Victor Rodrigues de Oliveira

Descobrindo Redes Neurais Artificiais - Minicurso sobre os modelos: Perceptron, Adaline e Perceptron Multicamadas

Dissertação de Mestrado apresentada ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, para obtenção do título de Mestre em Matemática.

Instituto Federal de Educação Ciência e Tecnologia de São Paulo

Campus São Paulo

Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT

Orientador: Prof. Dr. Marco Aurélio Granero Santos

São Paulo - SP

2025

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Sumário

1 Adaline

O Adaline é, assim como o Perceptron, um modelo de RNA também muito utilizado na classificação de dados entre duas classes distintas. Seu nome é um acrônimo obtido de *Adaptive Linear Element* e foi apresentado em 1960 por Bernard Widrow[1] e Marcian Hoff[2], meses depois que Rosenblatt apresentou o Perceptron (Widrow; Lehr, 1990). A princípio este modelo foi desenvolvido para ser utilizado no chaveamento de circuitos eletrônicos e estabeleceu um marco no que se refere ao uso de RNA em contextos industriais envolvendo sinais elétricos analógicos (Widrow; Lehr, 1990; Silva; Spatti; Flauzino, 2016).

Segundo Widrow e Lehr (1990), o Perceptron poderia se enquadrar, juntamente com o Adaline, na categoria de combinadores lineares adaptativos[3]. Muitas semelhanças, em questão de estrutura, funcionamento e estratégia de aprendizado, podem ser encontradas entre o funcionamento do Perceptron e do Adaline, como pode ser visto na Figura 1, em comparação com a Figura **??** apresentada no capítulo anterior.

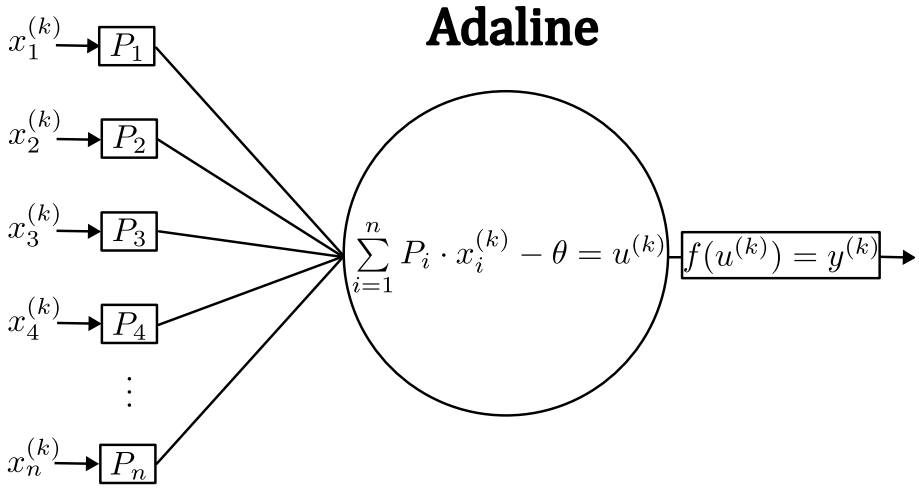

Figura 1 – Representação esquemática do Adaline. Elaborado pelo autor.

Ambas RNA implementam o neurônio de McCulloch-Pitts e portanto funcionam a partir da combinação linear entre os pesos sinápticos e as variáveis das amostras e, em seguida apresentam saídas contínuas que são interpretadas por funções de ativação que mantém seu o propósito de obter uma classificação binária.

Outro ponto em comum é o fato desta rede neural utilizar um treinamento de aprendizado *supervisionado* (Silva; Spatti; Flauzino, 2016).

---

[1] Perfil no site da Universidade de Stanford: `https://profiles.stanford.edu/bernard-widrow`. Acessado em 25 de fevereiro de 2025.

[2] Engenheiro Elétrico nascido em 1937, é reconhecido como um dos criadores do microprocessador.

[3] *Adaptative Linear Combiner*, no original, em inglês.

As principais diferenças do Adaline em relação ao Perceptron são seu algoritmo de aprendizagem, obtido a partir de uma função chamada de Erro Quadrático e denominado de *Regra Delta* ou de *Gradiente Descendente*[4] e o critério de parada para o treinamento da RNA que utiliza comparações entre épocas das saídas da função Erro Quadrático Médio (Silva; Spatti; Flauzino, 2016).

Outra diferença é que o Adaline sempre apresentará valores finais para pesos sinápticos após a conclusão do treinamento, mesmo que os dados de treinamento não sejam linearmente separáveis em sua totalidade. Para isto, basta que o número de variáveis das entradas sejam iguais ao número de pesos sinápticos e que mais da metade destes dados sejam linearmente separáveis (Widrow; Lehr, 1990).

1.1 A regra de aprendizado do Adaline

A regra de aprendizado do Adaline faz uso da função Erro Quadrático (E) para a obtenção do termo utilizado na atualização dos pesos sinápticos.

Para um conjunto de m amostras de treinamento, cada uma delas com n variáveis, o Erro Quadrático é definido pela Equação (1.1):

$$\begin{aligned} E(P) &= \frac{1}{2}\sum_{k=1}^{m}(d^{(k)} - u^{(k)})^2 \\ &= \frac{1}{2}\sum_{k=1}^{m}\left[d^{(k)} - \left(\sum_{i=1}^{n} P_i \cdot x_i^{(k)} - \theta\right)\right]^2 . \end{aligned} \tag{1.1}$$

Na Equação (1.1): P é a matriz dos pesos sinápticos e P_i indica os seus elementos; d é a matriz de saídas desejadas e $d^{(k)}$ é o valor esperado para uma amostra k qualquer; $u^{(k)} = \sum_{i=1}^{n} P_i \cdot x_i^{(k)} - \theta$ é o potencial de ativação produzido pela rede pela amostra k; $x_i^{(k)}$ é a i-ésima variável da amostra k e θ o limiar.

Além disto, a Equação (1.1) é uma equação matricial tal que as dimensões de seus elementos devem preservar as operações, deste modo e considerando m amostras de treinamento com n entradas cada, a dimensão da matriz d é $1 \times m$, da matriz dos pesos sinápticos P é $1 \times n$ e da matriz de entradas x é $n \times m$. Caso o usuário tenha interesse em acrescentar o limiar ao treinamento do neurônio, as variáveis n aumentam em uma unidade, sendo esta "-1", e mais um peso sináptico é acrescentado à matriz P.

A parcela do potencial de ativação $u^{(k)}$ produzido pela rede, representada pelo somatório em função de n envolvendo $P_i \cdot x_i$, pode ser reescrito como um produto escalar,

[4] Originalmente, em inglês, denominada *Least Mean Square (LMS)*.

ou seja,

$$\sum_{i=1}^{n} P_i \cdot x_i^{(k)} = P \cdot x^{(k)},$$

fazendo com que a Equação (1.1) possa ser reescrita sem um dos somatórios como:

$$\begin{aligned} E(P) &= \frac{1}{2}\sum_{k=1}^{m}(d^{(k)} - u^{(k)})^2 \\ &= \frac{1}{2}\sum_{k=1}^{m}\left[d^{(k)} - \left(P \cdot x^{(k)} - \theta\right)\right]^2. \end{aligned} \tag{1.2}$$

A Equação (1.2) é uma função de várias variáveis, $P_1, P_2, ..., P_n$, ou seja, é função da matriz de pesos sinápticos P.

Com o uso de ferramentas de cálculo diferencial pode-se estabelecer, a partir da Equação (1.2), qual é o fator de correção dos pesos sinápticos da rede que será utilizado no algoritmo de aprendizagem. Este processo garante que os pesos sinápticos obtidos no final do treinamento estarão aptos a produzir o menor erro possível para a classificação que o Adaline pode realizar, atribuindo a ele uma função de generalizador de características dos dados envolvidos (Widrow; Lehr, 1990).

O processo de minimização de uma função é realizado com o auxílio de sua derivada em relação a variável em questão. Dessa forma, o algoritmo para atualização da matriz de pesos sinápticos deve conter a derivada da função Erro Quadrático com relação ao respectivo peso sináptico. Como a função E é de várias variáveis, a derivada, ou melhor, o vetor de derivadas em questão é o chamado gradiente de E, escrito como ∇E.

Com isto, a atualização dos pesos sinápticos é dada pela Equação (1.3):

$$P^{(novo)} = P^{(antigo)} + \eta \cdot \nabla E, \tag{1.3}$$

em que η é a taxa de aprendizagem do algoritmo.

O fator de correção ∇E contido no algoritmo de aprendizado apresentado na (1.3) pode ser apresentado explicitamente ao aplicar-se o gradiente do Erro Quadrático na Equação (1.2). Para realizar este desenvolvimento pode-se prosseguir de acordo com Silva, Spatti e Flauzino (2016) em que, sem prejuízo ao caso geral, adota-se uma amostra qualquer k do conjunto de amostras da matriz de entradas x, e também x^T como a matriz

transposta da matriz de entradas, para então obter a Equação (1.4).

$$\begin{aligned}
\nabla E &= \frac{\partial E}{\partial P} \\
&= \frac{\partial E}{\partial u}\frac{\partial u}{\partial P} \\
&= \frac{\partial}{\partial u}\left[\frac{1}{2}\sum_{k=1}^{m}\left(d^{(k)}-u^{(k)}\right)^2\right]\frac{\partial}{\partial P}\left(d^{(k)}-u^{(k)}\right) \\
&= \sum_{k=1}^{m}\left(d^{(k)}-u^{(k)}\right)\frac{\partial}{\partial P}\left[d^{(k)}-\left(P\cdot x^{(k)}-\theta\right)\right] \\
&= \sum_{k=1}^{m}\left(d^{(k)}-u^{(k)}\right)\cdot\left(-x^{(k)}\right)^T. \qquad (1.4)
\end{aligned}$$

É importante ressaltar que P é uma matriz que possui os pesos sinápticos do neurônio como seus elementos, ou seja, as derivadas parciais são em relação a cada um destes pesos sinápticos. Outra observação importante é reparar no uso da regra da cadeira para derivadas, possibilitando o desenvolvimento da primeira para a segunda linha da Equação (1.4).

O gradiente de uma função é um vetor que aponta na direção de maior variação da função. Como o foco deste processo é encontrar o mínimo da função, deve-se inverter o sentido apresentado pela Equação (1.4). Levando-se este fato em consideração e substituindo-o em (1.3) tem-se:

$$P^{(novo)} = P^{(antigo)} + \eta \cdot \sum_{k=1}^{m}(d^{(k)} - u^{(k)}) \cdot (x^{(k)})^T. \qquad (1.5)$$

A Equação (1.4) diz que o treinamento, ou a atualização, na matriz de pesos sinápticos P, irá ocorrer até que a somatória das diferenças entre as saídas esperadas e o potencial de ativação produzidos pelas k amostras seja zero, ou um valor limite aceitável pelo usuário.

O Adaline também é treinado em épocas que se iniciam no primeiro ajuste realizado em sua matriz de pesos sinápticos, assim como o Perceptron. Em tese, como a função $E(P)$ produz saídas com valores reais, a regra de treinamento pode continuar sua busca pelos pesos sinápticos indefinidamente, sempre indo em direção a pesos mais eficientes em relação à época anterior.

Devido a isto, pode-se estabelecer um limite de épocas para a rede, implicando na interrupção do processo de treinamento sem dimensão do quão distante os pesos sinápticos estão dos valores otimizados.

Porém, um critério que utiliza de modo eficaz a maneira como se convergem os pesos sinápticos com o treinamento do Adaline é o de, com o uso dos erros que as saídas

apresentam a cada época e estabelecer uma margem de erro aceitável entre as saídas de diferentes épocas.

1.2 O treinamento do Adaline

O Adaline inicia o treinamento dos pesos sinápticos no momento em que a rede é inicializada. Isto significa que, independentemente dos valores atribuídos aos pesos sinápticos no início do treinamento, o algoritmo de aprendizado será executado. Isto acontece pois o Adaline utiliza um critério específico de parada para o seu treinamento com o uso da função Erro Quadrático Médio (EQM).

Considerando que na matriz x, há um total de m amostras de treinamento, a função EQM apresentada é definida como na Equação (1.6):

$$EQM(P) = \frac{1}{m}\sum_{k=1}^{m}\left(d^{(k)} - u^{(k)}\right)^2. \tag{1.6}$$

O processo de treinamento envolvendo o uso do Gradiente Descendente com o propósito de minimizar saída da função Erro Quadrático E também minimizará, indiretamente, a função Erro Quadrático Médio EQM. Isto é garantido pois ambas as funções EQM e E, são convexas, ou seja, existe um mínimo global para este tipo de função. A Figura 2 ilustra o exemplo com uso uma função real com duas variáveis reais.

O modo que a função EQM é utilizada consiste em comparar suas saídas em épocas sucessivas e, consequentemente, distintas. Se esta comparação apresentar um valor absoluto menor do que o estipulado, então a rede para de atualizar e otimizar os pesos sinápticos, encerrando seu treinamento. Definindo como ε o valor de referência para o erro, o critério de parada pode ser expresso como:

$$\left|EQM^{(anterior)} - EQM^{(atual)}\right| < \varepsilon. \tag{1.7}$$

Em outras palavras, a partir do momento em que a diferença entre os erros quadráticos médios apresentados para o Adaline em épocas sucessivas estiver dentro dos parâmetros considerados aceitáveis, a rede para de atualizar os pesos sinápticos.

Segundo Silva, Spatti e Flauzino (2016), dada a convexidade da função EQM, a curva que ilustra sua variação, em função do número de épocas de treinamento, é sempre decrescente como pode ser observado na Figura 3.

Bernard Widrow, um dos criadores do Adaline, afirma que quando um único Adaline é treinado com o Gradiente Descendente, ele convergirá para uma solução global única, Widrow e Lehr (1990, p.1420).

Em resumo, o treinamento consiste em reduzir o erro a cada época que se passa e verificar se a diferença entre os erros produzidos pela rede se encontra dentro dos parâmetros

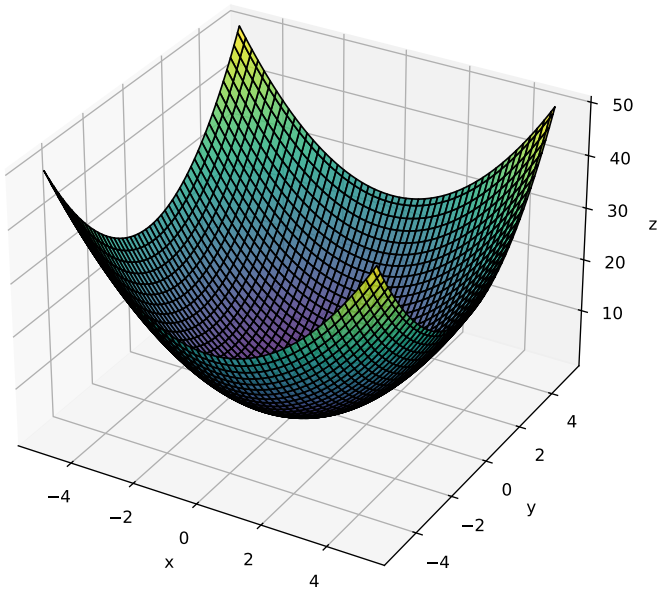

Figura 2 – Superfície de uma função Erro Quadrado Médio usual de um combinador linear. Adaptado de Widrow e Lehr (1990, p.1428).

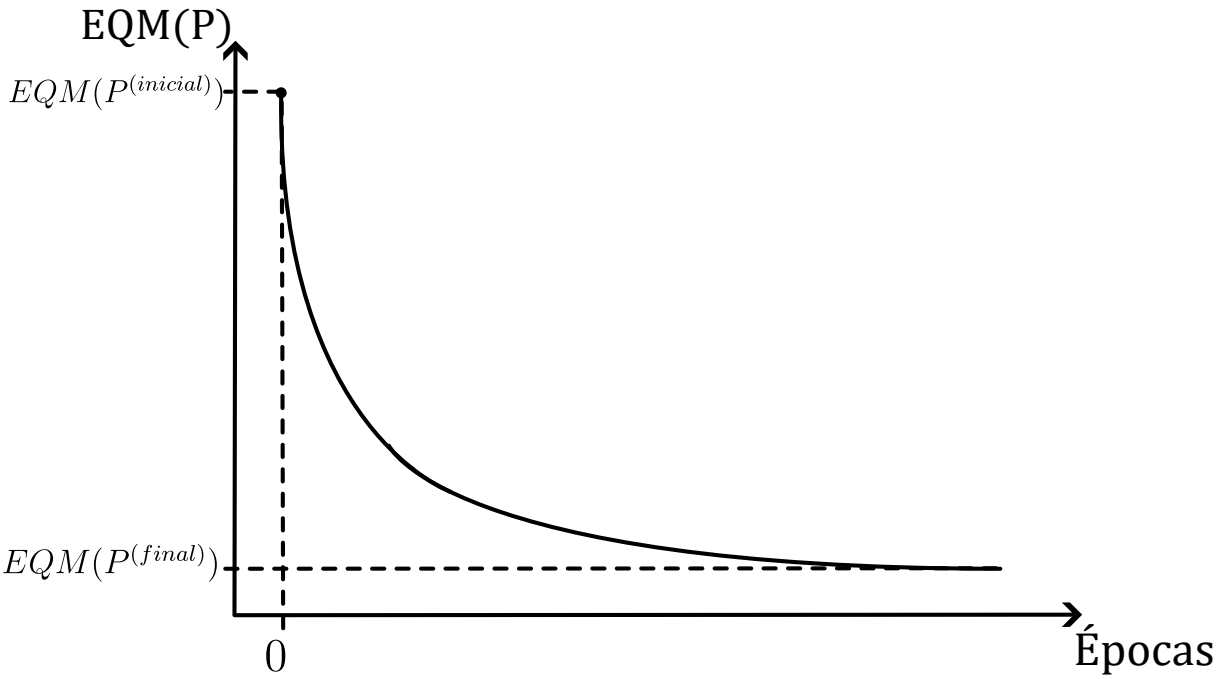

Figura 3 – Convergência da função Erro Quadrático Médio. Adaptado de Silva, Spatti e Flauzino (2016, p.83).

estabelecidos no início do treinamento, ou seja, se entre duas épocas consecutivas a otimização dos pesos for tão pequena quanto o valor estipulado ε, a rede considera que não há mais ajustes a serem feitos e encerra o treinamento. O fluxograma do processo de treinamento desta rede é apresentado na Figura 4.

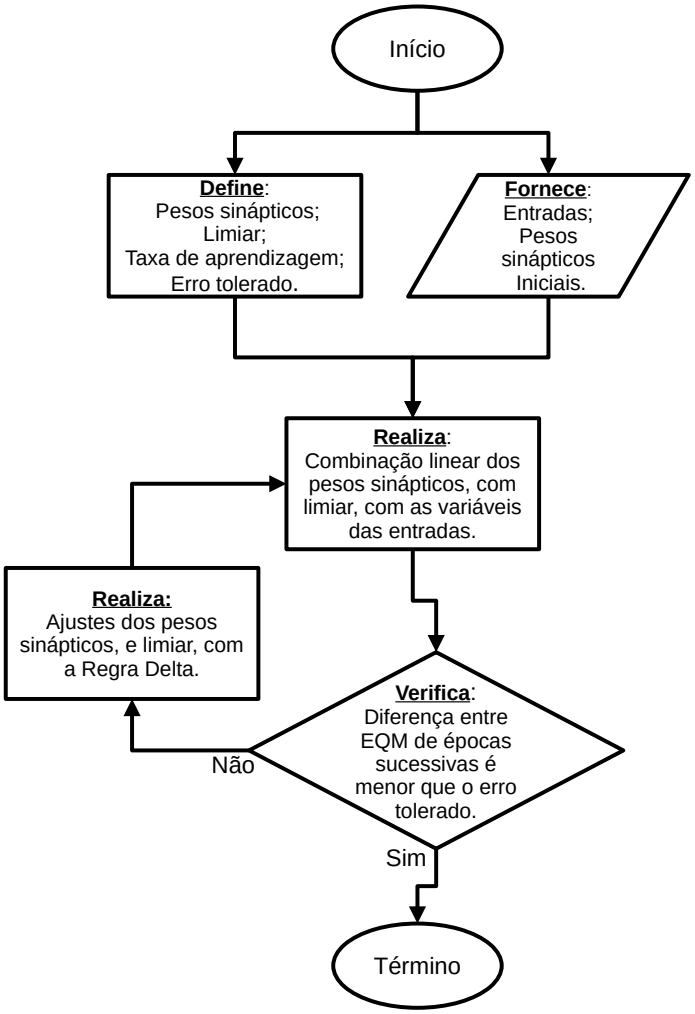

Figura 4 – Fluxograma apresentando as etapas do processo de treinamento do Adaline. Elaborado pelo autor.

1.3 As Flores Íris: caso não linearmente separável

Para o Perceptron, podemos dizer que em seu treinamento o foco repousa na magnitude do erro que a rede apresenta, por isso pode haver uma conclusão para o treinamento de forma que novos dados sejam categorizados incorretamente (Widrow; Lehr, 1990).

Em relação ao Perceptron, o Adaline inclui uma restrição extra referente ao cálculo e utilização do Erro Quadrático na atualização dos pesos sinápticos. Esta característica

permite que o Adaline seja capaz de construir um separador linear para dados que não sejam necessariamente linearmente separáveis.

Um exemplo disto pode ser visto ao considerar as flores Íris Versicolor e Íris Setosa como uma única categoria e treinar o Adaline para separar os dados desta categoria e da Íris Virgínica. A Figura 5 apresenta o diagrama de dispersão deste caso.

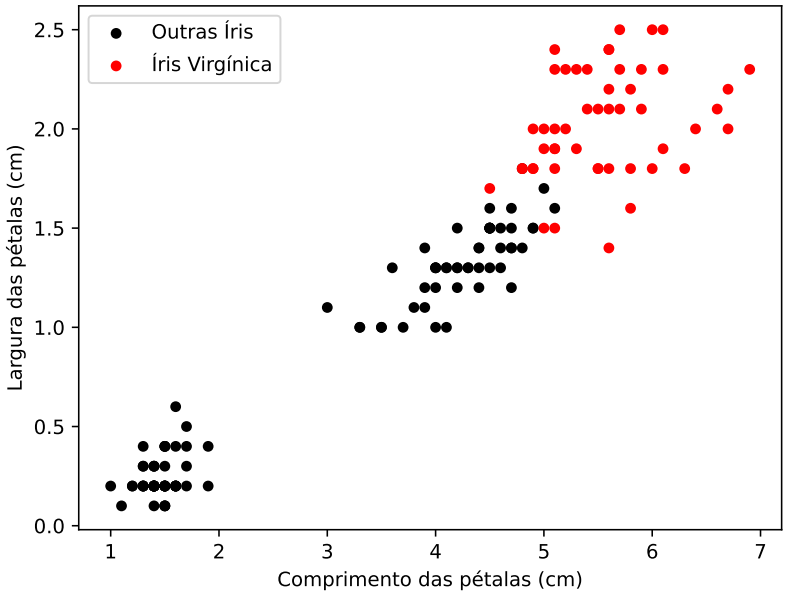

Figura 5 – Os dados dos comprimentos das pétalas das três flores Íris. Elaborado pelo autor.

Conforme pode ser observado pela Figura 5, esta distribuição não é linearmente separável, ou seja, não existe uma reta que separe os dois conjuntos de dados de modo que cada conjunto fique contido em apenas uma das regiões.

Uma RNA do tipo Perceptron, se treinada utilizando estes dados, não convergirá. Uma alternativa para que a Rede não permaneça treinando os pesos sinápticos durante um período de tempo indefinido seria estabelecer um número máximo de épocas para o treinamento.

Mesmo com esta nova condição de parada, não é possível garantir que o Perceptron apresente valores minimamente satisfatórios, ou seja, pesos sinápticos que estejam pelo menos apresentando uma pequena margem de erro. Um exemplo deste caso pode ser visto na Figura 6.

De acordo com Widrow e Lehr (1990), o método de aprendizado do Adaline e sua capacidade de generalização são seus maiores atributos, pois ele aprende sobre os padrões das características dos dados logo a princípio e então busca a melhor solução para os pesos sinápticos.

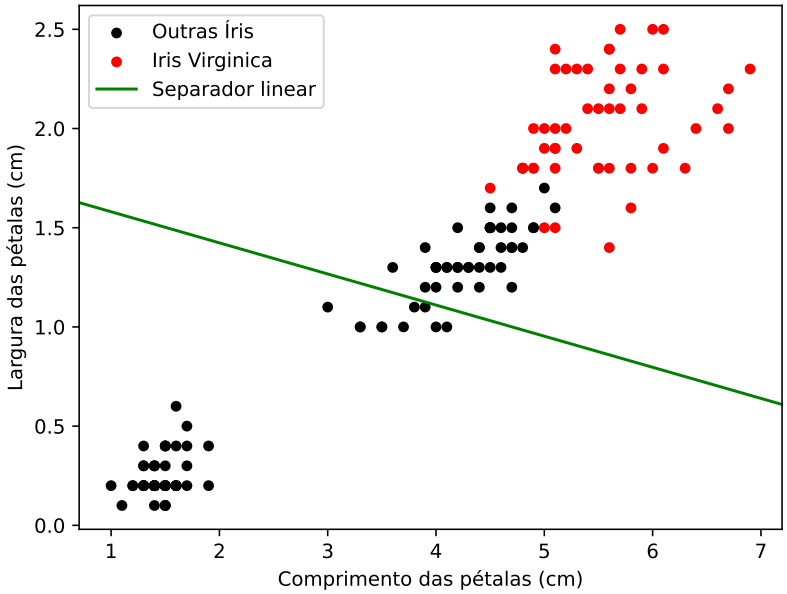

Figura 6 – Possibilidade de reta apresentada pelo Perceptron em dados não separáveis linearmente. Elaborada pelo autor.

Um exemplo desta vantagem do treinamento do Adaline frente ao do Perceptron pode ser visto na Figura 7, em que o treinamento do Adaline, focado na diminuição gradativa e direcionada do erro da rede, pode apresentar valores para os pesos sinápticos que representem uma reta com erro mínimo de separação dos dados não linearmente separáveis.

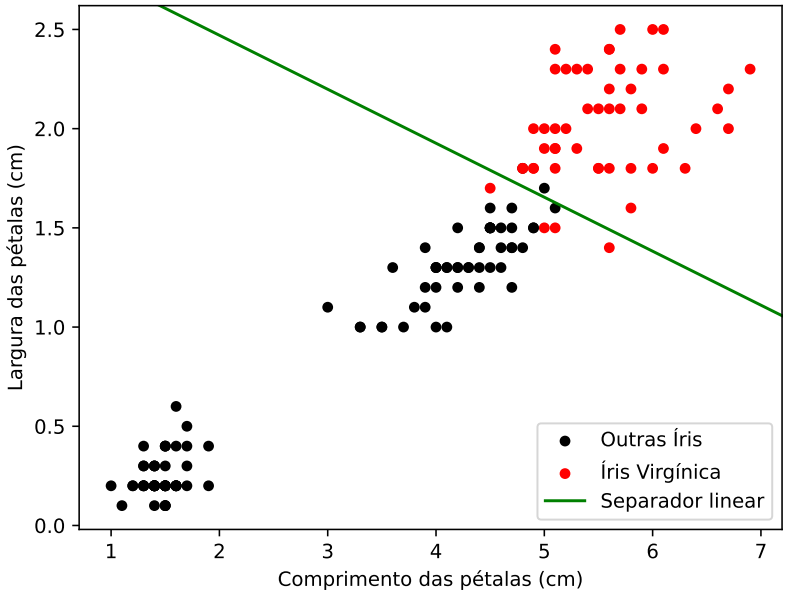

Figura 7 – Possibilidade de reta apresentada pelo Adaline em dados não separáveis linearmente. Elaborado pelo autor.

1.4 Implementando o Adaline em Python

O Algoritmo 1 apresenta a função `Adaline`, implementada em linguagem Python. Esta função recebe a matriz de amostras x, a matriz de saídas desejadas d, os pesos sinápticos iniciais aleatórios P, a taxa de aprendizagem η e o erro máximo tolerado ε.

```
import numpy as np
#Criação da função:
def Adaline(P,x,d,eta,epsilon):
    #Obtêm o número de amostras em x:
    m=len(x[0])
    #Combinação linear de pesos e entradas:
    u = np.dot(P,x)
    #Calcula o primeiro EQM:
    EQM_atual = np.sum(np.square(d-u))/m
    #Atribui número enorme à diferença entre EQMs:
    #(Fazer isto permite que o treinamento inicie)
    diferenca_EQM = np.inf
    #Zera o número de épocas:
    epocas = 0
    #Enquanto a diferenca entre EQMs for maior que epsilon:
    while (diferenca_EQM > epsilon):
        #Abre espaço para o novo EQM e guarda o anterior
        EQM_anterior = EQM_atual
        #Atualiza os pesos sinápticos:
        P += eta*np.dot((d-u),x.T)
        #Realiza a nova combinação linear:
        u = np.dot(P,x)
        #aumenta o número de épocas em 1:
        epocas = epocas + 1
        #Calcula o EQM da época atual:
        EQM_atual = np.sum(np.square(d-u))/m
        #Compara a diferença absoluta entre as EQMs:
        diferenca_EQM = abs(EQM_anterior - EQM_atual);
        print(f'Na epoca {epocas}, a matriz dos Pesos é {P}')
    print('A matriz dos pesos completamente treinada é:\n',P)
    print('Número de épocas:',epocas)
```

Algoritmo 1 – Função Adaline implementada em Python. Elaborada pelo autor.

Após realizar a definição da função `Adaline`, a rede estará pronta para receber os dados do treinamento e a definição de seus parâmetros.

Como exemplo, sua aplicação será ilustrada através do estudo da função booleana "OU". Esta função booleana é linearmente separável, como pode ser visto na seção **??** e por meio da Figura **??**, na qual uma reta é apresentada como candidata a separadora linear.

Sua tabela verdade pode ser adaptada para uma matriz x que contenha em sua última coluna valores "-1" com o propósito de possibilitar a obtenção do limiar θ em conjunto com o treinamento da matriz de pesos sinápticos P.

$$x = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \tag{1.8}$$

A disposição dos parâmetros de entrada da função Adaline é ilustrado no Algoritmo 2 de tal modo que as matrizes de amostras, x, das saídas desejadas, d e de pesos sinápticos P preservem as operações matriciais definida pela função `Adaline`.

Com o propósito de mostrar o modo que o algoritmo de treinamento do Adaline faz a busca pelos pesos sinápticos de forma direcionada, não importando quais sejam os pesos sinápticos iniciais, para o melhor conjunto de valores para pesos sinápticos possíveis, de acordo com a tolerância, será estabelecido um conjunto específico de pesos sinápticos iniciais.

Esta estratégia foi adotada, pois, executando este mesmo código com estes mesmos pesos sinápticos, o treinamento apresentará as mesmas épocas de treinamento e o mesmo valor final para os pesos sinápticos.

```
#Matriz de entradas com as amostras
x = np.array([[0, 0, 1, 1],[0, 1, 0, 1],[-1, -1, -1, -1]])
#Matriz de saídas desejadas
d = np.array([[0, 1, 1, 1]])
# Matriz de pesos iniciais
P = [[0.6, 0.5325319,  0.12664936]]
print(f'A matriz dos pesos é:\n{P}')
print(f'A matriz das entradas é:\n{x}')
print(f'A matriz de saídas desejadas é:\n{d}')
```

Algoritmo 2 – Fornecimento das entradas, saídas desejadas e pesos iniciais.
Elaborada pelo autor

A matriz de pesos P fornecida no Algoritmo 2 contém um elemento referente ao limiar, que no caso está na última coluna, o número 0.12664936. Utilizando uma taxa de aprendizagem $\eta = 0,01$ e a tolerância $\varepsilon = 0,0001$ na função `Adaline`, configuram-se os parâmetros necessários para o treinamento da rede com a função criada no Algoritmo 1. O Algoritmo 3, apresenta a sintaxe para este treinamento.

```
Adaline(P,x,d,0.01,0.0001)
```

Algoritmo 3 – Função Adaline com parâmetros definidos. Elaborado pelo autor.

Ao executar a linha de código fornecida no Algoritmo 3, após um total de aproximadamente 60 épocas, a rede como está configurada apresentará em sua saída a matriz treinada P, como $P = [[0.63299413, 0.59607851, -0.10840855]]$.

Dispondo os dados relativos às entradas para função booleana "OU" com suas respectivas saídas, juntamente o separador linear construído com os pesos sinápticos obtidos com este treinamento, tem-se a Figura 8.

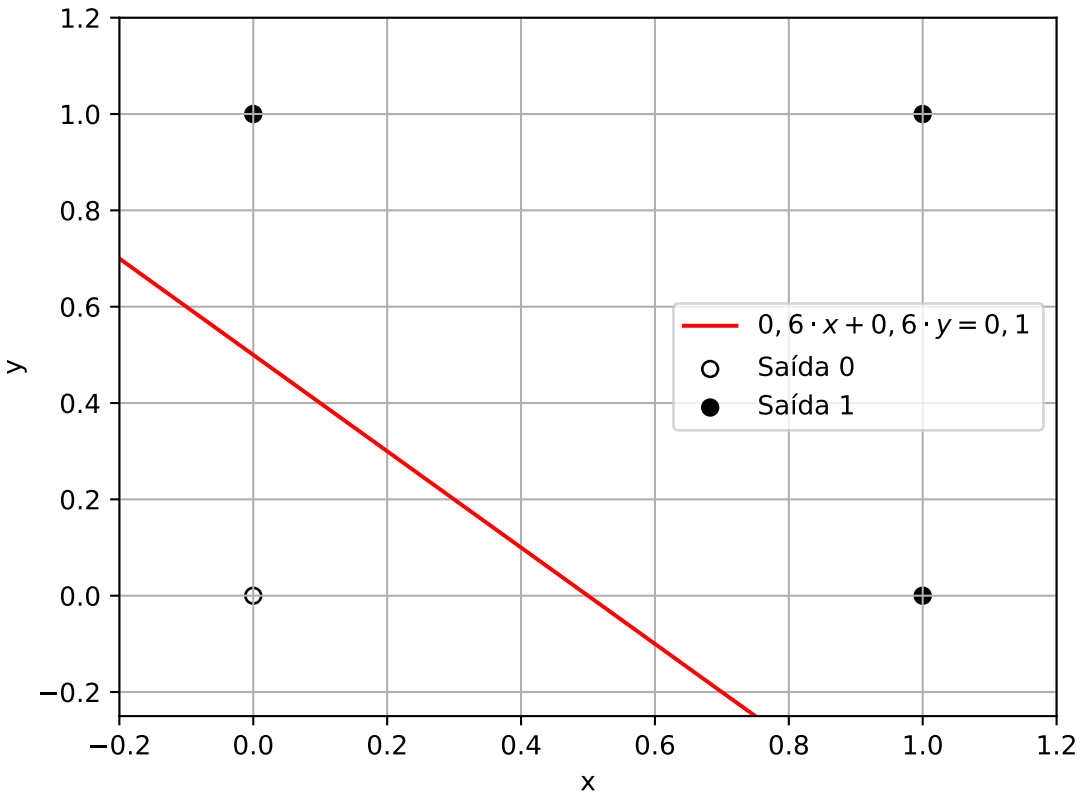

Figura 8 – Função booleana "OU" com reta obtida com treinamento. Elaborada pelo autor.

Mantendo os mesmos pesos sinápticos e taxa de aprendizado do exemplo anterior e, alterando apenas o erro tolerável para $\varepsilon = 0,000000001$, executa-se o Algoritmo 4 que, após 968 épocas de treinamento, tem como saída a matriz de pesos, $P = [0.5003851, 0.50038109, -0.2495456]$ e representação gráfica dada pela Figura 9.

```
P = [[0.6, 0.5325319,  0.12664936]]
Adaline(P,x,d,0.01,0.000000001)
```

Algoritmo 4 – Função Adaline com parâmetros definidos novamente. Elaborado pelo autor.

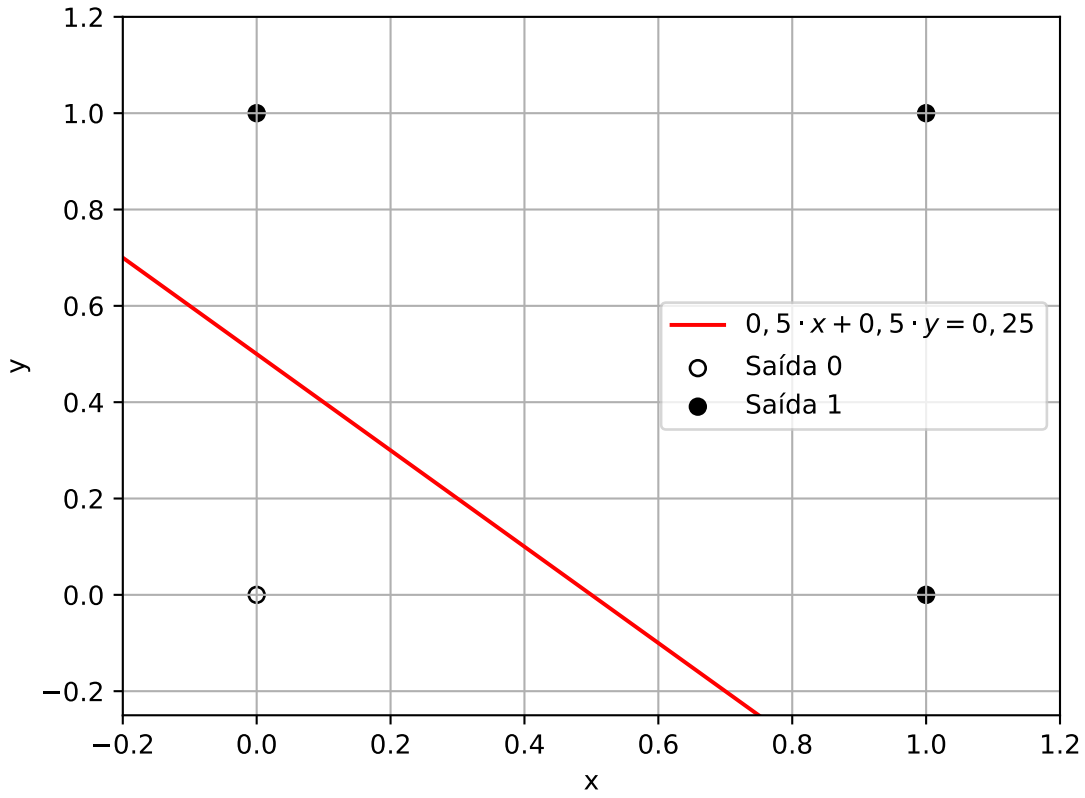

Figura 9 – Função booleana "OU" com coeficientes otimizados. Elaborado pelo autor.

A reta da Figura 9 está melhor posicionada devido ao menor erro ε admitido em relação à reta apresentada pela Figura 8, que possui um valor para ε relativamente maior. Isto implica no fato dos pesos sinápticos obtidos no treinamento realizado com o Algoritmo 4 estarem mais aptos a generalizar este Adaline para novos dados do que os pesos obtidos com o treinamento com o Algoritmo 3.

Indo além deste exemplo, o Adaline, segundo Widrow e Lehr (1990)

> "Com duas variáveis, um único Adaline pode implementar 14 das 16 funções lógicas existentes. Com mais variáveis, porém, apenas uma fração de todas as funções pode ser implementada, ou seja, que são linearmente separáveis. Combinações de neurônios artificiais ou[5] de redes neurais artificiais podem ser utilizadas para implementar funções que não são linearmente separáveis" (Widrow; Lehr, 1990, p.1418, tradução nossa).

A composição de neurônios diz respeito a conexão direta entre as saídas de uns na entrada de outros. A composição de dois ou mais neurônios do tipo Adaline é chamada de Madaline[6] e têm uma configuração específica, utilizando em sua estrutura algumas funções booleanas, e três tipos de treinamento desenvolvidos para ela (Widrow; Lehr, 1990). No capítulo **??** é apresentada as composições de Perceptrons com funções de ativação com saídas contínuas. Estas composições são chamadas de Perceptron Multicamadas.

[5] "With two inputs, a single Adaline can realize 14 of the 16 possible logic functions. With many inputs, however, only a small fraction of all possible logic functions is realizable, that is, linearly separable. Combinations of elements or networks of elements can be used to realize functions that are not linearly separable."

[6] Acrônimo para *Multiple Adaptive Linear Element.*

Bibliografia

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas**: Fundamentos Teóricos e Aspectos Práticos. 2. ed. São Paulo: Artliber, 2016. P. 431.

WIDROW, Bernard; LEHR, Michael A. **30 years of adaptive neural networks: perceptron, Madaline, and backpropagation**. **Proceedings of the IEEE**, v. 78, n. 9, p. 1415–1442, 1990. DOI: 10.1109/5.58323.