ISBN: 978-628-95207-8-1. ISSN: 2414-6390. Digital Object Identifier: 10.18687/LACCEI2024.1.1.1160

Newton's Method: A Nonlinear Optimization Study

Cesar Caytuiro Tapia, Mg.[1], Cristian Roberto Pastro, Mg.[2]
[1]Universidad Privada del Norte, Peru, cesar.caytuiro@upn.pe
[2] *Universidade Tecnológica Federal do Parana, Brasil, cristian-pastro@hotmail.com*

Abstract–Neural Networks are currently the subject of extensive study and reserch, serving as tools for many works in the academic area. For a neural network to operate properly, it is necessary to carry out a process called training. Training a neural network consists of minimizing the network error. To minimize such error, a first order method can be used. First order methods use a gradient vector of the error function (vector of first derivatives). One of the best-known first-order methods is gradient descent. This method, although functional, can, in many cases, take time to achieve its objective. In such cases it is common to resort to second order methods, such as Newton's Method, which uses a Hessian matrix (matrix of second derivatives). This work presents a comparison of the convergence of the two methods, exploring both aspects of effectiveness and speed of optimization.

Keywords—Neural Networks, Training, Gradient descent, Newton.

Método de Newton: Um Estudo de Otimização Não Linear

Cesar Caytuiro Tapia, Mg.[1], Cristian Roberto Pastro, Mg.[2]
[1]Universidad Privada del Norte, Peru, cesar.caytuiro@upn.pe
[2] Universidade Tecnológica Federal do Parana, Brasil, cristian-pastro@hotmail.com

Resumo–Redes Neurais são atualmente, objeto de muito estudo e pesquisa, servindo de ferramentas para muitos trabalhos na área acadêmica. Para uma rede neural operar de forma adequada, é necessário realizar um processo chamado treinamento. O treinamento de uma rede neural consiste em minimizar o erro da rede. Para minimizar tal erro, pode-se utilizar um método de primeira ordem. Métodos de primeira ordem utilizam um vetor gradiente da função de erro (vetor de primeiras derivadas). Um dos métodos de primeira ordem mais conhecidos é o gradiente descendente. Este método, embora funcional, pode em muitos casos, demorar para atingir seu objetivo. Em tais casos é comum se recorrer a métodos de segunda ordem, como o Método de Newton, que utiliza uma matriz Hessiana (matriz de derivadas segundas). Esse trabalho apresenta uma comparação sobre a convergência dos dois métodos explorando ambos os aspectos de efetividade e velocidade da otimização.

Keywords—Redes Neuronais, Treinamento, Gradiente descendente, Newton.

I. INTRODUÇÃO

Nos últimos anos, o estudo de métodos e de aplicações da Inteligência Artificial experimentou um grande crescimento, sendo utilizado para aprendizado de jogos [1], chatboots [2], verificação de exatidão diagnóstica cial aprendizado deda recuperação de imagem dermatológicas [3], detecção de faltas em linhas de energia elétrica [4], sistemas de aviso de colisão [5], atendimento inteligente ao cliente, diagnóstico médico inteligente, professores inteligentes, logística inteligente, chatterbot, sistemas financeiros inteligentes, e dentre uma variedade de outras aplicações [6].

No cenário contemporâneo, o aprendizado de Redes Neurais emerge como uma ferramenta fundamental para a compreensão e o estudo de métodos de Inteligência Artificial [7], sendo as Redes Neurais uma das técnicas mais utilizadas quando se fala em Inteligência Artificial [8].

Assim sendo, é de suma importância que alunos que desejam se aprofundar no estudo da Inteligência Artificial aprendam as bases dos métodos utilizados nas Redes Neurais, desde os artifícios matemáticos básicos que regem funcionamento de um neurônio artificial, até os métodos algorítmicos empregados na minimização dos erros das Redes Neurais.

O início do estudo de Redes Neurais dá-se pelo modelo matemático de neurônios artificiais proposto por [9]. Este modelo é descrito por um conjunto de n entradas x_j, cada entrada é multiplicada por um determinado peso sináptico ω_j e em seguida os resultados são somados e comparados a um limiar. Foi provado por [10] que a flexibilidade de uma rede neural se dá pela variação dos pesos sinápticos ω_j. Várias metodologias foram propostas ao longo dos anos, tanto do ponto de vista de suas arquiteturas, como também em relação aos otimizadores empregados. Entre eles o Perceptron [11], Adaline [12] e redes perceptron de múltiplas camadas ou Multi Layer Perceptron (MLP), podendo empregar o método de backpropagation na otimização dos pesos [13].

Para uma rede neural funcionar de forma adequada é necessário realizar o procedimento de treinamento da rede neural. Basicamente, um algoritmo de treinamento consiste em minimizar uma função de perda f. Esta função, em geral, é composta por um termo de erro e termos de regularização. O termo de erro avalia o quanto uma rede neural se ajusta ao conjunto de dados e a regularização tenta evitar o *overfitting*. A função de perda f depende do *biases* e do peso sináptico dos neurônios.

A minimização da função de perda f se dá através de métodos matemáticos e numéricos, buscando um conjunto de pesos sinápticos $\omega = \{\omega_1, \omega_2, \omega_3, .., \omega_n\}$ que minimizem o erro. Para este processo, podem ser utilizados métodos de primeira ou segunda ordem. A diferença destes métodos se dá pelo grau da derivada utilizada, onde os métodos de primeira ordem utilizam a primeira derivada da função de erro, enquanto os métodos de segunda ordem utilizam também a segunda derivada desta função. Neste trabalho dois métodos serão abordados, o gradiente descendente e o método de Newton, respectivamente de primeira e segunda ordem.

II. GRADIENTE DESCENDENTE

O método do gradiente descendente busca um mínimo local de uma função através de um esquema iterativo, onde cada passo se toma a direção contrária do gradiente da função a ser otimizada, esta direção corresponde ao declive máximo. De forma mais sucinta, para calcular a direção d_{i+1} ao qual o método deve seguir no passo $i + 1$ para minimizar a função $f(\omega)$, segue a Equação 1.

$$d_i = -\nabla f(\omega_i) = g(\omega_i) \quad (1)$$

O vetor de derivadas (ou vetor gradiente) da função $f(\omega)$ é denotado por $g(\omega)$ e pode ser definido como mostra a Equação 2. Onde ω é um vetor de n parâmetros da função f. Ou seja, $\omega = (x_1, x_2, ..., x_n)$.

$$g(\omega) = \begin{bmatrix} \frac{\delta f(\omega)}{\delta x_1} \\ \frac{\delta f(\omega)}{\delta x_2} \\ ... \\ \frac{\delta f(\omega)}{\delta x_n} \end{bmatrix} \quad (2)$$

Utilizando a ideia de seguir pelo caminho inverso do maior gradiente de f, tem-se que valor de cada iteração i é dado pela Equação 3, onde η é o valor do passo, indicando o quanto o algoritmo deve seguir na direção designada por ∇g_i. O valor de η geralmente é escolhido dinamicamente ao decorrer do algoritmo.

$$\omega_{i+1} = \omega_i - \eta g_i \quad (3)$$

E por fim, em uma rede neural, a Equação 3 fica como mostrado em 4, onde xj é o valor de entrada recebido pela conexão j e e_j é o valor calculado do erro do processador j.

$$\Delta\omega_{j+1} = \Delta\omega_j + \eta e_j x_j \quad (4)$$

O Algoritmo 8 mostra o método do gradiente descendente para uma função f em relação aos valores contidos em um vetor ω e com valores iniciais ω_0.

Algoritmo 1: MÉTODO DO GRADIENTE DESDEDENTE

Entrada: $f(x, y), \omega_0$

Saída: Mínimo ω da função f

1 **início**
2 $g \leftarrow$ Matriz do vetor gradiente de f;
3 $\omega \leftarrow \omega_0$;
4 **enquanto** critério de parada não for satisfeito faca
5 $\omega_{i+1} \leftarrow \omega_i - \eta_i g_i$
6 **fim**
7 **fim**
8 **retorna** ω

III. MÉTODO DE NEWTON

A técnica utilizada pelo método de Newton para minimizar uma função é encontrar as melhores direções de variação dos parâmetros utilizando a segunda derivada da função a ser minimizada. O método de Newton originalmente busca encontrar raízes (ou zeros) de funções, porém pode ser adaptado para encontrar mínimo de funções (ou o zero de sua primeira derivada).

Pode-se aproximar qualquer função $f(x)$ com base em suas derivadas, para isso utiliza-se a série de Taylor centrada em x_0, mostrada na Equação 5, sendo que $f^{(n)}$ representa a enésima derivada de f.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \quad (5)$$

Quanto mais termos da série de Taylor forem utilizados, com maior precisão a função $f(x)$ pode ser aproximada. Utilizando os três primeiros termos da série, temos a Equação 3, onde $\dot{f}$ e $\ddot{f}$ correspondem respectivamente a primeira e a segunda derivada da função f.

$$f(x) \approx f(x_0) + \dot{f}(x_0)(x - x_0) + 0.5\ddot{f}(x_0)(x - x_0)^2 \quad (6)$$

Nota-se que a função f depende de vários parâmetros contidos em um vetor de parâmetros ω, então a série de Taylor dada estas condições fica como mostrado na equação 7, onde ω_0 representa o vetor de parâmetros iniciais, f_0 representa a função nos valores iniciais, g_0 é o vetor de derivadas primeiras aplicadas no ponto inicial e H_0 é a matriz de derivadas segundas aplicadas no ponto inicial.

$$f(\omega) = f_0 + g_0(\omega - \omega_0) + 0.5(\omega - \omega_0)^2 H_0 \quad (7)$$

A matriz de derivadas segundas chama-se matriz Hessiana, sendo a Equação 8.

$$H(\omega) = \begin{bmatrix} \frac{\partial^2 f(\omega)}{\partial x_1^2} & \frac{\partial^2 f(\omega)}{\partial x_1 \partial x_2} & ... & \frac{\partial^2 f(\omega)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\omega)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\omega)}{\partial x_2^2} & ... & \frac{\partial^2 f(\omega)}{\partial x_2 \partial x_n} \\ ... & ... & ... & ... \\ \frac{\partial^2 f(\omega)}{\partial x_n \partial x_1} & \frac{\partial^2 f(\omega)}{\partial x_n \partial x_2} & ... & \frac{\partial^2 f(\omega)}{\partial x_n^2} \end{bmatrix} \quad (8)$$

Derivando a 7 em relação a ω tem-se a Equação 9.

$$\dot{f}(\omega) = g_0 + H_0(\omega - \omega_0) \quad (9)$$

Assumindo que para o ponto mínimo, o gradiente (ou derivada) de f será *zero*, pode-se simplificar a equação 9 para 10.

$$0 = g_0 + H_0(\omega - \omega_0) \quad (10)$$

Isolando ω em 10 e tomando o processo como iterativo, obtém-se a regra de atualização dos pesos mostrada na Equação 11:

$$\omega_{i+1} = \omega_i + H_i^{-1} g_i \quad (11)$$

O método de Newton é baseado em Newton-Raphson, que busca encontrar raízes (ou zeros) de funções e é mostrado na Equação 12.

$$x_{i+1} = x_i - \frac{f(x_i)}{\dot{f}(x_i)} \quad (12)$$

Como encontrar o mínimo de uma função equivale a encontrar o zero de sua primeira derivada, toma-se uma função $\dot{f}_2(x)$ tal que $f(x) = \dot{f}_2(x)$. Então o método de Newton-Raphson para a primeira derivada de uma função é mostrado na Equação 13.

$$x_{i+1} = x_i - \frac{\dot{f}_2(x_n)}{\ddot{f}_2(x_i)} \quad (13)$$

Percebe-se a semelhança entre a equação 11 e a equação 13. A diferença das duas equações é que enquanto a equação 13 trabalha com apenas uma variável, a equação 11 trabalha com um vetor de entradas ω, transformando a primeira derivada em um vetor gradiente e a segunda derivada em uma matriz Hessiana.

O Algoritmo 9 mostra o método de newton para uma função f em relação aos valores contidos em um vetor ω e com valores iniciais ω_0.

Algoritmo 2: MÉTODO DE NEWTON

Entrada: $f(x, y), \omega_0$

Saída: Mínimo ω da função f

início
  $g \leftarrow$ Matriz do vetor gradiente de f;
  $H \leftarrow$ Matriz hessiana de f;
  $\omega \leftarrow \omega_0$;
  enquanto critério de parada não for satisfeito **faca**
    $\omega_{i+1} \leftarrow \omega_i - H_i^{-1} g_i$
  fim
fim
retorna ω

IV. EXPERIMENTOS

Objetivando-se verificar a diferença de métodos de primeira ordem e métodos de segunda ordem, os dois métodos acima explicados, o gradiente descendente (primeira ordem) e o método de Newton (segunda ordem) foram implementadas no software MATLAB®. Após a implementação de ambos os métodos, testaram-se funções $\mathbb{R}^2 \rightarrow \mathbb{R}$, cada uma com duas variáveis independentes (x, y) e uma variável dependente (z), verificando para cada função o tempo de computação e o número de passos levados para realizar a computação/otimização. Também foi obtido um gráfico com este mesmo software para cada método e para cada equação.

Definiu-se o critério de parada como sendo quando módulo do vetor gradiente de cada método for menor que 10^{-8}. As funções utilizadas para o teste foram:

- $f_1(x, y) = 2x - y + 2x^2 + 2xy + y^2$; ponto inicial $(x, y) = (1, -5)$;
- $f_2(x, y) = -\left(\log\left(\left| \cos\frac{x}{100} \right| \right) e^x + e^{\frac{19y}{3000+1000\cos(x)}} + \ln\left(\left| (\sin(y)\sin(xy)) \right| \right) \right)$

  ponto inicial $(x, y) = (-0.1, 0.01)$;

Para o método do Gradiente Descendente, o valor de η foi obtido derivando-se a equação contida no algoritmo e igualando o resultado a zero, conseguindo o valor de η que melhor minimiza a função em cada iteração.

Para todos os testes foi obtido o gráfico de contorno com cada iteração marcada. Ambos os algoritmos foram testados no software MATLAB® R2018a, em um sistema operativo Windows 7 64-bits. O processamento foi realizado em um computador portátil Dell Inspiron 5458, contando com um processador Intel® Core™ i5-5200U CPU @ 2.20GHz e 8GB de memória RAM.

V. RESULTADOS E DISCUSSÃO

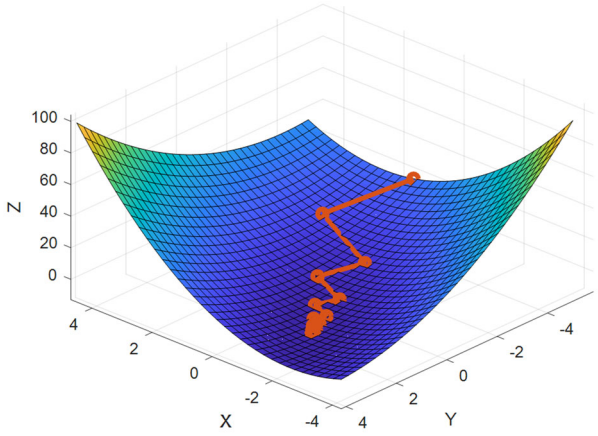
A. Simulação 1

Os resultados obtidos na otimização da função $f_1(x, y) = 2x - y + 2x^2 + 2xy + y^2$ são apresentados na Tabela 1.

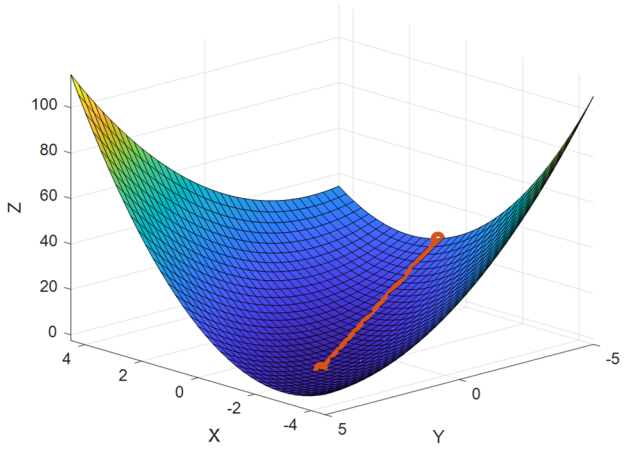
TABELA 1
RESULTADOS DE GRADIENTE DESDENDENTE E NEWTON

	Gradiente descendente	Newton
Resultado em x	-1.49999999436125	-1.5
Resultado em y	1.99999999262625	2
Número de iterações	64	2
Tempo de processamento (s)	34.3	1.0

Para esta mesma equação, os resultados obtidos após simulação podem ser observados no gráfico da Figuras 1(a) e 1(b) para os métodos do Gradiente Descendente e de Newton, respectivamente.



a) Gradiente descendente



b) Método de Newton

Fig. 1 Resultados obtidos para a primeira equação

Pode-se observar pela Figura 1 e Tabela 1 que o método de Newton para função $f_1(x, y)$ converge já no segundo passo. Isso se deve ao fato de que a equação $f_1(x, y)$ é de segundo grau, implicando que a segunda derivada em qualquer direção seja uma constante. Como o método se baseia em segundas derivadas, isso acaba levando a função à convergência imediata no mínimo local da função.

Já para o método do gradiente descendente necessário realizar o procedimento de descida de colina até chegar a um mínimo local. Isso leva, para a função em questão, muito mais passos se comparado com o método de Newton. Nota-se pela Figura 1 que apenas os primeiros 7 passos alteraram a resposta de forma significativa, enquanto os outros passos apenas melhoraram a precisão da resposta obtida. Mesmo considerando este fato, a diferença dos métodos, para a função f_1 a diferença no número de passos necessários para a convergência desejada é grande.

Não obstante o método de Newton necessitar o cálculo de uma Matriz Hessiana, para este caso o procedimento é agilizado devido ao fato da matriz ser uma constante. Além disso, a implementação utilizada do método de Newton faz o cálculo do menor valor de η possível em cada passo, implicando em um custo computacional e de tempo significativo para o cálculo numérico de derivadas em cada iteração, como observa-se pela Tabela 1, onde o método de Newton mostra-se mais rápido. Além disso, o maior número de iterações necessárias para o Gradiente Descendente contribui para o aumento do tempo de execução.

B. Simulação 2

Os resultados obtidos na otimização da função

$$f_2(x,y) = -\left(\log\left(\left| \cos\frac{x}{100} \right| \right) e^{x} + e^{\frac{19y}{3000+1000\cos(x)}} + \ln\left(\left| (\sin(y)\sin(xy)) \right| \right) \right)$$

são apresentados na Tabela 2.

TABELA 2
RESULTADOS DE GRADIENTE DESDENDENTE E NEWTON

	Gradiente descendente	Newton
Resultado em x	-	-0.99818733783477
Resultado em y	-	1.57492987049464
Número de iterações	-	13
Tempo de processamento (s)	-	10.0

Para esta mesma equação, os resultados obtidos após simulação podem ser observados no gráfico de contorno da Fig. 2 para o método de Newton. Não houve tempo hábil para o processamento do Gradiente descendente, causando instabilidade e graves travamentos no computador utilizado.

Para a simulação 2 observa-se mais nitidamente o fenômeno encontrado na simulação 1. O método do gradiente descendente, mesmo em teoria sendo mais rápido, mostrou-se muito mais ineficiente, se comparado ao método de Newton.

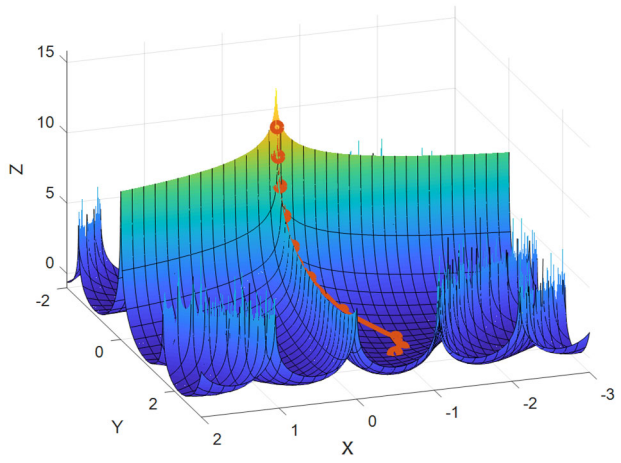


Fig. 2 Resultados obtidos para a segunda equação

Neste caso, a função $f2$ é mais complexa, possuindo termos logaritmos, exponenciais e trigonométricos interagindo entre si. Isso implica em alto custo computacional para a aferição do melhor valor possível de η, já que este depende do cálculo de derivadas em cada iteração. Mesmo o método de Newton necessitando a computação de uma matriz Hessiana, este processo é necessário ser realizado de forma numérica apenas uma vez, já que durante as iterações basta usar o cálculo já feito e aplicar na matriz os valores correntes da iteração atual.

Tais fatores em conjunto, contribuíram de maneira significativa para o melhor desempenho do método de segunda ordem nos testes simulados, gerando a resposta em um tempo significativamente menor.

VI. CONCLUSÃO

Com este trabalho, objetivou-se minimizar funções por meio de dois métodos, sendo um de primeira ordem e um de segunda ordem. Pode-se concluir que em casos gerais, no universo testado, os métodos de segunda ordem convergem mais rapidamente.

Métodos de otimização de segunda ordem têm se mostrado mais eficazes na maioria das situações, porém podem necessitar de uma maior capacidade computacional, já que fazem o uso de uma matriz Hessiana. Já métodos de primeira ordem são mais simples de implementar e necessitam de uma menor capacidade computacional, porém, em geral, convergem de forma mais lenta.

Para funções mais simples, como o caso da função $f1$, o método de Newton mostrou-se um pouco mais eficiente, obtendo a convergência necessária em menos passos. Caso o critério de parada escolhido fosse um número fixo de iterações, é possível que gradiente descendente tivesse um melhor desempenho de tempo. Porém caso tal critério fosse escolhido, a precisão e a qualidade da resposta poderia estar aquém do obtido por meio dos métodos de segunda ordem. Já para funções mais complexas, o método utilizado para calcular o η mostrou-se ineficiente, demandando muito esforço computacional.

Apesar desse trabalho se concentrar apenas em 2 métodos de otimização, uma grande quantidade de outros métodos, de primeira e segunda ordem, é encontrada na literatura, cada qual possuindo uma particularidade. Sem embargo, todos são baseados nos 2 métodos aqui apresentados, sendo o entendimento desses, portanto, essenciais na área de otimização/redes neurais.

REFERENCIAS

[1] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[2] Y. Wu, W. Wu, C. Xing, M. Zhou, and Z. Li, “Sequential Matching Network: A New Architecture for Multi-turn Response Selection in Retrieval-Based Chatbots,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, vol. 1, pp. 496–505. doi: 10.18653/v1/P17-1046.

[3] P. Tschandl, G. Argenziano, M. Razmara, and J. Yap, “Diagnostic accuracy of content-based dermatoscopic image retrieval with deep classification features,” *Br. J. Dermatol.*, vol. 181, no. 1, pp. 155–165, Jul. 2019, doi: 10.1111/bjd.17189.

[4] E. Calderon-Mendoza, P. Schweitzer, and S. Weber, “A double ended AC series arc fault location algorithm for a low-voltage indoor power line using impedance parameters and a neural network,” *Electr. Power Syst. Res.*, vol. 165, pp. 84–93, Dec. 2018, doi: 10.1016/j.epsr.2018.08.008.

[5] S. H. Lee, S. Lee, and M. H. Kim, “Development of a Driving Behavior-Based Collision Warning System Using a Neural Network,” *Int. J. Automot. Technol.*, vol. 19, no. 5, pp. 837–844, Oct. 2018, doi: 10.1007/s12239-018-0080-6.

[6] Y. Lu, “Artificial intelligence: a survey on evolution, models, applications and future trends,” *J. Manag. Anal.*, vol. 6, no. 1, pp. 1–29, Jan. 2019, doi: 10.1080/23270012.2019.1570365.

[7] K. Kumar and G. S. M. Thakur, “Advanced Applications of Neural Networks and Artificial Intelligence: A Review,” *Int. J. Inf. Technol. Comput. Sci.*, vol. 4, no. 6, pp. 57–68, Jun. 2012, doi:

10.5815/ijitcs.2012.06.08.
[8] R. Y. Choi, A. S. Coyner, J. Kalpathy-Cramer, M. F. Chiang, and J. Peter Campbell, “Introduction to machine learning, neural networks, and deep learning,” *Transl. Vis. Sci. Technol.*, vol. 9, no. 2, pp. 1–12, 2020, doi: https://doi.org/10.1167/tvst.9.2.14.
[9] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943, doi: 10.1007/BF02478259.
[10] F. Attneave, M. B., and D. O. Hebb, “The Organization of Behavior; A Neuropsychological Theory,” *Am. J. Psychol.*, vol. 63, no. 4, p. 633, Oct. 1950, doi: 10.2307/1418888.
[11] F. Rosenblatt, “(1958) F. Rosenblatt, ‘The perceptron: a probabilistic model for information storage and organization in the brain,’ Psychological Review 65: 386-408,” in *Neurocomputing, Volume 1*, The MIT Press, 1988, pp. 92–114. doi: 10.7551/mitpress/4943.003.0010.
[12] B. Widrow, “An Adaptive ‘Adaline’ Neuron Using Chemical ‘Memistors,’” *Stanford Electronics Laboratories Technical Report*. 1960.
[13] D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS, “Learning Internal Representations by Error Propagation,” in *Readings in Cognitive Science*, Elsevier, 1988, pp. 399–421. doi: 10.1016/B978-1-4832-1446-7.50035-2.