Uma proposta de algoritmo para treinamento de redes neurais artificiais: primeiros resultados em uma comparação com SGD, Adam e AdraGrad

Jean Vinicius da Silva Montanari
Graduando em Matemática
Universidade do Estado de Mato Grosso (UNEMAT)
Campus de Sinop - MT
vinicius.montanari@unemat.br

Luciana Mafalda Elias de Assis
Professora da Universidade do Estado de Mato Grosso (UNEMAT)
Campus de Sinop - MT
luciana.assis@unemat.br

Raul Abreu de Assis
Professor da Universidade do Estado de Mato Grosso (UNEMAT)
Campus de Sinop - MT
raul.assis@unemat.br

ABSTRACT

This paper presents a new supervised training algorithm for artificial neural networks, developed with the aim of improving network performance in classification tasks, where accuracy is crucial. The method is compared with the optimization algorithms Stochastic Gradient Descent, ADAM, and ADAGRAD. Experiments were conducted using different datasets and network architectures. The metrics used for evaluation were the number of errors and accuracy during the training and testing phases. The results indicate that the proposed method shows competitive performance, with specific advantages in certain contexts, making it a promising alternative to traditional optimization algorithms.

Keywords

Neural networks; Optimization; Training; Machine learning.

RESUMO

Este artigo apresenta um novo algoritmo de treinamento supervisionado para redes neurais artificiais, desenvolvido com o objetivo de melhorar o desempenho de redes em tarefas de classificação, em que a acurácia é fundamental. O método é comparado aos algoritmos de otimização *Stochastic Gradient Descent,* ADAM e ADAGRAD. Foram realizados experimentos com diferentes conjuntos de dados e arquiteturas de rede. As métricas utilizadas para avaliação foram o número de erros e a acurácia, durante as fases de treinamento e teste. Os resultados indicam que o método proposto apresenta desempenho competitivo, com vantagens específicas em determinados contextos, sendo uma alternativa promissora aos algoritmos tradicionais de otimização.

Palavras-chave

Redes neurais; Otimização; Treinamento; Aprendizado de Máquina.Neural.

1. INTRODUÇÃO

O avanço das redes neurais artificiais tem impulsionado significativamente o desenvolvimento de soluções na área de aprendizado de máquina, especialmente em tarefas como classificação, reconhecimento de padrões e processamento de dados complexos.

O desempenho dessas redes está diretamente relacionado à escolha de algoritmos de otimização capazes de ajustar os parâmetros do modelo de forma eficiente, garantindo uma boa capacidade de generalização.

Este trabalho aborda especificamente uma proposta de algoritmo de treinamento supervisionado em tarefas de classificação. Uma comparação é feita com os algoritmos de treinamentos baseados no Gradiente Estocástico com Minilotes, uma técnica amplamente utilizada por proporcionar equilíbrio entre estabilidade na convergência e custo computacional.

Além desse método, também são analisados algoritmos adaptativos como ADAM e ADAGRAD. Nossa proposta de algoritmo, denominada WEGRAD, introduz um mecanismo de ponderação do gradiente com o objetivo de melhorar o desempenho com foco na acurácia, isto é, a classificação de forma correta do maior número possível de pontos da base de dados.

A escolha desse objeto de estudo justifica-se pela ampla utilização das redes neurais em diferentes áreas e pela necessidade constante de aprimorar os métodos de treinamento, tornando-os mais eficientes, precisos e capazes de lidar com diferentes tipos de dados e problemas. Estudar e compreender o comportamento desses algoritmos é essencial para o desenvolvimento de modelos mais robustos e eficientes.

Dessa forma, o objetivo deste trabalho é analisar, propor e comparar algoritmos de otimização aplicados ao treinamento de redes neurais, avaliando seus desempenhos com foco na acurácia, em diferentes bases de dados e variando um parâmetro fundamental, a taxa de aprendizagem.

2. FUNDAMENTAÇÃO TEÓRICA

Os modelos de redes utilizadas neste trabalho serão baseados em redes do tipo perceptron, que revisamos rapidamente, a seguir.

2.1 Rede Perceptron

Segundo [8], o *Perceptron*, idealizado por Rosenblatt em 1958, é a forma mais simples de uma rede neural artificial, inspirado na retina e utilizado para reconhecer padrões geométricos.

A Figura 1(adaptada de [8], p.58), ilustra uma rede *Perceptron* constituída de n sinais de entrada, representativas do problema a

ser mapeado, e somente uma saída, pois a mesma é constituída de somente um único neurônio.

Apesar de ser uma rede considerada simples, o *Perceptron* despertou grande interesse da comunidade científica desde sua proposição, atraindo pesquisadores interessados no potencial da área de redes neurais e inteligência artificial [8].

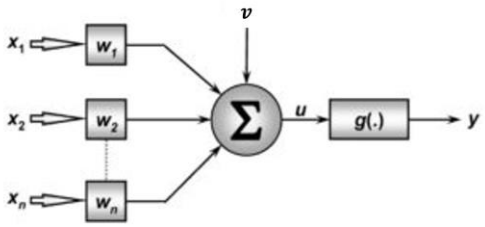

Figura 1: Ilustração da rede *Perceptron.*

Em termos matemáticos, o processamento interno realizado pelo *Perceptron* pode ser descrito pelas seguintes expressões:

$$u = \sum_1^n x_i \cdot w_i + v, \quad (1)$$

na qual v é chamado de parâmetro de "viés" e w_i são chamados "pesos sinápticos". O sinal do neurônio é dado então através de uma função de ativação $g(u)$:

$$y = g(u). \quad (2)$$

No primeiro passo do funcionamento do *Perceptron*, é realizado o cálculo do potencial de ativação do neurônio, representado por u. Esse valor é obtido por meio do somatório ponderado dos sinais de entrada, em que cada entrada x_i é multiplicada pelo respectivo peso sináptico w_i, refletindo importância daquela entrada para o processo de decisão do neurônio. A esse somatório é adicionado o termo de viés v, responsável por deslocar a função de ativação, permitindo à rede ajustar sua fronteira de decisão mesmo quando todos os valores de entrada forem nulos. Dessa forma, o *Perceptron* simples se apresenta como um modelo fundamental para o entendimento das redes neurais, sendo capaz de realizar classificações lineares com base em uma combinação ponderada das entradas e uma função de ativação.

2.2 Redes Perceptron Multicamadas

A fim de superar as limitações do *Perceptron* simples, foi desenvolvido o *Perceptron* Multicamadas. Segundo [8] (p.91), "As Redes *Perceptron* de múltiplas camadas (PMC) são caracterizadas pela presença de pelo menos uma camada intermediária (escondida) de neurônios, situada entre a camada de entrada e a respectiva camada neural de saída."

A Figura 2 (retirada de [8], p.92), representa uma estrutura pertencente à arquitetura *feedforward*. Nessa configuração, o funcionamento da rede neural ocorre de forma sequencial, em que as saídas de uma camada são utilizadas como entradas para a camada seguinte, até atingir a camada final [3].

De acordo com [8] (p. 93), "os estímulos ou sinais são apresentados à rede em sua de entrada [...] os neurônios da camada de saída recebem os estímulos advindos dos neurônios da última camada intermediária, produzindo um padrão de resposta que será a saída disponibilizada pela rede".

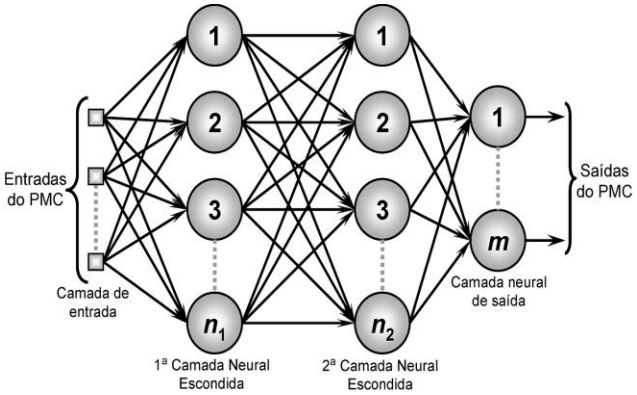

Figura 2: Representação de redes Perceptron Multicamadas.

Diferentemente do *Perceptron* simples, em que um único neurônio realiza o mapeamento concentrado do processo, as redes *Perceptron* Multicamadas (PMC) distribuem o conhecimento relacionado ao comportamento entrada/saída entre todos os seus neurônios. Nesse contexto, o problema real é modelado tanto pela camada de entrada, que recebe os estímulos iniciais, quanto pela camada de saída, que gera a resposta esperada.

2.3 Treinamento Supervisionado

O treinamento supervisionado é uma das estratégias mais utilizadas no campo do aprendizado de máquina e consiste em fornecer à rede neural um conjunto de dados de entrada com as respectivas saídas desejadas, denominados exemplos rotulados.

Conforme descrito por [8], essa abordagem exige a disponibilização de uma tabela de dados representativa, contendo os sinais de entrada e suas correspondentes saídas desejadas. Essa tabela, também conhecida como tabela de atributos/valores, deve ser capaz de refletir o comportamento do sistema a ser modelado, fornecendo subsídios suficientes para que as estruturas neurais possam formular "hipóteses" sobre o que se deve ser aprendido.

Nosso método aplica-se onde a tarefa da rede é classificar os dados em um conjunto discreto de classes. No caso em que temos m classes trabalhamos com m neurônios na camada de saída. Uma possível interpretação para o estado dos neurônios da camada de saída é a probabilidade de um certo dado estar na classe correspondente ao neurônio. Para a tarefa de classificação costuma-se simplesmente classificar um dado na classe com a maior probabilidade correspondente.

2.4 Algoritmo Gradiente Estocástico com Minilote (SGD)

É um algoritmo fundamental para o treinamento de redes neurais artificiais. Seu funcionamento baseia-se na aplicação da regra da cadeia do Cálculo Diferencial para calcular o gradiente do erro em relação aos pesos da rede.

O processo inicia-se na camada de saída, onde o erro entre a saída produzida pela rede e a saída desejada é calculado. Em seguida, esse erro é propagado para trás através das camadas da rede, ajustando os pesos de forma a minimizar a função de custo. Esse

ajuste é realizado com base em um algoritmo de otimização, como o gradiente descendente, que utiliza as derivadas parciais do erro para cada peso, atualizando-os de maneira proporcional ao valor desses gradientes. O objetivo é reduzir gradualmente o erro da rede ao longo de várias iterações do algoritmo, chamadas de "épocas" de treinamento.

Para otimizar o processo de treinamento e equilibrar o custo computacional com a estabilidade da convergência, é comum empregar a retropropagação com minilotes. Nessa abordagem, o conjunto de dados de treinamento é dividido em pequenos subconjuntos de amostras, denominados minilotes. As atualizações dos parâmetros são feitas, então, após a retropropagação ser realizada em cima de cada minilote.

2.5 Algoritmo ADAGRAD

De acordo com [2] e [7], o ADAGRAD é um algoritmo que realiza a adaptação individual das taxas de aprendizado para cada parâmetro do modelo, com base na soma acumulada dos quadrados dos gradientes. Esse mecanismo permite que parâmetros que recebem atualizações frequentes tenham suas taxas de aprendizado progressivamente reduzidas, enquanto parâmetros menos atualizados mantêm taxas relativamente maiores. No Quadro 1, apresentamos os passos requeridos pelo algoritmo ADAGRAD.

Quadro 1: Algoritmo ADAGRAD

Etapa 1	**Descrição:** Inicializar parâmetros **Fórmula / Ação:** $\theta_0 \leftarrow$ Valores iniciais
Etapa 2	**Descrição:** Inicializar acumulador de gradientes ao quadrado **Fórmula / Ação:** $G_0 \leftarrow 0$
Etapa 3	**Descrição:** Para cada iteração t=1 até T: **Fórmula / Ação:** —
Etapa 3.1	**Descrição:** Gradiente da função de custo em relação a θ **Fórmula / Ação:** $g_t = \nabla_\theta C(\theta_{t-1})$
Etapa 3.2	**Descrição:** Acumulador com os quadrados dos gradientes **Fórmula / Ação:** $G_t = G_{t-1} + g_t \odot g_t$
Etapa 3.3	**Descrição:** Atualizar os parâmetros com taxa adaptativa **Fórmula / Ação:** $\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{G_t+\epsilon}} \odot g_t$

O vetor θ representa os parâmetros do modelo, ou seja, os pesos e vieses que são otimizados durante o treinamento. A taxa de aprendizado, denotada por α, controla o tamanho dos passos dados na direção oposta ao gradiente. Sendo θ_0 os parâmetros (Pesos e Vieses) iniciais da rede, G_t é o acumulador das componentes dos gradientes ao quadrado, g_t o gradiente para função de custo em relação a cada parâmetro θ_0, t é o índice do minilote, θ_t o conjunto completo de parâmetros da rede (pesos sinápticos e vieses) depois da iteração no minilote t, ϵ uma constante pequena para evitar divisão por 0. Vale ressaltar que a operação $\odot$ representa uma multiplicação componente por componente. A função de custo C é definida como:

$$C = \sum_{i=1}^{m} (y_i - Y_i)^2 \quad (3)$$

A função calcula o somatório dos erros quadráticos entre a saída prevista pela rede y_i e a saída desejada (rótulo) Y_i para cada amostra i, na qual i varia de 1 até m, que é o número total de neurônios na camada de saída da rede.

2.6 Algoritmo ADAM

Proposto por [5], é um otimizador baseado em descida de gradiente estocástica que combina momentos de primeira e segunda ordem para calcular taxas de aprendizagem adaptativas por parâmetro. Com baixa demanda de memória e fácil implementação, é amplamente utilizado em problemas com grandes volumes de dados e parâmetros. No Quadro 1, apresentamos os passos requeridos pelo algoritmo ADAM.

Quadro 2: Algoritmo ADAM.

Etapa 1	**Descrição:** Inicializar parâmetros **Fórmula / Ação:** $\theta_0 \leftarrow$ Valores iniciais
Etapa 2	**Descrição:** Inicializar momento **Fórmula / Ação:** $m_0 \leftarrow 0$
Etapa 3	**Descrição:** Inicializar variância **Fórmula / Ação:** $v_0 \leftarrow 0$
Etapa 4	**Descrição:** Inicializar contador de tempo **Fórmula / Ação:** $t \leftarrow 0$
Etapa 5	**Descrição:** Para cada iteração t = 1 até T: **Fórmula / Ação:** —
Etapa 5.1	**Descrição:** Gradiente da função de custo em relação a θ **Fórmula / Ação:** $g_t = \nabla_\theta C(\theta_{t-1})$
Etapa 5.2	**Descrição:** Atualizar momento (1ª média móvel) **Fórmula / Ação:** $m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$

Etapa 5.3	**Descrição:** Atualizar variância (2ª média móvel) **Fórmula / Ação:** $v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$
Etapa 5.4	**Descrição:** Corrigir o viés do momento **Fórmula / Ação:** $m'_t = \frac{m_t}{(1-\beta_1)^t}$
Etapa 5.5	**Descrição:** Corrigir o viés da variância **Fórmula / Ação:** $v'_t = \frac{v_t}{(1-\beta_1)^t}$
Etapa 5.6	**Descrição:** Atualizar os parâmetros **Fórmula / Ação:** $\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v'_t + \varepsilon}} \cdot m'_t$

Os coeficientes β_1 e β_2 são fatores de decaimento exponencial utilizados para calcular médias móveis dos gradientes e de seus quadrados, respectivamente. Para garantir estabilidade numérica e evitar divisões por zero, o algoritmo utiliza um pequeno valor ε, geralmente igual a 10^{-8}.

A cada iteração t, é calculado o gradiente g_t, correspondente à derivada da função de custo em relação aos parâmetros. Em seguida, são atualizados dois vetores auxiliares: o vetor de momento m_t , que armazena a média móvel dos gradientes, e o vetor de variância v_t, que armazena a média móvel dos quadrados dos gradientes. Como tanto m_t quanto v_t são inicialmente nulos, é realizada uma correção de viés para evitar distorções nos primeiros passos. Isso gera os vetores corrigidos m'_t e v'_t , que são então utilizados na atualização dos parâmetros. A função de custo C é a mesma equação 3 acima.

2.7 Algoritmo *Weighted Gradient /* WEGRAD (Método Proposto)

Neste trabalho, propomos um novo método de treinamento de redes neurais, baseado no uso de minilotes e no ajuste adaptativo dos gradientes associados a amostras incorretamente classificadas. O objetivo é tanto acelerar a convergência, como fugir de mínimos locais, reforçando o aprendizado sobre exemplos nos quais a rede apresenta maior dificuldade.

O procedimento de treinamento ocorre da seguinte forma: inicialmente, os parâmetros da rede (pesos e vieses) são sorteados aleatoriamente em um intervalo pequeno centrado em zero. Em cada época, as amostras de treinamento são embaralhadas e divididas em minilotes de tamanho fixo. Dessa forma, para cada minilote:

- Cada amostra é processada pela rede (passagem direta) e seus gradientes são calculados via retropropagação.

- O gradiente estocástico é calculado somando-se a influência de cada ponto de amostra do minilote, mas o peso de cada elemento do minilote no gradiente depende se a rede conseguiu classificar a amostra com sucesso ou não. Se a predição da rede para a amostra for incorreta, os gradientes correspondentes são multiplicados por um fator de penalização p com $p \geq 1$. Se a predição for correta, os gradientes são acumulados normalmente. Isso cria um gradiente ponderado, que designaremos por u_t para diferenciá-lo do gradiente estocástico normal g_t.

- Ao final do minilote, os parâmetros da rede são atualizados utilizando o gradiente ponderado calculado no minilote.

- Tanto a taxa de aprendizagem quanto o peso de penalização são atualizados de acordo com o número total de épocas que se deseja treinar a rede. O índice da época será designado por τ. De forma que representamos $\alpha(\tau)$ e $p(\tau)$ como a taxa de aprendizagem e o fator de penalização na $\tau - ésima$ época respectivamente.

A atualização dos parâmetros θ ao final de cada minilote é feita da seguinte forma:

$$\theta_t = \theta_{t-1} - \alpha(\tau) \cdot u_t \quad (4)$$

onde $\alpha(\tau)$ é a taxa de aprendizado e u_t o gradiente ponderado calculado para o minilote t.

A título de ilustração, se u_t representa a variável de acumulação do gradiente ponderado para o minilote t e Δu a contribuição de um elemento do minilote para o gradiente. Neste caso, a atualização é feita da seguinte forma:

- se o dado está sendo classificado de forma incorreta:

$$u_t \leftarrow u_t + p(\tau) \cdot \Delta u \quad (5)$$

- se o dado está sendo classificado da forma correta:

$$u_t \leftarrow u_t + \Delta u \quad (6)$$

Outra forma de escrever o cálculo do gradiente ponderado é:

$$u_t = \sum_{i=1}^{n} w(i) \cdot \Delta u_i \quad (7)$$

na qual n é o total de elementos em um minilote, t é o índice do minilote, e $w(i) = 1$ se o elemento i está sendo classificado de modo correto e $w(i) = p(\tau)$ se está classificado de forma errada.

Além disso, o método adota duas estratégias dinâmicas de ajuste da taxa de aprendizado e o fator de penalização ao longo do treinamento. A taxa de aprendizado, decresce linearmente conforme a época:

$$\alpha(\tau) = \alpha_0 - \frac{\alpha_0 * (\tau - 1)}{N}, \quad (8)$$

onde α_0 é a taxa de aprendizado inicial e N é o número total de épocas em que se quer realizar o treinamento.

O fator de penalização cresce linearmente ao longo das épocas:

$$p(\tau) = p_0 + k * \frac{p_0 * (\tau - 1)}{N}, \quad (9)$$

onde p_0 é o fator de penalização inicial e k é um parâmetro que mede o peso da penalização limitando o máximo peso da penalização. Em todas nossas simulações, utilizamos $k = 100$, sendo que o parâmetro não foi otimizado nem sua variação estudada ainda.

3. METODOLOGIA

Neste estudo, foram utilizados quatro conjuntos de dados com diferentes características e níveis de complexidade, com o objetivo de comparar o desempenho dos algoritmos de treinamento SGD, ADAM, ADAGRAD e o método proposto, WEGRAD.

Os dados selecionados representam distintos domínios e desafios, desde problemas simples de decisão até tarefas complexas de classificação com alta dimensionalidade e dados visuais.

3.1 Base de dados Jogo da Velha

O primeiro conjunto corresponde ao jogo da velha, um problema clássico e de baixa complexidade, amplamente utilizado em estudos de inteligência artificial. A base de dados, construída especificamente para este trabalho, contém vetores representando diferentes estados do tabuleiro e suas respectivas jogadas ideais, determinadas pelo algoritmo *Minimax*.

Esta base, então, é composta por um total de 6617 posições possíveis (através de jogadas legais) para o tabuleiro do jogo e, para cada posição, o vetor de saída esperado (Y_i) representa em qual quadrado os jogadores devem jogar. No caso, os quadrados são rotulados de 1 a 9, de forma que a camada de saída tem 9 neurônios. Para o treinamento, foi utilizada uma rede neural com quatro camadas, composta por 18 neurônios na camada de entrada (representando o estado do tabuleiro), 130 e 100 neurônios nas duas camadas ocultas, respectivamente, e 9 neurônios na camada de saída (representando a jogada a ser efetuada).

3.2 Base de dados MNIST

O segundo conjunto é *o* MNIST *(Modified National Institute of Standards and Tchnology)*, que consiste em 70.000 imagens de dígitos manuscritos (60.000 para treinamento e 10.000 para teste), cada uma com 28x28 *pixels* em tons de cinza. Esta base de dados funciona da seguinte maneira: cada imagem representa um número de 0 a 9. As imagens possuem dimensão de 28x28 *pixels* em escala de cinza, totalizando 784 características (*pixels*) por amostra, e cada *pixel* possui um valor de intensidade que varia de 0 (preto) a 1 (branco).

Durante o treinamento, cada imagem é associada a um rótulo que indica o dígito correspondente, permitindo que os algoritmos aprendam a mapear os padrões visuais para suas respectivas classes. A base foi obtida a partir de um código disponibilizado no MATLAB *Central File Exchange*, desenvolvido por [6]. Para o treinamento, foi utilizada uma rede neural com quatro camadas, composta por 784 neurônios na camada de entrada, 80 e 60 neurônios nas duas camadas ocultas, respectivamente, e 10 neurônios na camada de saída (representando cada digito de 0 a 9).

3.3 Base de dados CIFAR-10

O terceiro conjunto é o CIFAR-10, composto por 60.000 imagens coloridas (32x32 *pixels*, RGB) distribuídas entre 10 classes: avião, automóvel, pássaro, gato, cervo, cachorro, sapo, cavalo, navio e caminhão. As imagens apresentam variações em cor, posição, fundo e iluminação, o que torna o problema de classificação mais desafiador.

Essa base exige que os algoritmos capturem características discriminantes em um contexto visual complexo, com classes semelhantes entre si. Para o treinamento, foi utilizada uma rede neural com quatro camadas, composta por 3072 neurônios na camada de entrada, 80 e 60 neurônios nas duas camadas ocultas, respectivamente, e 10 neurônios na camada de saída. A base de dados foi obtida no site oficial do *Canadian Institute for Advanced Research* (CIFAR-10), mantido pela Universidade de Toronto, onde o conjunto é disponibilizado em diferentes formatos.

3.4 Base de dados *Covertype*

O quarto conjunto utilizado é o *Covertype*, composto por mais de 580.000 amostras tabulares, cada uma com 54 atributos, sendo 10 contínuos relacionados a características geográficas (como elevação, inclinação, distâncias a rios, estradas e áreas de incêndio) e 44 binários que indicam tipos de área selvagem e tipos de solo. O objetivo é prever o tipo de cobertura florestal entre sete classes: *Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir* e *Krummholz.*

Trata-se de um problema que envolve dados de alta dimensionalidade, com variáveis heterogêneas e escalas variadas. Para o treinamento, foi utilizada uma rede neural com quatro camadas, composta por 54 neurônios na camada de entrada, 80 e 60 neurônios nas duas camadas ocultas e 7 neurônios na camada de saída, correspondentes às sete classes. A base de dados foi obtida no repositório público UCI *Machine Learning Repository*, mantido pela Universidade da Califórnia, Irvine, onde o conjunto é disponibilizado em formato texto, contendo registros tabulares prontos para aplicações em aprendizado de máquina.

Para uma avaliação comparativa equilibrada entre os algoritmos de treinamento, todas as redes neurais foram treinadas por 100 épocas em cada conjunto de dados. Essa abordagem permite observar tanto o comportamento inicial dos algoritmos quanto sua convergência em um horizonte de treinamento prolongado. Para os algoritmos baseados em momentos adaptativos (ADAM e ADAGRAD), foram fixados os seguintes hiperparâmetros, conforme recomendações da literatura e práticas estabelecidas:

- ε (epsilon) = 1×10^{-8} (termo de regularização para estabilidade numérica)

- β_1 = 0,9 (fator de decaimento para o momento de primeira ordem)

- β_2 = 0,999 (fator de decaimento para o momento de segunda ordem)

A taxa de aprendizagem (conforme ilustrado no Quadro 3) foi sistematicamente variada em um intervalo amplo para avaliar a sensibilidade dos algoritmos a este parâmetro. Foram testados sete valores distintos:

Quadro 3: Taxas de Aprendizagem utilizadas.

Taxas de Aprendizagem (α)						
0,1	0,05	0,01	0,005	0,001	0,0005	0,0001

Esta abordagem permite identificar tanto a faixa ótima de taxas de aprendizagem para cada algoritmo-conjunto de dados quanto sua robustez a ajustes deste parâmetro.

4. ANÁLISE E DISCUSSÃO DOS RESULTADOS

Nesta seção, são apresentados e discutidos os resultados obtidos a partir da comparação entre quatro algoritmos de otimização: SGD (*Stochastic Gradient*), ADAGRAD, ADAM e WEGRAD. As métricas de avaliação incluem acurácia (em termos de jogadas erradas e percentual de erro) e desempenho geral (média e desvio padrão).

4.1 Resultados para a base de dados Jogo da Velha

No Quadro 4, apresentamos os resultados do número de erros para a base Jogo da Velha a partir da comparação entre quatro algoritmos de otimização: SGD (*Stochastic Gradient*), ADAGRAD, ADAM e WEGRAD.

Quadro 4: Número de erros para base Jogo da Velha.

ACURÁCIA (NÚMERO DE ERROS)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	5965.0	461.0	4991.0	555.0
	0.05	5965.0	984.0	2.0	812.0
	0.01	104.0	3442.0	449.0	3632.0
	0.005	73.0	3661.0	1209.0	5365.0
	0.001	777.0	5451.0	4073.0	5505.0
	0.0005	1454.0	5451.0	4754.0	5480.0
	0.0001	3588.0	5451.0	5493.0	5470.0
Média		2560.9	3557.3	2995.9	3831.3
Desvio Padrão		2609.7	2121.0	2349.1	2252.7
Melhor		73.0	461.0	2.0	555.0

O método WEGRAD destacou-se como o mais eficiente para o problema em questão, alcançando o menor número de jogadas erradas (apenas 2) com uma taxa de aprendizado de 0,05. Esse resultado sugere que o WEGRAD é particularmente adequado para problemas de menor escala, como o Jogo da Velha, devido à sua capacidade de realizar ajustes precisos com taxas de aprendizado intermediárias.

O ADAM também apresentou bons resultados, especialmente em taxas menores (0,01 e 0,005), demonstrando robustez em diferentes configurações. Por outro lado, ADAGRAD e SGD tiveram desempenho inferior, registrando um alto número de erros em taxas de aprendizado reduzidas. Esse comportamento pode indicar instabilidade ou convergência lenta, limitando sua eficácia nesse contexto.

Em resumo, enquanto o WEGRAD mostrou-se a melhor escolha para otimização neste cenário, o ADAM surge como uma alternativa viável, especialmente em configurações mais conservadoras. Já ADAGRAD e SGD pode demandar ajustes adicionais para um melhor resultado.

No Quadro 5, apresentamos o número de acertos (acurácia) para a base de dados Jogo da Velha, a partir da comparação entre quatro algoritmos de otimização: SGD (*Stochastic Gradient*), ADAGRAD, ADAM e WEGRAD.

Quadro 5: Número de acertos (acurácia) para base de dados Jogo da Velha

ACURÁCIA (% ACERTO)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	9.85%	93.03%	24.57%	91.61%
	0.05	9.85%	85.13%	99.97%	87.73%
	0.01	98.43%	47.98%	93.21%	45.11%
	0.005	98.90%	44.67%	81.73%	18.92%
	0.001	88.26%	17.62%	38.45%	16.81%
	0.0005	78.03%	17.62%	28.15%	17.18%
	0.0001	45.78%	17.62%	16.99%	17.33%
Média		61.30%	46.24%	54.72%	42.10%
Desvio Padrão		39.44%	32.05%	35.50%	34.04%
Melhor		98.90%	93.03%	99.97%	91.61%

O WEGRAD demonstrou superioridade no problema do Jogo da Velha, alcançando 99,97% de acerto com uma taxa de aprendizado de 0,05. Esse resultado sugere que o método proposto é altamente eficiente em problemas discretos e de pequena escala, onde a adaptabilidade do WEGRAD a diferentes taxas de aprendizado se mostrou vantajosa.

Em contraste, o ADAM apresentou o pior desempenho (9,9% de acerto para taxa de 0,1), indicando alta sensibilidade a taxas de aprendizado elevadas. No entanto, seu desempenho melhorou significativamente em taxas mais baixas (97,3% para 0,005), o que sugere que o ADAM requer um ajuste fino para ser eficaz.

Os métodos ADAGRAD e SGD tiveram desempenho intermediário, com médias de acurácia em torno de 85%. O SGD mostrou maior consistência, enquanto o ADAGRAD apresentou maior variabilidade dependendo da taxa de aprendizado.

4.2 Resultados para a base de dados CIFAR-10

No Quadro 6, apresentamos os resultados do número de erros para a base de dados CIFAR-10, a partir da comparação entre quatro algoritmos de otimização: SGD (*Stochastic Gradient*), ADAGRAD, ADAM e WEGRAD.

Quadro 6: Número de erros para base de dados CIFAR-10.

ACURÁCIA (NÚMERO DE ERROS)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	45000	24458	45000	28520
	0.05	45024	20414	45000	26834
	0.01	44975	21913	29864	21890
	0.005	45179	27518	27820	22106

	0.001	26059	40247	17260	30244
	0.0005	22224	41571	18813	39411
	0.0001	24921	45074	25592	44822
Média		36197.43	31599.29	29907.00	30546.71
Desvio Padrão		11092.87	10347.11	11266.45	8626.77
Melhor		22224	20414	17260	21890

O WEGRAD apresentou o melhor desempenho pois obteve o menor número de erros (17.260) com taxa de 0,001, seguido por ADAGRAD (20414 com taxa 0,05). ADAM e SGD apresentaram desempenho consistente, mas não superaram os outros métodos.

O ADAM, assim como, o WEGRAD, apresentaram os piores resultados em taxas altas, com erros próximos ao máximo (60.000 amostras).

Para conjuntos de dados mais complexos como CIFAR-10, SGD e ADAGRAD foram mais robustos, especialmente em taxas de aprendizado intermediárias. O WEGRAD destacou-se com a melhor média e o melhor caso, sugerindo, ainda que provisoriamente, que o método é capaz de desempenhar bem também em conjuntos de dados maiores, conforme ilustrado no Quadro 7.

Quadro 7: Número de acertos (acurácia) para base de dados CIFAR-10

ACURÁCIA (% ACERTO)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	25.00%	59.24%	25.00%	52.47%
	0.05	24.96%	65.98%	25.00%	55.28%
	0.01	25.04%	63.48%	50.23%	63.52%
	0.005	24.70%	54.14%	53.63%	63.16%
	0.001	56.57%	32.92%	71.23%	49.59%
	0.0005	62.96%	30.72%	68.65%	34.32%
	0.0001	58.47%	24.88%	57.35%	25.30%
Média		39.67%	47.33%	50.16%	49.09%
Desvio Padrão		18.49%	17.25%	18.78%	14.38%
Melhor		62.96%	65.98%	71.23%	63.52%

Para o conjunto CIFAR-10, que representa um problema mais complexo (classificação de imagens coloridas), o WEGRAD obteve a maior acurácia (71,23% para taxa de 0,001), seguido pelo ADAGRAD (65,98% para taxa de 0,05). Esse resultado indica que o método proposto se mostrou competitivo frente aos métodos adaptativos existentes.

O ADAM teve o pior desempenho inicial (25% para taxa de 0,1), mas melhorou em taxas menores (62,9% para 0,0005), reforçando sua dependência de hiperparâmetros cuidadosamente selecionados.

4.3 Resultados para a base de dados MNIST

No Quadro 6, apresentamos os resultados do número de erros para a base de dados MNIST, a partir da comparação entre quatro algoritmos de otimização: SGD (*Stochastic Gradient*), ADAGRAD, ADAM e WEGRAD.

Quadro 8: Número de erros para base de dados MNIST.

ACURÁCIA (NÚMERO DE ERROS)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	54149	249	38090	124
	0.05	54077	283	1597	127
	0.01	1314	1836	4	281
	0.005	299	4627	10	459
	0.001	195	42056	22	2630
	0.0005	216	53258	87	5653
	0.0001	874	53258	2739	53258
Média		15874.86	22223.86	6078.43	8933.14
Desvio Padrão		26124.77	25849.43	14155.55	19651.13
Melhor		195.00	249.00	4.00	124.00

WEGRAD novamente se destacou, com apenas 4 erros em taxa 0.01, seguido por SGD (124 erros em taxa 0,1). ADAM teve bom desempenho com taxas menores (195 erros com taxa de 0,001). O método ADAGRAD teve alta variabilidade, com erros elevados em taxas baixas (53.258 erros).

A relativa robustez do método WEGRAD na base de dados MNIST reforça a ideia de que o método pode ser competitivo. A ilustração de tais resultados está presente no Quadro 9.

Quadro 9: Número de acertos (acurácia) para base de dados MNIST.

ACURÁCIA (% ACERTOS)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	9.74%	97.07%	37.51%	97.66%
	0.05	10.32%	97.12%	95.57%	97.75%
	0.01	96.26%	95.91%	97.97%	97.70%
	0.005	97.27%	91.80%	97.93%	97.42%
	0.001	96.97%	30.24%	97.60%	95.10%
	0.0005	96.89%	11.35%	97.32%	90.83%

	0.0001	96.89%	11.35%	94.80%	11.35%
Média		72.05%	62.12%	88.39%	83.97%
Desvio Padrão		42.37%	42.11%	22.47%	32.12%
Melhor		97.27%	97.12%	97.97%	97.75%

No MNIST, o WEGRAD novamente se destacou, atingindo 97,97% de acerto com taxa de 0,01, o melhor resultado entre todos os algoritmos testados. Esse desempenho reforça a eficácia do método em problemas balanceados e de classificação simples.

O SGD também apresentou excelentes resultados 97,75% para taxa de 0,05, demonstrando robustez em diferentes taxas de aprendizado.

O ADAGRAD, por outro lado, teve um colapso de desempenho em taxas baixas (11,35% para ≤ 0,0005), indicando que seu mecanismo de adaptação pode ser instável em certos regimes de treinamento.

4.4 Resultados para a base de dados COVERTYPE

No Quadro 10, apresentamos os resultados do número de erros para a base de dados COVERTYPE, a partir da comparação entre os quatro algoritmos de otimização: SGD (*Stochastic Gradient*), ADAGRAD, ADAM e WEGRAD.

Quadro 10: Número de erros para base de dados COVERTYPE.

ACURÁCIA (NÚMERO DE ERROS)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	297701	90991	578253	68426
	0.05	315675	104647	563933	65313
	0.01	110858	150350	77853	61464
	0.005	64862	157628	80541	66218
	0.001	47284	167715	114330	112019
	0.0005	52442	194868	132127	127732
	0.0001	93261	297706	175289	160637
Média		140297.51	166272.14	246046.57	94544.14
Desvio Padrão		115964.63	68176.01	224509.70	39175.57
Melhor		47284	90991	77853	61464

ADAM obteve o menor número de erros absoluto (47284 erros com taxa de 0,001), seguido por SGD (61.464 erros com taxa de 0,01). WEGRAD teve desempenho variável, com resultados ruins em taxas altas, mas competitivos em taxas intermediárias (77.853 com 0,01). O SGD mostrou-se mais estável para conjuntos de dados grandes e esparsos como COVERTYPE, enquanto ADAM E WEGRAD podem requerer ajustes mais cuidadosos das taxas. Veja o Quadro 11.

Quadro 11: Número de acertos (acurácia) para base de dados COVERTYPE

ACURÁCIA (% ACERTOS)	α	ADAM	ADAGRAD	WEGRAD	SGD
	0.1	48.76%	84.34%	0.47%	88.22%
	0.05	45.67%	81.99%	2.94%	88.76%
	0.01	80.92%	74.12%	86.60%	89.42%
	0.005	88.84%	72.87%	86.14%	88.60%
	0.001	91.86%	71.13%	80.32%	80.72%
	0.0005	90.97%	66.46%	77.26%	78.02%
	0.0001	83.95%	48.76%	69.83%	72.35%
Média		83.95%	72.87%	77.26%	88.22%
Desvio Padrão		19.96%	11.73%	38.64%	6.74%
Melhor		91.86%	94.34%	86.60%	89.42%

Para o conjunto COVERTYPE, que possui um desbalanceamento significativo entre classes, o ADAM obteve o melhor desempenho (91,86% para taxa = 0,001), seguido pelo SGD (89,42% para taxa de 0,01).

O ADAM teve baixa acurácia em taxas altas (48,8% para 0,1), mas melhorou em taxas menores (83,9% para 0,0001), confirmando sua sensibilidade à mudança de parâmetros.

O WEGRAD apresentou instabilidade, mas apresentou média próxima aos demais algoritmos.

4.5 Comentário geral dos resultados

Os resultados experimentais revelaram diferenças significativas no desempenho dos quatro algoritmos avaliados. O WEGRAD, método proposto neste estudo, destacou-se como o mais robusto, superando os demais em três dos quatro conjuntos de dados testados. Seu desempenho ótimo foi observado em taxas de aprendizado intermediárias (0,01 a 0,001), nas quais demonstrou notável capacidade de equilibrar velocidade de convergência e precisão. No Jogo da Velha, por exemplo, alcançou 99,97% de acerto com taxa de 0,05, enquanto no MNIST registrou apenas 4 erros com taxa 0,01, comprovando sua eficácia especialmente em problemas de média e pequena escala.

O SGD apresentou desempenho confiável em conjuntos de dados maiores e mais complexos, como CIFAR-10 e COVERTYPE, nas quais sua simplicidade algorítmica mostrou-se vantajosa. No entanto, seu comportamento foi mais instável em outros contextos, exigindo ajuste fino da taxa de aprendizado para atingir desempenho ideal. Na base de dados COVERTYPE, em particular, manteve erro abaixo de 16% em todas as taxas testadas, demonstrando boa estabilidade.

O ADAM, por sua vez, mostrou consistência em diversas configurações, especialmente com taxas baixas (0,001 a 0,0001), embora raramente tenha alcançado os melhores resultados absolutos. Sua robustez o torna uma opção segura para aplicações gerais, mas nossa análise indica que existem alternativas mais eficientes para problemas específicos.

O ADAGRAD apresentou os resultados mais limitados, com desempenho satisfatório apenas em taxas altas (0,1) e degradação acentuada em valores menores. Essa limitação parece estar relacionada ao seu mecanismo de adaptação de taxa, que rapidamente diminui o passo de aprendizado, comprometendo sua utilidade prática na maioria dos cenários.

5. CONCLUSÃO E CONSIDERAÇÕES FINAIS

Os resultados obtidos permitem concluir que o algoritmo WEGRAD apresentou desempenho competitivo e, em alguns casos, superior aos métodos tradicionais, especialmente em problemas de média e pequena escala. No Jogo da Velha e no MNIST, o WEGRAD não apenas superou os demais em termos de acurácia, como também apresentou menor número de erros (erro absoluto), demonstrando robustez na tarefa de classificação e uma excelente capacidade de generalização quando configurado com taxas de aprendizado intermediárias.

No entanto, para bases de dados grandes, como CIFAR-10 e COVERTYPE, o desempenho do WEGRAD mostrou-se mais sensível à mudança das taxas de aprendizado. Apesar disso, o método foi capaz de alcançar resultados competitivos e, em alguns casos, superiores aos demais métodos como no caso da base de dados CIFAR-10, sugerindo que a estratégia de ponderação dos gradientes pode ser promissora também em cenários mais desafiadores, desde que acompanhada de um ajuste criterioso dos parâmetros.

Diante dos resultados apresentados, o algoritmo WEGRAD constitui uma contribuição relevante para o campo de otimização de redes neurais, oferecendo uma alternativa viável, particularmente eficaz em problemas nos quais o balanceamento do aprendizado sobre amostras difíceis é determinante para o sucesso do treinamento. Ressalta-se, entretanto, que o método carece de investigações adicionais, especialmente no que se refere à otimização dos parâmetros e à análise de sua escalabilidade em arquiteturas mais profundas e em bases massivas.

O método apresentado foi comparado com alguns algoritmos já bem estabelecidos na literatura. Os resultados mostram que, sob as medidas de desempenho relacionadas com a acurácia (número total de erros, percentual de acerto) o método é promissor, mas estudos com um número maior de épocas e mais bases de dados são necessários para afirmar com maior certeza.

Durante a implementação do método, diversas alternativas de aperfeiçoamento e melhoramento foram cogitadas, uma vez que o método é relativamente simples e outras estratégias podem ser combinadas com a ideia de utilizar um gradiente ponderado. O uso de momentos e de um ajuste dinâmico do fator de penalidade em termos da acurácia da rede em cada época são as ideias mais promissoras para futuros trabalhos.

Uma análise mais detalhada da comparação da ordem de complexidade, número de operações e tempo de execução com os outros métodos também são fatores importantes a serem abordados em trabalhos futuros.

6. REFERÊNCIAS

[1] BLACKARD, J. A.; DEAN, D. J.; ANDERSON, R. S. Covertype Data Set. UCI Machine Learning Repository, 1998. Disponível em: https://archive.ics.uci.edu/ml/datasets/covertype. Acesso em: 01 janeiro 2025.

[2] DEAN, Jeffrey et al. Large scale distributed deep networks. Advances in Neural Information Processing Systems, [S. l.], v. 25, p. 1223-1231, out. 2012.

[3] HAYKIN, Simon. Redes neurais: princípios e prática. In: HAYKIN, Simon. Redes neurais: princípios e prática. 2. ed. Porto Alegre: Bookman, 2001.

[4] KRIZHEVSKY, A. Learning Multiple Layers of Features from Tiny Images. Technical Report, University of Toronto, 2009. Disponível em: https://www.cs.toronto.edu/~kriz/cifar.html. Acesso em: 01 janeiro 2025.

[5] KINGMA, Diederik P.; BA, Jimmy Lei. Adam: a method for stochastic optimization. International Conference on Learning Representations, San Diego, v. 1, p. 1-15, 2015.

[6] LANGELAAR, Johannes. MNIST neural network training and testing. MATLAB Central File Exchange, 2025. Disponível em: https://www.mathworks.com/matlabcentral/fileexchange/73010-mnist-neural-network-training-and-testing. Acesso em: 01 janeiro 2025.

[7] OJHA, Varun; NICOSIA, Giuseppe. Backpropagation neural tree. Neural Networks, [S. l.], v. 149, p. 66–83, 2022.

SILVA, SPATTI, FLAUZINO. Redes neurais artificiais para engenharia e ciências aplicadas: Curso prático. 1. ed. São Paulo: Artliber, 2010