvantagens da abordagem de meta-aprendizado proposta em relação a outros métodos. Criar gráficos que ilustram e comprovam as vantagens e desvantagens mencionadas.

4. Resultados Esperados

Os resultados esperados até o final do projeto de POC II envolvem realizar uma análise das técnicas de otimização convencionais aplicadas às RNAs para MA, permitindo uma comparação minuciosa de suas vantagens e desvantagens em termos de convergência, eficiência computacional e capacidade de generalização em relação ao algoritmo proposto que envolve PE.

Espera-se que os resultados contribuam significativamente para o conhecimento científico, podendo ser disseminados em artigos e conferências, enquanto também se espera que tenham aplicações práticas em cenários do mundo real. No geral, o projeto tem o potencial de aprimorar a compreensão do meta-aprendizado e fornecer soluções valiosas para problemas de otimização de redes neurais.

5. Etapas e Cronogramas

O desenvolvimento deste projeto em POC II será separado nas etapas citadas na Metodologia e seguirá o seguinte cronograma:

Etapa	**Tempo em Semanas**
Implementação do Layer-wise Relevance Propagation	4
Formulação das Hipóteses	2
Execução dos Modelos	3
Analise dos Resultados	3
Total	**12**

Assim, o cronograma completo do projeto, considerando POC I e II, ficará da seguinte forma:

Etapa	Tempo em Semanas
Seleção das Arquiteturas	3
Seleção das Bases de Dados	3
Pré-processamento dos Dados	3
Implementação do Pipeline de PE	3
POC I	**12**
Implementação do Layer-wise Relevance Propagation	4
Formulação das Hipóteses	2
Execução dos Modelos	3
Analise dos Resultados	3
POC II	**12**
Total	**24**

References

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

Chelsea Finn, Pieter Abbeel, S. L. (2017). Model-agnostic meta-learning for fast adaptation of deep networks.

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

Grégoire Montavon, Alexander Binder, S. L. W. S. . K.-R. M. (2019). Layer-wise relevance propagation:an overview. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*.

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

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

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.

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

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*.

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.

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

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