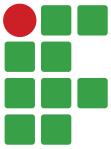


INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
Campus São Paulo
Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT

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

Victor Rodrigues de Oliveira

São Paulo - SP, 2025

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

Victor Rodrigues de Oliveira
Descobrindo Redes Neurais Artificiais - Minicurso sobre os modelos: Perceptron, Adaline e Perceptron Multicamadas/ Victor Rodrigues de Oliveira. – São Paulo - SP, 2025-
97 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Marco Aurélio Granero Santos

Dissertação (Mestrado) – Instituto Federal de Educação Ciência e Tecnologia de São Paulo
Campus São Paulo
Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, 2025.

1. Ensino Superior. 2. Aprendizado de Máquina. I. Victor Rodrigues de Oliveira. II. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo. III. Campus São Paulo. IV. Minicurso voltado para o aprendizado de Redes neurais Artificiais.

CDU 02:141:005.7

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.

Aprovado, São Paulo - SP, 26 de fevereiro de 2025:

Prof. Dr. Marco Aurélio Granero Santos
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - IFSP
Orientador e Presidente da banca

Prof. Dr. Samuel Francisco
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - IFSP
Membro da banca

Profª. Drª. Larissa Marques Sartori
Escola de Administração de Empresas de São Paulo da Fundação Getulio Vargas - FGV-EAESP
Membro da banca

São Paulo - SP
2025

À minha família.

Agradecimentos

Agradeço aos meus pais, Raquel e Elias, por todo o suporte, amor e atenção que me deram, desde quando mais novo até atualmente. Agora, mais velho, tenho um pouco da dimensão do quanto se sacrificaram lá atrás para que eu pudesse hoje desenvolver trabalhos como este.

Aos meus irmãos, Lucas e Otávio, por todo o companheirismo e afeto ao longo desta jornada e, mais especificamente, por me ajudarem com a produção e verificação de qualidade das videoaulas.

À minha esposa, Ariane, pela companhia, incentivo e cuidado que tive a sorte de receber ao longo destes anos de estudo e, principalmente, durante a elaboração deste trabalho.

Aos meus amigos e companheiros, pela fraternidade e por me ajudarem a manter minha curiosidade científica e meu espírito investigativo.

Ao Buda, ao Dharma e à Sangha.

"Mas é o óbvio que é tão difícil ver na maior parte do tempo. As pessoas dizem: 'Está na cara'. Mas que partes da sua cara você consegue ver, a não ser que alguém lhe mostre um espelho?"

— Isaac Asimov, *Eu, Robô*.

Resumo

Este trabalho apresenta uma introdução às Redes Neurais Artificiais (RNA) por meio da estruturação de um minicurso articulado em três frentes integradas. Estas frentes são compostas de: um conjunto de videoaulas cujo objetivo é apresentar os aspectos práticos e aplicados das RNA; um trabalho escrito contendo a teoria associada aos temas, o referencial teórico e links para acesso aos materiais e um conjunto de cadernos interativos digitais. Neste trabalho são apresentadas as RNA: *Perceptron*, o primeiro neurônio artificial com treinamento por algoritmo; o *Adaline*, a primeira RNA empregada em larga escala em problemas aplicados a sinais analógicos e regressões lineares e o Perceptron Multicamadas (PMC), a RNA que teve seu algoritmo de treinamento tido com referência para os algoritmos de treinamento das RNA nas últimas décadas. As videoaulas abordam estes assuntos com a adição de recursos visuais e outros exemplos que utilizam os cadernos interativos desenvolvidos em linguagem de programação Python. Para facilitar o acompanhamento dos temas estudados, estas frentes também foram reunidas no site Descobrindo Redes Neurais Artificiais (2025).

Palavras-chave: Redes Neurais Artificiais. Perceptron. Adaline. Perceptron Multicamadas. Minicurso.

Abstract

This work aims to present an introduction of Artificial Neural Networks (ANN) with a minicourse composed of three main approaches: A set of video lessons to present the practical and applied aspects of ANN; a written essay about the theory of ANN; and a group of interactive notebooks. During this work are presented: the Perceptrons, the first artificial neuron trained with the use of an algorihm; the Adaline, the first ANN used in large-scale applied to analog signals and linear regressions; and the Multilayer Perceptron (MLP), an ANN with an trainning algorithm used as reference in the training of other ANN in the last decade. The video lessons address these subjects with the addition of visual resources and other examples that use the interactive notebooks developed in Python programming language. To facilitate monitoring of the topics studied, this material is in the website: Descobrindo Redes Neurais Artificiais (2025).

Keywords: Artificial Neural Networks. Perceptron. Adaline. Multi-Layer Perceptron. Mini-course.

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Lista de abreviaturas e siglas

RNA	Rede(s) Neural(is) Artificial(is)
PMC	Perceptron Multicamadas
PE	Produto Educacional
XOU	Função booleana ou-exclusivo

Lista de símbolos

η	Letra grega Eta
$\vee$	Operador booleano “OU”
$\wedge$	Operador booleano “E”
$\oplus$	Operador booleano “XOU”
$\mathbb{R}$	Conjunto dos números reais

Sumário

Introdução

A presença das Redes Neurais Artificiais (RNA) na tomada de decisões, atualmente, é vasta e abrange diferentes campos da ciência e do cotidiano. De acordo com Telles, Barone e Silva (2020), Silva, Spatti e Flauzino (2016) e Kinsley e Kukieła (2020) seu uso pode ser identificado em áreas como medicina, ecologia, farmácia, acústica, indústria de alimentos, no setor automotivo e também no setor empresarial, por exemplo.

De fato, ela pode ser utilizada como uma ferramenta para o auxílio na tomada de decisões devido ao poder computacional aplicado à análise de grandes conjuntos de dados (Big Data) disponíveis desde as últimas décadas (Telles; Barone; Silva, 2020).

No âmbito educacional, Silva et al. (2023) realizaram um estudo tipo Estado da Arte para avaliar as aplicações de RNA, identificando estudos com foco em cenários como previsão de desempenho acadêmico, avaliação de nível cognitivo, reconhecimento de escrita à mão, detecção e reconhecimento de faces e eficiência de ensino.

O uso das RNA que aqui serão apresentadas estão fortemente baseados na ideia de classificação de padrões como forma de desenvolver ferramentas adequadas aos assuntos em que se aplicam. Os métodos de classificação de padrões são puramente matemáticos e sua implementação e análise podem ser identificados por meio de aspectos do pensamento computacional.

O pensamento computacional, referido como competência geral e habilidade específica na Base Nacional Comum Curricular (BNCC) relativa a Matemática (Brasil, 2017), bem como especificamente tratada de forma exclusiva em um documento complementar em Brasil (2022), pode ser relacionado com o estudo e identificação de padrões diversos, elaboração e execução de algoritmos.

A fim de correlacionar os estudos aplicados ao tema das RNA com o ensino de matemática na educação básica, este trabalho busca destacar elementos, conhecimentos e ferramentas matemáticas que são necessários para o entendimento do processo de construção e treinamento de RNA do tipo Perceptron, Adaline e Perceptron Multicamadas, os objetos de estudo deste trabalho.

Todas as construções de algoritmos serão executados por meio da linguagem de programação Python devido a sua condição de *Software livre* (Python Software Foundation, 2023; Gad; Jarmouni, 2021) de popularidade (Spectrum, 2024). Para a participação por meio de leitura e escrita dos algoritmos serão utilizados arquivos IPython, formato utilizado pelo software Jupyter Notebook, chamados de cadernos interativos.

Estes cadernos foram elaborados com o propósito de facilitar a implementação dos

algoritmos ao permitir que os blocos de comandos sejam executados de forma assíncrona. Além disto, é possível executar estes cadernos através do uso da plataforma Google Colab que é sem custos e,

> [...] permite que qualquer pessoa escreva e execute código Python arbitrário pelo navegador e é especialmente adequado para aprendizado de máquina, análise de dados e educação. Mais tecnicamente, o Colab é um serviço de notebooks hospedados do Jupyter que não requer nenhuma configuração para usar e oferece acesso sem custo financeiro a recursos de computação como GPUs (Google Inc., 2023).

Dessa forma, os cadernos elaborados possuem a característica de serem interativos, possibilitando o relacionamento de elementos presentes nos estudos de RNA e conhecimentos de matemática associado a elas que estão presentes no currículo da educação básica, como a investigação e experimentação (Brasil, 2017).

Para o leitor que não possui nenhuma familiaridade com linguagens de programação estruturada, recomenda-se a leitura do Capítulo 1 de Martins (2021), onde é feita a ambientação e apresentação de exemplos simples da linguagem de programação Python de modo a estabelecer um ponto de partida comum aos participantes para que possam usufruir por completo do minicurso.

Diante do exposto, este trabalho apresenta-se como um Produto Educacional (PE) composto por 13 videoaulas, 4 cadernos do Jupyter Notebook e este material teórico de suporte, com o intuito de complementar o aprendizado sobre RNA.

Segundo as diretrizes propostas por Rizzatti et al. (2020) para um PE, estes materiais estarão disponíveis para replicação por terceiros a fim de responder a pergunta "Quais conhecimentos matemáticos estão envolvidos no aprendizado de máquina?".

Sendo assim, no capítulo 1, é introduzido o modelo de neurônio artificial apresentado por McCulloch e Pitts (1943), bem como seu funcionamento com o uso de linguagem matemática. É importante mencionar que apesar da inspiração do neurônio artificial se basear no que se entendia até então do funcionamento de um neurônio biológico, as semelhanças se encerraram neste fato. Sobre isso podemos citar Widrow e Lehr (1990, p.1417, tradução nossa), "Embora o pássaro tenha servido de inspiração para o desenvolvimento do avião, os pássaros não têm hélices e os aviões não funcionam por meio do bater de asas com penas".[1]

O capítulo 2, introduz uma estratégia de treinamento para o neurônio de McCulloch-

[1] "Although the bird served to inspire development of the airplane, birds do not have propellers, and airplanes do not operate by flapping feathered wings", no original, em inglês.

Pitts por meio do Perceptron. Esta RNA merece uma atenção especial devido aos impactos que teve no aumento de expectativas no uso das redes neurais, seja na ficção-científica ou na bolsa de valores. Este modelo inclusive mereceu um estudo detalhado realizado por Minsky e Papert (1988), que ao apresentar as limitações para esta RNA, influenciou na queda de interesse nos estudos relativos às RNA.

Segundo Widrow e Lehr (1990) a diminuição da empolgação em relação ao Perceptron influenciou o direcionamento das pesquisas com o Adaline, que é objeto de estudos no capítulo 3. Inicialmente aplicado em processamento de sinais analógicos, este modelo de RNA utiliza em seu treinamento a ideia inteligente de mudar o foco na busca do acerto para o foco em diminuir o erro e apresentar o melhor que a rede pode oferecer ao problema em que foi aplicada.

No capítulo 4, será apresentado o Perceptron Multicamadas. Por muito tempo esta RNA não superou o estigma que sua forma mais simples, o Perceptron com um neurônio e não havia um consenso nas estratégias adotadas para seu treinamento. Após o estabelecimento de uma regra de treinamento, o *backpropagation*, que detém o potencial de ajustar os pesos sinápticos para valores otimizados, houve um período de reavivamento dos esforços relativos nos estudos das RNA que perdura até hoje.

No capítulo 5 estão descritos os conteúdos apresentados nas videoaulas e nos cadernos Jupyter Notebook. Os assuntos abordados nas videoaulas estão de acordo com os conteúdos apresentados aqui no texto. O principal diferencial entre eles é a forma em que são apresentados, unindo as vantagens do texto relativas aos diferentes tempos de consumo de informação de cada leitor, com o direcionamento proveniente dos recursos visuais e que as produções audiovisuais podem oferecer. Por vezes, tópicos discutidos neste texto estarão lá apresentados, e vice-versa, porém existem exemplos exclusivos para cada um deles.

Os cadernos interativos tem parte de seu código apresentado neste texto e são utilizados nas videoaulas. Eles foram elaborados com o mínimo de informação necessária para seu entendimento, então recomenda-se assistirem as videoaulas antes de estudá-los por meio da execução dos códigos contidos neles ou em sua modificação. Recomenda-se esta ordem para o consumo e interação com os elementos deste Produto Educacional. Com isto, recomenda-se que sejam assistidas as videoaulas antes da leitura de cada capítulo correspondente, seguido da interação com os cadernos correspondentes.

1 Neurônio de McCulloch e Pitts

Os pesquisadores Warren McCulloch [1] e Walter Pitts [2], em 1943, apresentaram um modelo teórico que usa álgebra booleana, com "0" e "1" indicando sinais negativos e positivos, respectivamente, para simular o funcionamento de um neurônio biológico e aplicá-lo na tomada de decisões.

Este modelo foi apresentado no artigo "A logical calculus of the ideas immanent in nervous activity (1943)". Esse trabalho foi um marco, considerado por muitos o inicial, para o desenvolvimento do campo de estudos relativos às Redes Neurais e Inteligência Artificial. O neurônio de McCulloch-Pitts é o que se chama de discriminador binário, ou seja, é um classificador de dados para duas classes distintas (Kovács, 2023).

Com o uso deste neurônio artificial, foram elaboradas máquinas eletrônicas em que o modelo do neurônio era adaptado com uso de componentes eletrônicos como resistores ajustáveis e sensores ópticos, e desempenhava o papel de identificador de padrões, como pode ser visto na Figura 1 (Silva; Spatti; Flauzino, 2016).

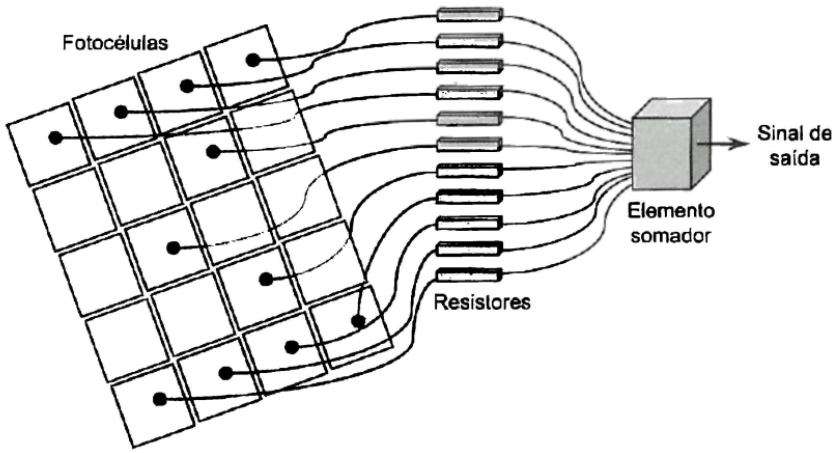


Figura 1 – Modelo ilustrativo de uma aplicação do Neurônio de McCulloch-Pitts com componentes eletrônicos. Fonte: Silva, Spatti e Flauzino (2016, p.58)

Para possibilitar a identificação de padrões que o Neurônio de McCulloch-Pitts é capaz de executar, que é relativa a categorização destes dados em uma de duas classes distintas, é necessário que os dados também sejam separáveis em duas categorias.

[1] Warren Sturgis McCulloch, nascido em 1898 e falecido em 1969, foi um neurofisiologista, era conhecido pelo seu trabalho sobre os fundamentos de certas teorias do cérebro e pela sua contribuição para a área da cibernética (Tappert, 2019).

[2] Walter Pitts, nascido em 1923 e falecido em 1969, foi um matemático formado no MIT. Escreveu diversos artigos sobre métodos de simular, ou até reproduzir, o funcionamento do cérebro humano por meio da linguagem matemática e de máquinas elétricas. Visto em https://home.csulb.edu/~cwallis/artificialn/walter_pitts.html. Acesso em: 25 de fevereiro de 2025.

Esta divisão é realizada de modo que, sendo possível dispor os dados em um plano cartesiano em duas dimensões, eles seriam completamente separados por uma reta. Por conta desta característica, o Neurônio de McCulloch-Pitts também é chamado de discriminador, ou separador linear.

A Figura 2 apresenta um plano cartesiano no qual uma reta de equação $2x - y = 1$ é representada, possibilitando uma discussão sobre dos espaços que ela delimita.

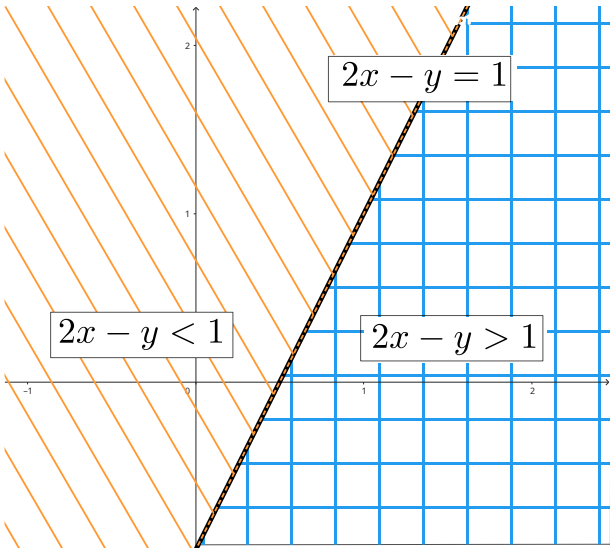


Figura 2 – Plano dividido em duas regiões pela reta de equação $2x - y = 1$.
Elaborado pelo autor.

Analisando o plano no $\mathbb{R}^2$ exemplificado na Figura 2, a região laranja representa os pares de pontos (x, y) em que $2x - y < 1$ e a região azul em que $2x - y > 1$ enquanto a reta $2x - y = 1$ é a separadora linear.

De um modo geral, para dados que podem ser dispostos no plano $\mathbb{R}^2$, ou seja, com duas variáveis, o discriminador linear é uma reta que faz a separação do plano em outros dois semiplanos com os dados contidos em um ou outro deles.

Com três variáveis, ou seja, no espaço tridimensional $\mathbb{R}^3$, uma superfície plana ou simplesmente um plano faz o papel do discriminador linear. Em dimensões maiores, e com mais variáveis, o discriminador linear que separa o espaço $\mathbb{R}^n$ em duas regiões distintas é comumente chamado de hiperplano.

É importante enfatizar que a separação linear realizada deverá ser apresentada conforme as características relativas aos dados. Eventualmente, pode não ser possível realizar a separação com retas, planos ou hiperplanos, sendo necessária a utilização de outras curvas e superfícies para tal. Porém, este tipo de separação não tem possibilidade de ser executada com somente um neurônio de McCulloch-Pitts (Minsky; Papert, 1988).

1.1 O funcionamento do Neurônio de McCulloch-Pitts

A implementação do neurônio artificial com componentes eletrônicos, exemplificado na Figura 1, receberá os dados cuidadosamente tratados para serem lidos de modo eficiente pelos sensores, e então apresentará uma saída, um valor numérico, que estará associado a uma categoria.

O funcionamento do neurônio artificial de McCulloch-Pitts pode ser apresentado como no diagrama da Figura 3.

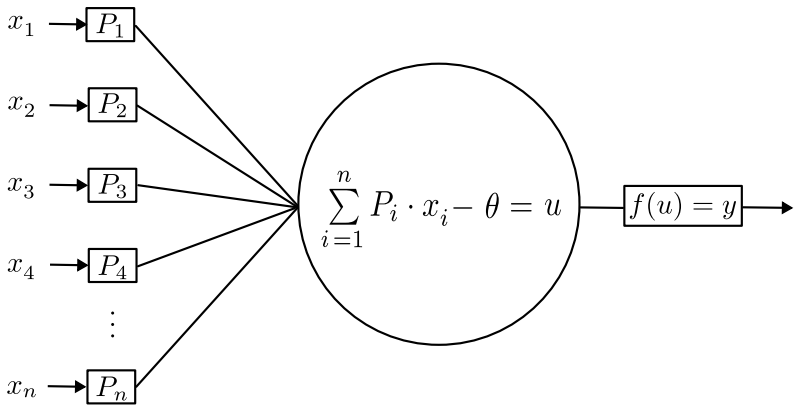


Figura 3 – Diagrama apresentando as entradas e o funcionamento do neurônio artificial de McCulloch-Pitts. Elaborada pelo autor.

As variáveis $x_1, x_2, \cdots, x_n$ representam a informação externa que, quando traduzida para a linguagem matemática, serão repassadas ao neurônio artificial. Estas informações são valores que podem se originar de medições objetivas de fenômenos e características em determinados contextos ou de outras fontes como conjuntos numéricos.

Os valores $P_1, P_2, \cdots, P_n$ são chamados de pesos sinápticos e são ponderadores da importância das variáveis de mesmo índice. Isto significa que os pesos reforçam determinadas características apresentadas pelas variáveis e, atribuem maior ou menor importância para elas com o objetivo de realizar a categorização.

O símbolo $\sum$ é usado para indicar a soma, neste caso relativa ao produto de cada entrada com seu respectivo peso sináptico, estabelecendo uma combinação linear, com o índice i variando de 1 até o número n que é máximo de variáveis fornecidas pelas amostras ao neurônio artificial.

A letra greta θ representa um número que comumente é chamado de limiar, *bias* ou *threshold*, que atua como um valor de referência em relação à combinação linear obtida.

A função de ativação f recebe este valor, já considerando o θ, e então devolve uma saída y que, ao ser analisada pelo operador da Rede Neural Artificial, indicará a atribuição das categorias. A classificação, neste caso, é realizada apresentando duas possibilidades

distintas e a função de ativação que determinará, a partir dos valores reais apresentados por u, as saídas binárias (Kovács, 2023).

Em resumo, para o neurônio de McCulloch-Pitts, a soma dos produtos dos valores de entradas por seus respectivos pesos deve atingir um valor maior do que θ para que a função de ativação possa devolver o valor associado a um conjunto e, quando a soma não superar esse limiar, a função de ativação devolverá o valor associado a outro conjunto. Em se tratando de redes neurais envolvendo o neurônio de McCulloch-Pitts o limiar também será o parâmetro de referência na função de ativação. Os pesos sinápticos e limiar tem seus valores fornecidos durante a implementação da rede neural (Kovács, 2023).

1.2 Implementando funções booleanas com o neurônio de McCulloch-Pitts

Como forma de exemplificar o funcionamento do neurônio artificial, serão implementadas as funções booleanas "E" e "OU" com duas variáveis. Estas duas funções podem ser implementadas com o uso de somente um neurônio artificial. Ambas são caracterizadas por receberem em suas entradas dois valores binários "0" e "1" e devolverem uma saída binária que também será um dentre estes dois valores.

Considerando que estes dados estão com duas variáveis, é possível realizar sua disposição em um plano cartesiano com eixos x_1 e x_2, de forma que eles sejam considerados pares ordenados. Seguindo nesta adaptação, pode-se então assumir que categorizar estes dados linearmente é equivalente a obter uma reta que divide o plano em duas regiões de forma que em cada uma delas estejam os pontos que apresentam a mesma saída pela função de ativação, como pode ser observado na Figura 4 ao considerar que os pesos sinápticos do neurônio podem ser interpretados, em um caso com duas variáveis, como coeficientes da equação de uma reta.

A função booleana "E" apresentará o valor "1" somente quando nas duas entradas este valor estiver presente. Considerando f como a função "E", a tabela verdade para as entradas x_1 e x_2 assumindo valores binários é dada por:

x_1	x_2	$f(x_1, x_2) = x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 1 – Tabela verdade de função booleana "E".

onde o sinal "$\wedge$" indica a atuação da função booleana "E" nas variáveis x_1 e x_2.

Dispondo os pontos referentes às entradas da função "E" no plano cartesiano, tal que os pares de entrada que têm saída "1" são representados por um ponto e, os pares de entrada com saída "0" são representados por um círculo, tem-se a representação gráfica do problema associado à função booleana "E" ilustrado na Figura 4.

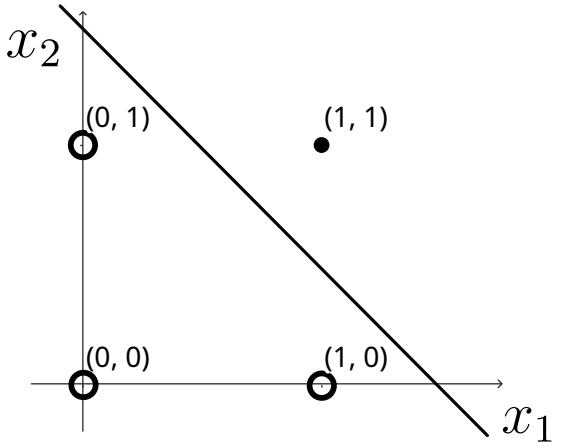


Figura 4 – Representação de uma função booleana "**E**" dentre as várias possíveis.
Elaborado pelo autor.

Ainda na Figura 4, a reta apresentada é uma candidata a separadora linear enquanto os pontos $(0,0), (0,1)$ e $(1,0)$ estão agrupados em uma região, e o ponto $(1,1)$ está em outra. A tarefa da Rede Neural Artificial será, então, a de encontrar os coeficientes que esta reta deve ter para que exista uma separação, linear, como a apresentada na Figura 4.

Deste modo, o neurônio de McCulloch-Pitts pode ser representado, para este problema, por meio do diagrama de blocos ilustrado na Figura 5, em que os parâmetros pesos P_1 e P_2 e o limiar θ devem ser ajustados pela rede neural de modo a produzir a classificação desejada.

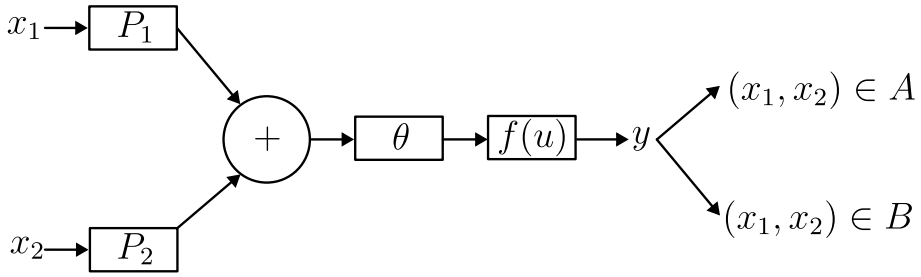


Figura 5 – Diagrama das etapas a serem executadas pelo neurônio artificial.
Elaborado pelo autor.

É possível escrever a equação (1.1) com pesos sinápticos ponderando as variáveis das amostras das entradas x_1 e x_2 utilizando a Tabela 1.

$$\begin{cases} P_1 \cdot 0 + P_2 \cdot 0 = 0 \\ P_1 \cdot 0 + P_2 \cdot 1 = 0 \\ P_1 \cdot 1 + P_2 \cdot 0 = 0 \\ P_1 \cdot 1 + P_2 \cdot 1 = 1 \end{cases} \tag{1.1}$$

A Equação 1.1 apresenta um sistema de equações lineares. Este sistema é chamado de **Sobredeterminado** e, geralmente, não apresenta valores que satisfaçam todas as equações simultaneamente devido ao número de equações, também chamado de restrições, ser maior que o número de variáveis (Anton; Rorres, 2012, p.241).

Adotando, por exemplo, $P_1 = P_2 = 1$ o sistema linear da Equação (1.1), considerando também o limiar θ resulta em:

(x_1, x_2)	$P_1.x_1 + P_2.x_2 - \theta = u$
$(0,0)$	$1.0 + 1.0 - \theta = \quad - \theta$
$(0,1)$	$1.0 + 1.1 - \theta = 1 - \theta$
$(1,0)$	$1.1 + 1.0 - \theta = 1 - \theta$
$(1,1)$	$1.1 + 1.1 - \theta = 2 - \theta$

Tabela 2 – Combinação linear entre os pesos e variáveis dos dados da função "E". Elaborado pelo autor.

O limiar θ opera como um critério para a rede e deve ser escolhido (ou ajustado) de forma que as saídas apresentem enquadramento a uma ou outra categoria. Desse modo, pode-se adotar um valor para θ no intervalo $]1, 2]$ pois, de acordo com a Tabela 2, a combinação linear que deverá estar associada à saída "1" é somente a da última linha enquanto todos os valores das outras linhas devem ser associados a "0".

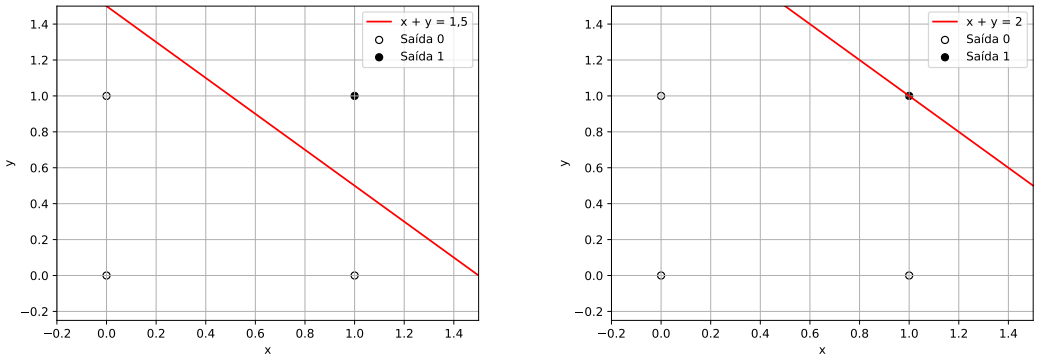
Adotando-se então o limiar $\theta = 1,5$, a função de ativação pode ser escrita como:

$$y = f(u) = \begin{cases} 1 & se \quad u \geqslant 0 \\ 0 & se \quad u < 0. \end{cases} \tag{1.2}$$

Sendo assim, os coeficientes da reta de equação: $ax + by = c$ para que ela atue como separador linear para o conjunto de quatro pontos com correspondência com a função boolena "E" podem ser: $a = b = 1$ e o coeficiente independente $c = 1,5$.

Este resultado pode ser observado na Figura 6a, em que a reta $x + y = 1,5$ divide o plano em duas regiões: uma delas, $x + y < 1,5$, contendo os pontos $(0,0)$, $(0,1)$ e $(1,0)$ cujas saídas da rede neural são "0"; e, outra região, $x + y \geqslant 1,5$, contendo o ponto $(1,1)$ cuja saída é "1".

Explorando um pouco mais esta situação, caso a reta tenha coeficiente independente c igual a 2, a expressão para função de ativação apresentada na Equação (1.2) ainda permaneceria válida, porém, o ponto $(1, 1)$ estaria contido na reta de equação $x + y = 2$, como visto na Figura 6b.



(a) Gráfico da função "E" representado pela reta $x + y = 1,5$.

(b) Gráfico da função "E" representado pela reta $x + y = 2$.

Figura 6 – Gráficos da função "E" com diferentes limiares. Elaborado pelo autor.

Para representar a função booleana "OU", pode-se adotar a mesma estratégia utilizada no exemplo referente a função booleana "E", ou seja, obter um separador linear conforme ilustrado na Figura 7. Neste caso, também pode-se utilizar os valores recebidos dos pesos sinápticos do neurônio como coeficientes da equação da reta separatriz.

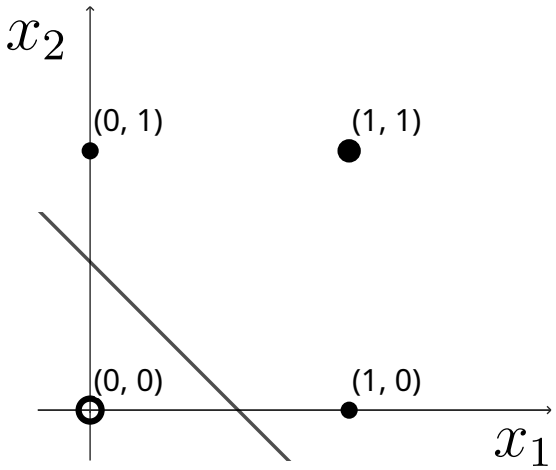


Figura 7 – Representação gráfica de uma função booleana "OU". Elaborado pelo autor

A tabela verdade para as entradas x_1 e x_2 assumindo valores binários é observada na Tabela 3 em que o sinal de $\vee$ está indicando a atuação da função booleana "**OU**" nas variáveis x_1 e x_2.

x_1	x_2	$f(x_1, x_2) = x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 3 – Tabela verdade da função booleana "OU".

Escrevendo as combinações lineares entre as entradas que x_1 e x_2 podem assumir e os pesos P_1 e P_2, tem-se

$$\begin{cases} P_1 \cdot 0 + P_2 \cdot 0 = 0 \\ P_1 \cdot 0 + P_2 \cdot 1 = 1 \\ P_1 \cdot 1 + P_2 \cdot 0 = 1 \\ P_1 \cdot 1 + P_2 \cdot 1 = 1 \end{cases} \tag{1.3}$$

A Equação (1.3) também é um sistema linear sobredeterminado. Similar ao que foi feito no exemplo da função booleana "E", também pode-se estabelecer os pesos sinápticos convenientemente. Inclusive, seria possível repetir aqui os pesos sinápticos adotados na Tabela 2.

Porém, para evidenciar as diferentes possibilidades de pesos que podem ser utilizados, e a conveniência do limiar escolhido, serão adotados dois valores iguais a $\frac{1}{2}$, ou seja, $P_1 = P_2 = \frac{1}{2}$. Realizando os procedimentos, linha por linha, tem-se:

(x_1, x_2)	$P_1.x_1 + P_2.x_2 - \theta = u$
$(0,0)$	$\frac{1}{2}.0 + \frac{1}{2}.0 - \theta = \quad - \theta$
$(0,1)$	$\frac{1}{2}.0 + \frac{1}{2}.1 - \theta = \frac{1}{2} - \theta$
$(1,0)$	$\frac{1}{2}.1 + \frac{1}{2}.0 - \theta = \frac{1}{2} - \theta$
$(1,1)$	$\frac{1}{2}.1 + \frac{1}{2}.1 - \theta = 1 - \theta$

Tabela 4 – Combinação linear dos pesos com as variáveis.

Desse modo, percebe-se que o limiar θ pode ser um valor entre 0 e $\frac{1}{2}$, pois, de acordo com a Tabela 4, a combinação linear que deverá ser associada a "0" é somente a da primeira linha, enquanto que todos os valores das outras linhas devem ser associados a "1".

Adotando $\theta = \frac{1}{2}$, a função de ativação pode ser escrita como:

$$y = f(u) = \begin{cases} 1 & se \quad u \geqslant 0 \\ 0 & se \quad u < 0. \end{cases} \tag{1.4}$$

que representa a implementação da função lógica "OU" com um neurônio de McCulloch-Pitts.

A Figura 8 ilustra graficamente os dados de entrada, as saídas e o separador linear dado pela reta de equação: $\frac{x}{2} + \frac{y}{2} = \frac{1}{2}$.

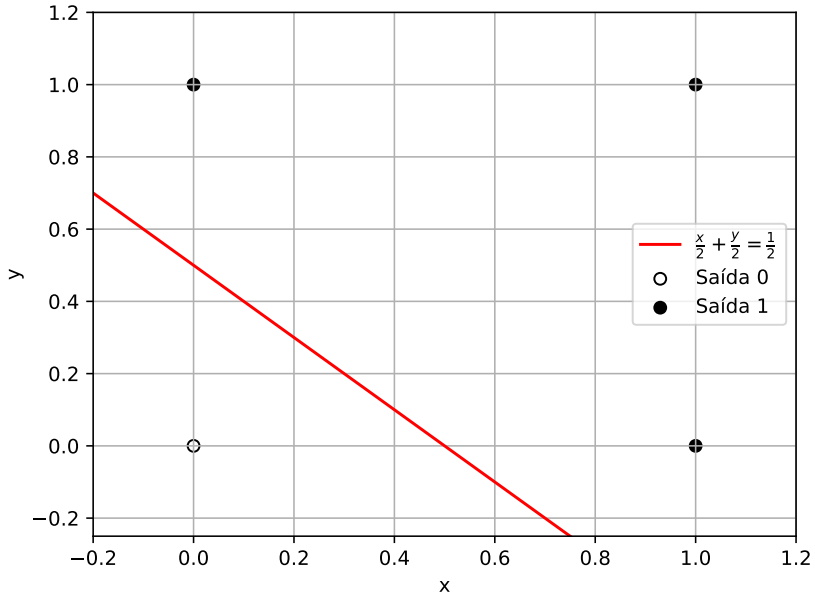


Figura 8 – Gráfico da função "OU" representado pela reta de equação $\frac{x}{2} + \frac{y}{2} = \frac{1}{2}$. Elaborado pelo autor.

Os pesos sinápticos P_1 e P_2 de ambos neurônios foram atribuídos arbitrariamente, e os limiares θ, de modo conveniente. A observação do atendimento das condições da função booleana estabelece alguns limites para esta escolha, mas, teoricamente, existem infinitas combinações de pesos e limiares, e consequentemente, de coeficientes da reta que realizem a separação do plano do modo necessário a representar corretamente cada uma das funções exemplificadas.

É importante notar que a função de ativação utilizada nas duas implementações esteve diretamente ligada com o limiar θ de maneira que este valor foi explicitamente adotado no critério desta função. Em ambos os casos, pode-se identificar o uso da função sinal, ou degrau,[3] que pode ser expressa, para algum número u real,

$$H(u) = \begin{cases} 1 & se \quad u \geqslant \theta \\ 0 & se \quad u < \theta \end{cases}, \tag{1.5}$$

o que resulta nas saídas da função de ativação serem somente valores binários.

Nos exemplos apresentados anteriormente foi possível realizar a separação dos dados com somente um neurônio pois as variáveis nas entradas eram binárias e linearmente separáveis, ou seja, por meio de combinações lineares dos pesos com as entradas, são capazes serem classificados em uma ou outra categoria.

[3] Também conhecida como função de Heaviside, em homenagem ao matemático e engenheiro eletricista Oliver Heaviside (1850 - 1925)

Nem todos os conjuntos de dados são linearmente separáveis e isso faz com que um neurônio de McCulloch-Pitts possa não apresentar uma classificação adequada.

Para duas entradas, x_1 e x_2, a função booleana *ou-exclusivo*, abreviada como "XOU", apresenta valores binários "1" para combinações que possuam exclusivamente uma ou outra entrada igual à "1", e "0" caso contrário, conforme ilustrado na Figura 9.

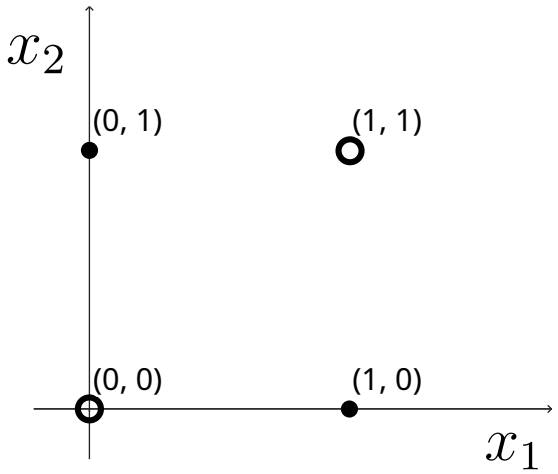


Figura 9 – Disposição dos pontos no plano com saídas associadas à função booleana "XOU". Elaborado pelo autor.

Esta função não pode ser implementada com somente um neurônio artificial (Minsky; Papert, 1988). Ainda segundo Widrow e Lehr (1990) e Kovács (2023), na impossibilidade de um neurônio representar uma função booleana, uma combinação de funções, e consequentemente de neurônios, será capaz de realizá-la. Entretanto, este tipo de rede consiste no uso de mais de uma camada de neurônios, assunto tratado no capítulo 4.

Um fator importantíssimo para o neurônio de McCulloch-Pitts assim apresentado, é a atribuição dos pesos sinápticos e limiares manualmente. Nos exemplos que foram utilizados isto não demandou muita criatividade, porém, este neurônio artificial é capaz de realizar a categorização de conjuntos de dados linearmente separáveis com centenas, ou milhares de dados.

Dessa forma, nestes casos há a necessidade da atribuição automatizada, por meio de algoritmos, de pesos sinápticos adequados. Isto é chamado de treinamento da Rede Neural. Nos capítulos 2 e 3, serão apresentados dois modos distintos de se realizar o treinamento e a obtenção destes pesos sinápticos de modo automático.

2 Perceptron

Conforme as tecnologias digitais foram desenvolvidas e aprimoradas, elementos como o Perceptron, o Adaline e outras Redes Neurais Artificiais (RNA) que utilizam a ideia de treinamento com o auxílio de algoritmos, surgiram com suas respectivas propostas para ajustar automaticamente os pesos sinápticos do neurônio artificial (Géron, 2019).

Originalmente concebido com a intenção de simular o funcionamento da retina para reconhecer padrões geométricos (Figuras 10 e 11), o Perceptron é constituído por um neurônio artificial desenvolvido por McCulloch e Pitts.

O Perceptron utiliza este modelo de neurônio com a adição da possibilidade de estabelecer pesos sinápticos adequados a cada problema proposto de maneira automatizada, por meio de um treinamento. Este foi um dos motivos desta RNA atrair diversos pesquisadores e atribuírem diversas expectativas a este modelo para aplicações no campo da Inteligência Artificial (IA)(Silva; Spatti; Flauzino, 2016).

Esta estratégia de obter os pesos sinápticos adequados contribuiu para o aumento das expectativas em relação às capacidades das RNA e, também, no que diz respeito a atuação das máquinas computadorizadas em tomadas de decisões que antes eram exclusivas dos seres humanos (Géron, 2019; Haykin, 2001).

Devido a grande atenção que foi depositada no Perceptron, em 1969, Minsky e Papert (1988) apresentam pela primeira vez, por meio da formalização matemática, as limitações que o Perceptron possui. Essa abordagem é dita como um dos itens que colaborou com a estagnação relativa ao aprendizado de máquina e IA, ainda na década de 1980 (Silva; Spatti; Flauzino, 2016; Bottou, 2017).

No livro Perceptrons: An Introduction to Computational Geometry (1988) é apresentado o "Teorema da convergência". Este teorema garante que, se o conjunto de classes do problema a ser mapeado for linearmente separável, então o Perceptron converge. Este teorema permite atribuir o título de verificador de separabilidade linear de um conjunto de dados com tal potencial. Um dos exemplos utilizados para apresentar as limitações do Perceptron é sua incapacidade de implementar a função booleana *ou-exclusiva* ("XOU").

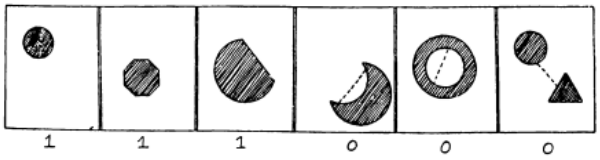


Figura 10 – Classificação em figuras planares convexas e não-convexas que um neurônio de McCulloch-Pitts é capaz de realizar: 1 é para convexas e 0 é para não-convexas.
Fonte: Minsky e Papert (1988, p.6).

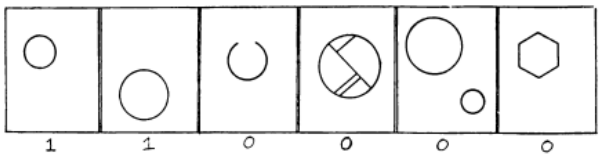


Figura 11 – Classificação que um neurônio McCulloch-Pitts é capaz de realizar para figuras planares como sendo circulares ou não.
Fonte: Minsky e Papert (1988, p.6).

Seu criador, Frank Rosenblatt[1], apresentou este modelo de RNA em 1960 (Widrow; Lehr, 1990). Por consistir na implementação de um neurônio de McCulloch-Pitts, o neurônio desta rede ainda mantém a função de armazenar informações numéricas. O diferencial é o treinamento realizado para que ocorram os ajustes de seus pesos sinápticos por meio da adaptação de um modelo de aprendizado desenvolvido na tentativa de explicar o funcionamento de neurônios biológicos (Kovács, 2023; Géron, 2019).

A estrutura simples do Perceptron também reflete seu modo de funcionamento simplificado. Como esta é uma implementação do Neurônio de McCulloch-Pitts, o processo de receber e processar as informações fornecidas a ele ainda é o mesmo, ou seja, quando nele são introduzidas as variáveis referentes aos dados, é realizada a ponderação de cada uma delas de forma que haja uma combinação linear destes valores com seus respectivos pesos sinápticos, a comparação com o limiar e, ao final do processo, a apresentação de uma saída dentre duas possíveis após ser aplicada uma função de ativação nesta diferença entre combinação linear e o limiar.

Para esta RNA, todos os pesos sinápticos são treinados de acordo com o algoritmo, incluindo o limiar, que também será tratado como um peso sináptico neste processo.

Os valores dos pesos sinápticos P_i e do limiar θ são a princípio aleatórios, e o processo de treinamento estabelecerá os valores finais adequados que permitirão ao Perceptron realizar as categorizações corretamente. Uma representação esquemática apresentando o processo amostra por amostra pode ser visto na Figura 12, no qual k representa o número da amostra de treinamento em questão.

[1] Nascido em 1928 e falecido em 1971, foi um psicólogo pesquisador (Tappert, 2019).

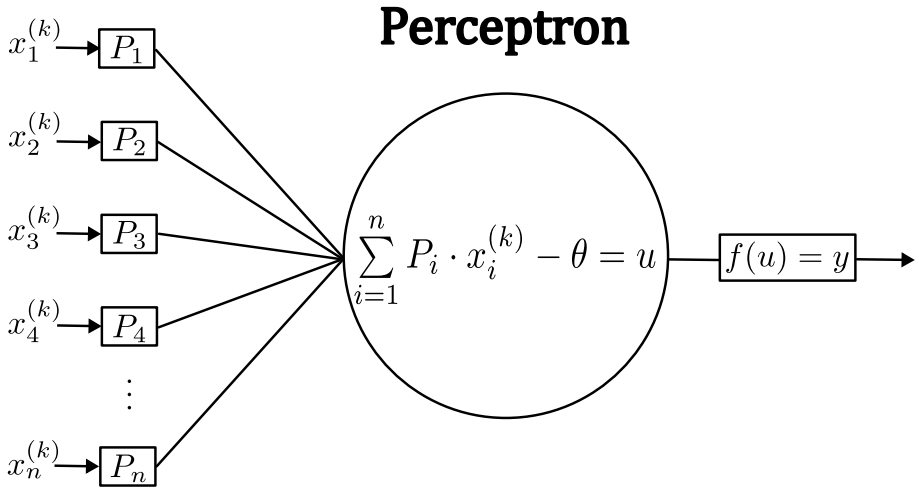


Figura 12 – Representação esquemática em linha do funcionamento do Perceptron. Adaptado de Silva, Spatti e Flauzino (2016).

O tipo de treinamento pelo qual o Perceptron é submetido se enquadra no chamado *treinamento supervisionado*. Segundo Géron (2019, p.8), existem quatro categorias principais de aprendizado: supervisionado, não supervisionado, semissupervisionado e por reforço.

O aprendizado supervisionado consiste em fornecer ao neurônio as saídas desejadas associadas, respectivamente, às entradas que recebe. Estas saídas desejadas também são chamadas de rótulos e serão utilizadas na regra de aprendizagem para, com auxílio da saída calculada pelo neurônio, determinar os ajustes necessários pelo algoritmo de treinamento.

2.1 A regra de aprendizagem do Perceptron

O processo de treinamento do Perceptron foi adaptado da primeira regra de aprendizagem auto-organizada proposta para neurônios biológicos, o postulado, ou regra de Hebb (Haykin, 2001).

Esta regra foi apresentada por Donald Hebb [2] em 1949 e é uma proposta de padrão de aprendizagem em que as modificações realizadas em células nervosas de um agrupamento de neurônios organizado espacialmente se tornassem permanentes por meio da presença, ou ausência, de estímulos sinápticos. Estas modificações seriam, posteriormente, permanentes e se tornariam as regras de funcionamento das conexões das células nervosas em questão.

Ainda de acordo com Haykin (2001), o postulado de Hebb pode ser representado por duas regras, que compõem o que se chama de sinapse hebbiana:

[2] Donald Olding Hebb, nascido em 1904 e falecido em 1985, foi um médico graduado em Fisiologia e Psicologia pela universidade McGill (Tappert, 2019).

- Se dois neurônios em ambos lados de uma sinapse (conexão) são ativados simultaneamente (sincronizadas) então a força daquela sinapse é relativamente incrementada;

- Se dois neurônios em ambos os lados de uma sinapse são ativados assincronamente, então sua conexão é enfraquecida ou eliminada.

No contexto de aprendizado de máquina, a sinapse hebbiana fornece uma estratégia para realização do treinamento de redes neurais de forma que se estabeleça um tipo de ativação, ou ausência de ativação, no neurônio.

Essa estrutura de funcionamento, ao ser aplicada no Perceptron, implica em uma necessidade de treiná-lo com dados característicos de forma que já se saiba previamente quais dados se enquadram em quais categorias, ou seja, devemos fornecer a ele as saídas desejadas para a rede de forma que cada amostra contida nas entradas tenha uma respectiva classificação desejada.

O uso da regra de Hebb adaptada ao contexto das RNA consiste em, se os resultados não coincidirem com as saídas desejadas, os pesos associados e os valores do limiar são incrementados e, se houver uma correspondência, os pesos sinápticos e limiares específicos permanecem inalterados. Este processo todo sempre será finito, ou seja, será concluído com sucesso, se as categorias em questão forem linearmente separáveis (Minsky; Papert, 1988; Haykin, 2001).

A fim de caracterizar um conjunto de dados linearmente separáveis, a seção 2.2 apresenta um estudo de caso em que é possível visualizar um conjunto específico de dados.

2.2 Separabilidade linear: O caso das flores Íris

Para exemplificar conjuntos de dados que possam ou não serem separados linearmente, serão utilizados os dados obtidos em Fisher (1988). Estes dados são muito utilizados para estudos estatísticos e para testar modelos classificadores de dados, que é o caso do Perceptron, e é um dos conjuntos de dados mais bem conhecidos e utilizados no mundo (Unwin; Kleiman, 2021).

Os dados se referem a três variantes da flor do gênero Íris (Figura 13) e incluem comprimento e largura de suas sépalas e pétalas, formando assim um conjunto de dados com cinco atributos para cada amostra, sendo quatro destes numéricos e um de identificação.

No conjunto de dados obtidos em Fisher (1988) há um total de 150 amostras, divididas igualmente entre a Íris Setosa, Íris Versicolor e Íris Virgínica. Utilizando somente a medida das pétalas pode ser elaborado um gráfico de dispersão, observado na Figura 14.

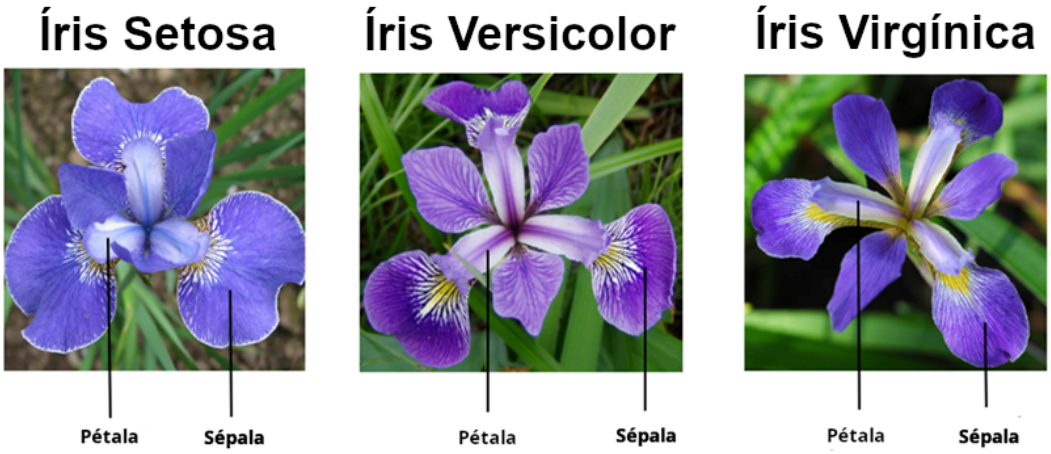


Figura 13 – Os três tipos de flores Íris que foram catalogadas nos dados apresentados em Fisher (1988). Adaptado de https://miro.medium.com/v2/resize:fit:1000/1*Hh53mOF4Xy4eORjLilKOwA.png

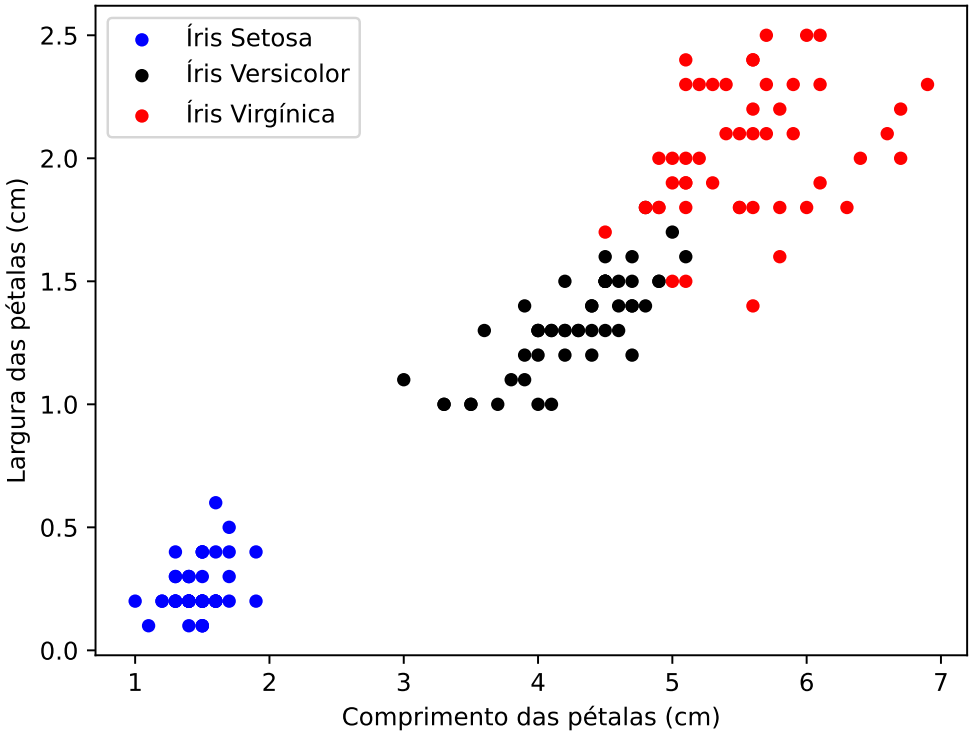


Figura 14 – Gráfico de dispersão da Íris Setosa, Íris Versicolor e Íris Virgínica. Elaborado pelo autor com base nos dados obtidos de Fisher (1988).

Os dados apresentam um aglomerado de informações relativos as espécies Versicolor e Virgínica o que, neste caso, representa uma dificuldade em se tratando da utilização do

Perceptron. Isto ocorre devido a não linearidade necessária para o treinamento da RNA capaz de realizar a catalogação das espécies destes tipos a partir desses dados, ou seja, a fronteira de separação entre estas duas categorias não pode ser expressa por uma reta.

Entretanto, a Figura 14 mostra que a espécie Setosa é linearmente separável das demais. Considerando então as espécies Versicolor e Virgínica como uma única classe, obtém-se duas classes linearmente separáveis: Íris Setosa e Outras Íris, conforme pode ser observado na Figura 15.

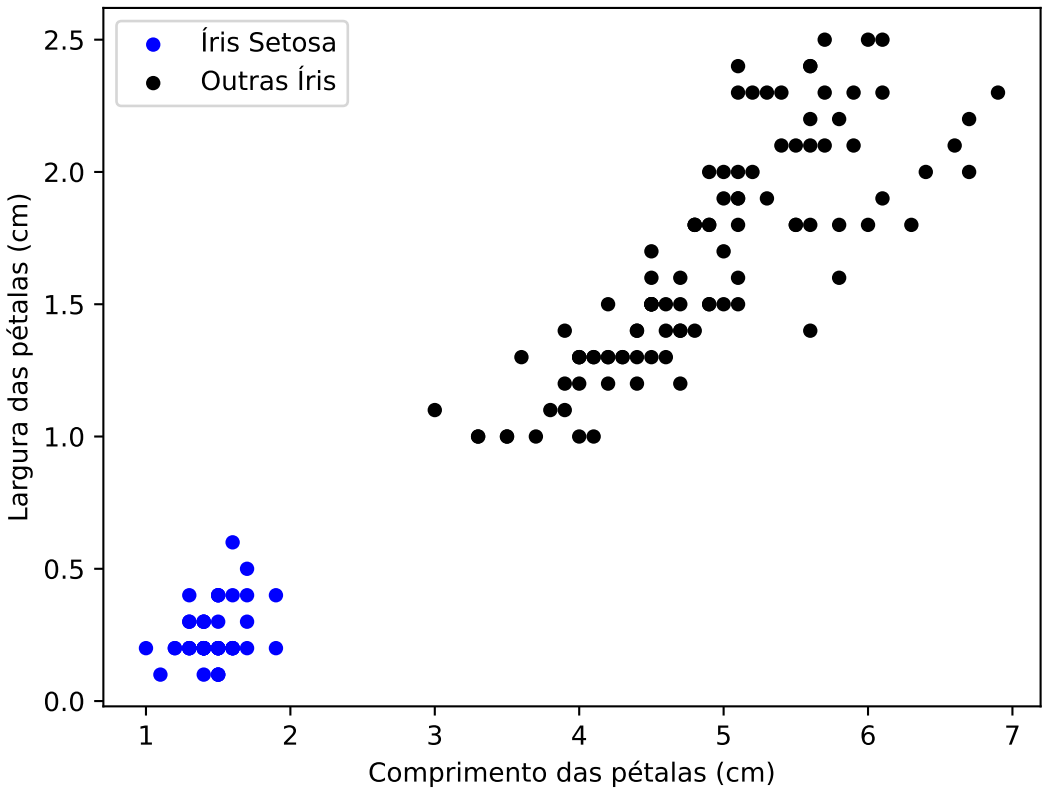


Figura 15 – Gráfico de dispersão da Íris Setosa em relação às outras duas espécies. Elaborado pelo autor.

As Figuras 14 e 15 levam em consideração somente uma das características das flores, no caso, o comprimento e largura das pétalas. Isto permite a construção de uma reta que atue como um separador linear entre duas regiões, como pode ser visto na Figura 16.

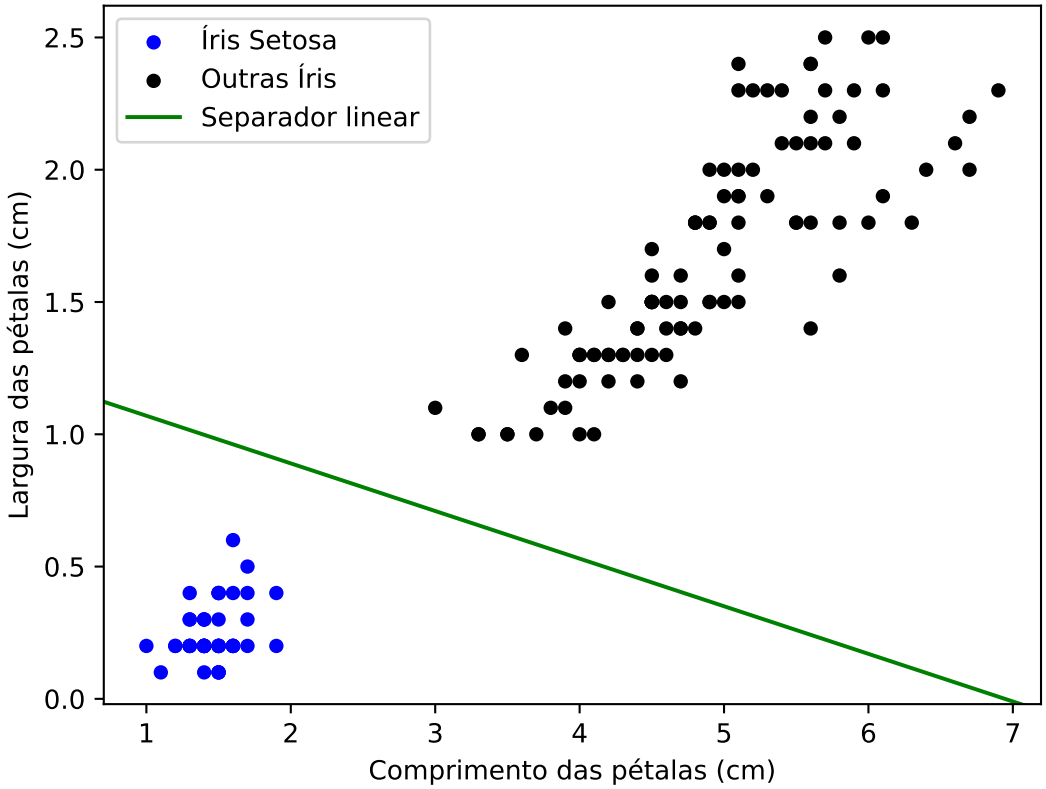


Figura 16 – Reta separando as flores com comprimentos linearmente separáveis. Elaborada pelo autor.

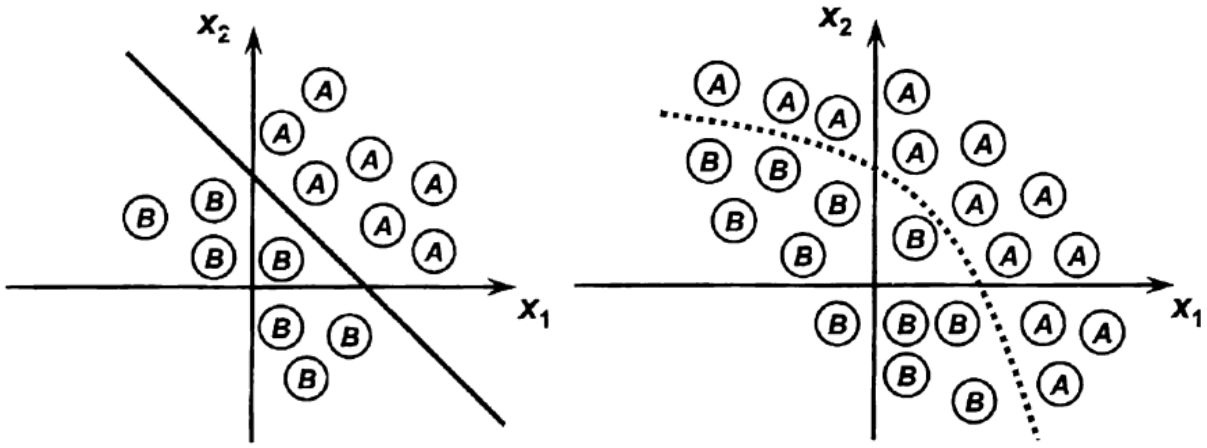
O conjunto de dados das flores apresenta para cada entrada quatro variáveis, as larguras e comprimentos das sépalas e pétalas. A utilização de um número maior de variáveis impossibilita a visualização dos dados da forma como foram coletados e, além disso, se houver uma separabilidade linear, o Perceptron encontrará não uma reta, mas sim um hiperplano.

Uma estratégia para apresentar estes dados é reduzir sua dimensionalidade. Segundo Géron (2019) o algoritmo mais popular é a *Análise dos Componentes Principais* (ACP ou PCA - *Principal Components Analysis*). No capítulo 4, esta técnica será utilizada para apresentar graficamente os dados do treinamento juntamente com este mesmo conjunto de dados das flores.

2.3 Interpretação geométrica do processo de treinamento

Ao realizar a separação dos dados, também é possível atribuir uma interpretação geométrica ao processo de treinamento, assim como para sua conclusão. A Figura 17, extraída de Silva, Spatti e Flauzino (2016), apresenta em um plano cartesiano com dados divididos em duas classes "A" e "B", e dois exemplos de fronteiras. A Figura 17a apresenta

uma possibilidade de fronteira linear enquanto a Figura 17b apresenta a impossibilidade de uma fronteira linear na maneira em que os dados estão dispostos.



(a)
Fronteira linear separando duas classes "A" e "B".

(b)
Exemplo de impossibilidade de fronteira linear entre as classes "A" e "B".

Figura 17 – Fronteiras entre as Classes "A" e "B".
Adaptado de Silva, Spatti e Flauzino (2016, p.62)

Neste caso, o treinamento do Perceptron pode ser interpretado como a busca pelos coeficientes de uma reta capaz de realizar a separação linear entre os dados. Observe que esta é a mesma ideia apresentada no capítulo 1 através do neurônio de McCulloch-Pitts, porém realizada de forma automática. Estendendo para três dimensões, estes coeficientes seriam relativos aos coeficientes de um plano em $\mathbb{R}^3$.

Na Figura 17a pode ser visto um exemplo de separação de dados em duas classes, "A" e "B", de forma que a fronteira de separação entre essas classes é a reta.

Na Figura 17b é possível perceber que não há a possibilidade de ser construída uma fronteira linear de separação entre os dados, isso significa que os dados não são linearmente separáveis e, se houver a tentativa de realizar a separação por meio do algoritmo de treinamento do Perceptron, não haverá convergência (Géron, 2019; Haykin, 2001).

De fato, como pode ser visto em Haykin (2001) e em Minsky e Papert (1988), o Perceptron convergirá, ou seja, encontrará valores para os hiperplanos que separem dois grupos de dados disjuntos se for garantido com antecedência que estes dados são linearmente separáveis. Utilizando o argumento lógico da contrapositiva, podemos utilizar o Perceptron como um verificador de separabilidade linear, pois, se o Perceptron não convergir, isto significa que os dados em questão não são linearmente separáveis.

2.4 O treinamento do Perceptron

Visando facilitar a representação dos conjuntos de dados, pesos e limiares envolvidos no processo de treinamento do Perceptron, estes serão apresentados e representados de forma matricial e, as operações envolvidas em seus cálculos respeitarão as operações associadas à Álgebra Matricial. Caso o leitor tenha alguma dúvida ou questionamento sobre esta álgebra, recomenda-se o estudo deste tópico encontrado em livros didáticos de matemática elementar destinados ao ensino médio como Matemática (2000), Fundamentos de Matemática Elementar (2013) e, para nível superior, Álgebra Linear com aplicações (2012).

Utilizar matrizes para lidar com as etapas do treinamento e uso das RNA entra em acordo com uma estratégia de aprendizagem denominada *aprendizagem em lotes*. Este método consiste em utilizar todos os dados disponíveis para o treinamento, enquanto estabelece um contraste com a *aprendizagem em linha* (ou *online*), em que os dados são fornecidos um a um às RNA e o treinamento ocorre conforme novos dados são adicionados (Géron, 2019).

O treinamento de uma RNA é o processo pelo qual a rede realiza atualizações dos pesos sinápticos, incluindo o limiar, em busca daqueles que realizarão satisfatoriamente a tarefa proposta. Inicialmente, os pesos sinápticos e o limiar são valores reais aleatórios e o processo de treinamento só é finalizado quando os pesos satisfazem determinadas condições impostas ao algoritmo pelo usuário.

O treinamento do Perceptron é organizado em épocas e, cada época se inicia com a atualização dos pesos sinápticos com novos valores. Chamando de P a matriz contendo os pesos sinápticos e de x a matriz contendo as amostras de treinamento, a combinação linear entre os pesos e as variáveis que representam as amostras de treinamento pode ser expressa por:

$$P \cdot x, \tag{2.1}$$

pois, a matriz de entradas x apresenta n linhas, que é o número de variáveis de cada amostra, e m colunas, referente ao número de amostras. Dessa forma, a matriz dos pesos P deve ser de uma matriz linha com n colunas, ou seja,

$$P \cdot x = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & \cdots & P_{1,n} \end{bmatrix} \cdot \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \cdots & x_{n,m} \end{bmatrix}. \tag{2.2}$$

Ainda de acordo com o funcionamento do neurônio de McCulloch-Pitts, faz-se necessário realizar a comparação com o limiar. Como a matriz resultante da Equação (2.2)

possui dimensão $1 \times m$, uma matriz do limiar L pode ser escrita com a mesma dimensão, de modo que todos seus elementos serão iguais à θ, que é o limiar.

Adaptando os processos descritos para linguagem matemática, tem-se:

$$P \cdot x - L = \tag{2.3}$$
$$\begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,n} \end{bmatrix}_{1\times n} \cdot \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{bmatrix}_{n\times m} - \begin{bmatrix} \theta & \theta & \cdots & \theta \end{bmatrix}_{1\times m},$$

que resulta em

$$P \cdot x - L = \tag{2.4}$$
$$\begin{bmatrix} (P_{1,1}x_{1,1} + \cdots + P_{1,n}x_{n,1}) - \theta & \cdots & (P_{1,1}x_{1,m} + \cdots + P_{1,n}x_{n,m}) - \theta \end{bmatrix}_{1\times m}.$$

Em cada elemento da matriz apresentada na Equação (2.4) será realizada a aplicação na função de ativação f. O objetivo da função de ativação é limitar as saídas dos neurônios de forma que sua amplitude seja reduzida a um intervalo finito que tenha significado relativo às entradas (Silva; Spatti; Flauzino, 2016; Haykin, 2001).

Após isto, é obtida uma matriz onde cada um de seus elementos indicará para qual categoria as respectivas variáveis foram classificadas. A função de ativação que é usualmente utilizada com o Perceptron é a função degrau, e portanto, as saídas desta RNA serão discretas e devem ser uma de duas possíveis, ou seja, uma escolha binária.

Esta matriz com saídas obtidas pela RNA será chamada de y e, no caso do treinamento supervisionado, seus respectivos elementos serão comparados, um a um, com os elementos da matriz com as saídas desejadas d.

As operações entre as matrizes, apresentadas na Equação (2.3), podem ser realizadas graças às dimensões compatíveis com o produto matricial. De modo reduzido, tem-se o sistema:

$$\begin{cases} u = P \cdot x - L \\ y = f(u). \end{cases} \tag{2.5}$$

O processo descrito até o presente momento, avalia se os pesos sinápticos estão adequados ao conjunto de dados e suas respectivas saídas esperadas. Este resultado é avaliado através da comparação da matriz y, matriz de classificação dos dados pela rede neural e da matriz d, matriz dos resultados esperados.

Caso haja uma divergência entre estes resultados, os pesos sinápticos devem ser atualizados através da regra de aprendizado de Hebb, descrita matematicamente como:

$$P^{(novo)} = P^{(antigo)} + \eta \cdot (d - y) \cdot x^T \tag{2.6}$$

em que x^T é a matriz transposta de x.

A letra grega η (eta) é chamada de taxa de aprendizagem e é um número entre 0 e 1 responsável por determinar o quanto do erro apresentado pela diferença entre as saídas obtidas pela rede neural e esperadas, $(d - y)$, é incorporado na atualização que os pesos (Kovács, 2023).

Quando ocorre a primeira atualização dos pesos sinápticos considera-se também a ocorrência da primeira época de treinamento. Desta forma, a cada atualização realizada, mais uma época será contabilizada. As épocas, ou seja, os ajustes dos pesos sinápticos, continuam ocorrendo e o processo de treinamento só é finalizado no momento em que os pesos deixam de ser reajustados, ou seja, quando a diferença $(d - y)$ for igual a zero.

Um fluxograma do treinamento pode ser representado como na Figura 18.

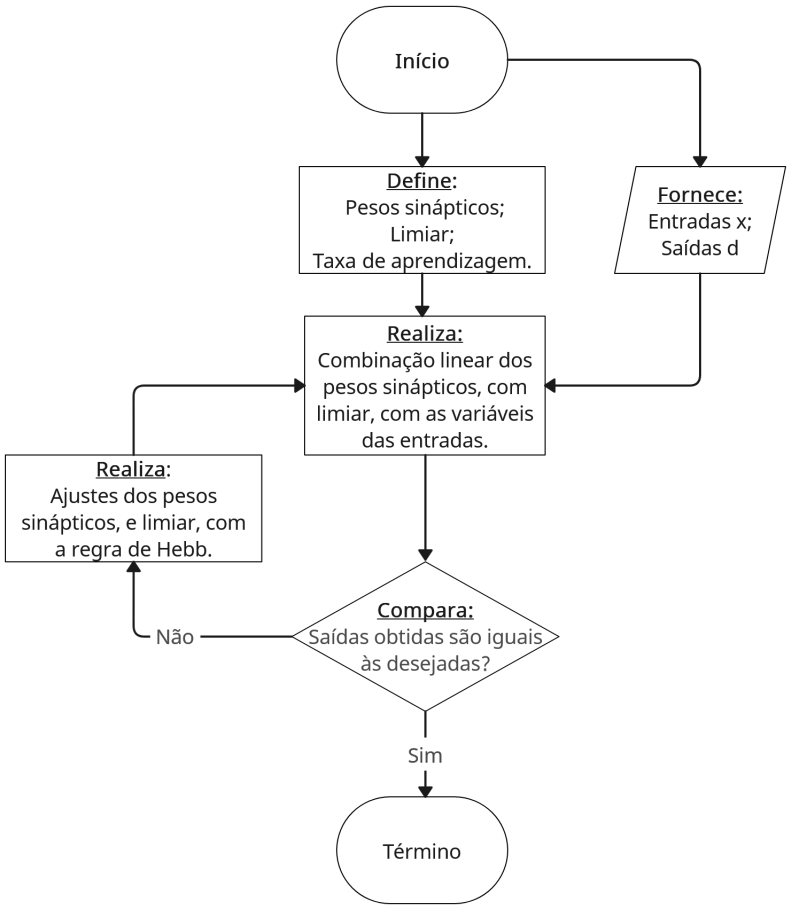


Figura 18 – Fluxograma do treinamento do Perceptron. Elaborado pelo autor.

Será apresentado na seção 2.4.1 um exemplo de treinamento do Perceptron para a

função booleana "E", e nele serão discutidos os detalhes deste ajuste em um caso particular, sem prejuízo a casos gerais.

2.4.1 Implementando a função booleana "E"

A seção 1.2 mostrou que o problema da função booleana "E" é linearmente separável. Além disso, mostrou ser possível determinar, empiricamente, os coeficientes da equação de uma reta que separa os dados.

Este resultado mostra que o Perceptron convergirá, ou seja, apresentará uma solução para este problema após ser adequadamente treinado, na forma de uma matriz de pesos sinápticos que geometricamente podem ser interpretados como os coeficientes da equação de uma reta.

A obtenção destes pesos tem início com a construção da matriz dos dados de entrada x', com os dados tabela verdade apresentada na Tabela 1. Observe que os dados para a construção da matriz x são acondicionados de modo a preservar as operações matriciais definidas nas Equações (2.1) a (2.4).

$$x' = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.7}$$

A matriz das entradas apresentada na Equação (2.7) pode ser modificada, sem que isto interfira nas características das variáveis, adicionando uma linha com números "-1". Esta estratégia faz com que o limiar θ possa ser introduzido na matriz de pesos sinápticos, fazendo com que o seu treinamento ocorra em conjunto.

Realizando-se esta modificação, a matriz x representa o conjunto de parâmetros de entrada para o treinamento da rede neural e, é dada por:

$$x = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix}. \tag{2.8}$$

A adição da linha com "-1" faz com que o limiar θ seja incluído na matriz de pesos de modo a preservar as operações matriciais envolvidas entre as matriz dos pesos P e a matriz dos dados de entrada x.

Matematicamente tem-se:

$$\begin{aligned} u &= P \cdot x - L \\ &= \begin{bmatrix} P_{11} & P_{12} & \theta \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \end{aligned} \tag{2.9}$$

Com o propósito de finalizar este exemplo sem se estender muito no processo e de modo a obter a rápida convergência do Perceptron, considere os pesos sinápticos $0,9$ e o limiar $0,7$ dados por $P = \begin{bmatrix} 0,9 & 0,9 & 0,7 \end{bmatrix}$, que são relativamente próximos aos pesos sinápticos adotados na Tabela 2.

Atualizando a Equação (2.9) tem-se:

$$\begin{aligned} u &= \begin{bmatrix} 0,9 & 0,9 & 0,7 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} -0,7 & 0,2 & 0,2 & 1,1 \end{bmatrix}. \end{aligned} \tag{2.10}$$

A próxima etapa é a utilização da função de ativação para identificar para qual categoria os valores indicam a classificação. Para isto, utiliza-se a função degrau que pode ser expressa como:

$$f(u) = \begin{cases} 1, & \text{se } u \geqslant 0, \\ 0, & \text{se } u < 0 \end{cases}, \tag{2.11}$$

com u sendo cada um dos elementos da matriz obtida na Equação (2.10). Neste caso, considera-se que aplicar uma função em uma matriz é o mesmo que aplicar a função em cada um de seus elementos.

Como resultado tem-se a matriz das saídas obtidas pela rede neural, denominada de y e dada por:

$$\begin{aligned} y &= f(u) \\ &= f\left(\begin{bmatrix} -0,7 & 0,2 & 0,2 & 1,1 \end{bmatrix}\right) \\ &= \begin{bmatrix} f(-0,7) & f(0,2) & f(0,2) & f(1,1) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} \end{aligned} \tag{2.12}$$

Com a matriz y obtida, realiza-se a comparação com a matriz de saídas desejadas d, dada por: $d = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$.

Como há discrepância entre os resultados obtidos pela rede neural e as saídas desejadas, tem-se início uma nova época de treinamento.

Adotando uma taxa de aprendizagem $\eta = 0,1$ e utilizando o algoritmo apresentado na Equação (2.6) para calcular os novos pesos sinápticos, tem-se:

$$P^{(novo)} = \begin{bmatrix} 0,9 & 0,9 & 0,7 \end{bmatrix} + 0,1 \cdot \begin{bmatrix} 0 & -1 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}. \tag{2.13}$$

Realizando as operações apresentadas na Equação (2.13), obtém-se:

$$P^{(novo)} = \begin{bmatrix} 0,8 & 0,8 & 0,9 \end{bmatrix}, \tag{2.14}$$

e, com esta nova matriz de pesos, atualizam-se as Equações (2.10) e (2.12), por:

$$u^{(novo)} = \begin{bmatrix} 0,8 & 0,8 & 0,9 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -0,9 & -0,1 & -0,1 & 0,7 \end{bmatrix} \tag{2.15}$$

e

$$\begin{aligned} y &= f(u) \\ &= f\left(\begin{bmatrix} -0,9 & -0,1 & -0,1 & 0,7 \end{bmatrix}\right) \\ &= \begin{bmatrix} f(-0,9) & f(-0,1) & f(-0,1) & f(0,7) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{2.16}$$

A matriz das saídas calculadas y, obtida na Equação (2.16), é exatamente igual a matriz de saídas desejadas d, fazendo com que o erro obtido pela rede neural em seu treinamento seja zero, implicando em uma igualdade $P^{(novo)} = P^{(antigo)}$, concluindo assim o treinamento do Perceptron.

Utilizando os pesos sinápticos contidos na matriz $P = \begin{bmatrix} 0,8 & 0,8 & 0,9 \end{bmatrix}$ como coeficientes de uma equação de reta $ax + by = c$, respectivamente, tem-se a representação visual da reta que separa os dados das entradas, como pode ser visto na Figura 19.

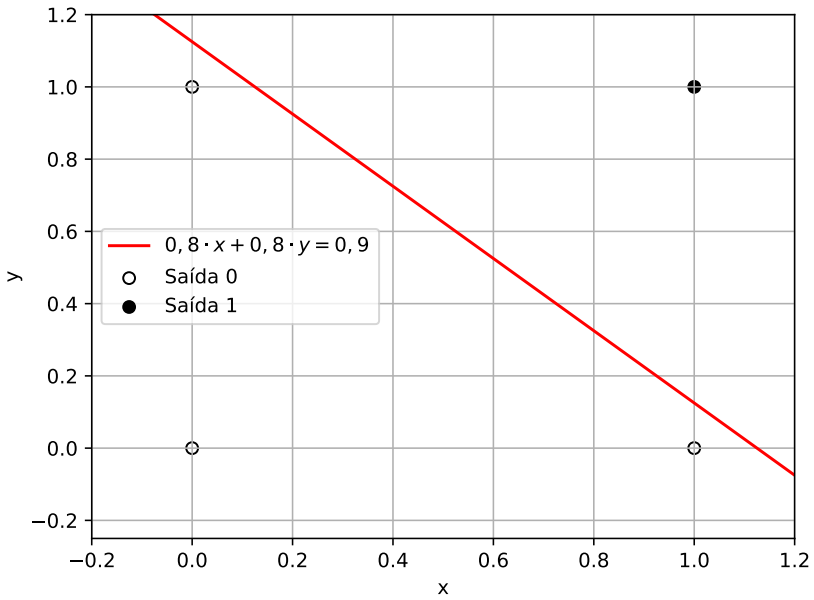


Figura 19 – Gráfico da função booleana "E" com pesos treinados. Elaborado pelo autor.

Este mesmo método pode ser utilizado para treinar a função booleana "OU". Para isto, bastaria ajustar a matriz de saídas desejadas d e repetir o processo para ajuste dos pesos sinápticos.

Uma forma de ajustar os parâmetros de treinamento com diferentes taxas de aprendizagem η, diferentes pesos sinápticos iniciais e observar as mudanças que ocorrem durante o processo de treinamento é por meio do uso da implementação do algoritmo de treinamento em alguma linguagem de programação.

2.5 Utilizando Python para treinar um Perceptron

Os códigos de programação aqui apresentados podem ser salvos em um arquivo com extensão `.py` e executados ou inseridos em um caderno no Google Colab ou no Jupyter Notebook.

Para implementar o treinamento realizado na seção 2.4.1, inicialmente faz-se a importação da biblioteca `numpy`, segundo o Algoritmo 1, a qual é responsável por fornecer as ferramentas que permitem realizar as operações de matriciais de produto e transposta, além do cálculo da função de ativação.

```
#Importa a biblioteca
import numpy as np
```

Algoritmo 1 – Importando a biblioteca `numpy`. Elaborado pelo autor.

De posse das funções matemáticas necessárias, implementa-se uma função chamada de `Perceptron` que receberá como parâmetros de entrada η, x, d e P. Esta função será responsável por realizar o treinamento do Perceptron com base nos parâmetros de entrada, atualizando os pesos sinápticos a cada época de treinamento. Esta função pode ser observada no Algoritmo 2.

```
#Criação da função Perceptron:
def Perceptron(P,x,d,eta):
# Estabelece como 0 o número de épocas:
epocas = 0
#Realiza a primeira combinação linear entre P e x:
u = np.dot(P,x)
#Realiza a aplicação da função de ativação:
y = np.where(u>=0,1.0,0.)
#Enquanto y for diferente de d::
while np.array_equal(y,d)==False:
    #Adiciona uma época:
    epocas = epocas + 1
    #Compara as matrizes y e d:
    diferenca_saida = d-y
    #Realiza o ajuste dos pesos:
    P += eta*(np.dot(diferenca_saida,x.T))
    #Nova combinação linear com os novos pesos:
    u = np.dot(P,x)
    #Apresenta a nova matriz y:
    y = np.where(u>=0,1,0)
    print(f'Na época {epocas}, a matriz dos Pesos é {P}')
print('A matriz dos pesos completamente treinada é:\n',P)
print('Número de épocas:',epocas)
```

Algoritmo 2 – Construção da função `Perceptron`. Elaborado pelo autor.

Para que a função Perceptron realize os ajustes dos pesos sinápticos, os parâmetros desta função devem ser carregados no ambiente através dos comandos contidos do Algoritmo 3.

```
#Insere a matriz de entradas x:
x = np.array([ [0, 1, 0, 1],[0, 0, 1, 1],[-1, -1, -1, -1]])
#Saídas desejadas d
d = np.array([[0, 0, 0, 1]])
print(f'A matriz das entradas é:\n{x}')
print(f'A matriz de saídas desejadas é:\n{d}')
```

Algoritmo 3 – Fornecimento dos dados das amostras para treinamento com a função `Perceptron`. Elaborado pelo autor.

Após configurar todos estes parâmetros, pode-se utilizar os pesos sinápticos obtidos da implementação da função booleana “E”, na seção 2.4.1, e então utilizar a função criada no Algoritmo 4 para iniciar o treinamento.

```
Perceptron([0.9, 0.9, 0.7],x,d,0.1])
```

Algoritmo 4 – Matriz de pesos sinápticos. Elaborado pelo autor.

Caso o usuário tenha interesse em gerar novos pesos sinápticos de forma aleatória e automaticamente, basta utilizar a função `random.rand`, da biblioteca `numpy`, e obter pesos iniciais aleatórios compatíveis com a matriz de entradas x com o uso de `len(x)`. Desta forma, são atribuídos pesos sinápticos iniciais aleatórios para a matriz de pesos sinápticos P ao utilizar a linha de comando contida no Algoritmo 5.

```
# len(x) é o número de colunas de x
P = np.random.rand(1,len(x))
print(f'A matriz dos pesos é:\n{P}')
```

Algoritmo 5 – Geração de pesos sinápticos aleatórios de acordo com a quantidade de variáveis das entradas. Elaborado pelo autor.

e, em seguida, executar novamente a função Perceptron com a sintaxe dada como no Algoritmo 6, com η sendo um valor preestabelecido entre 0 e 1.

```
Perceptron(P,x,d,eta)
```

Algoritmo 6 – Função `Perceptron` com parâmetros inseridos. Elaborado pelo autor.

Finalizado o treinamento, inicia-se a fase de teste, ou seja, da verificação se a rede neural aprendeu a identificar corretamente o padrão aprendido durante a fase de treinamento.

O uso da linguagem de programação apresenta uma vantagem que facilita a compreensão de uma característica do Perceptron que pode ser uma fragilidade ou potencialidade, a depender da situação. Se uma busca rápida por pesos sinápticos que satisfaçam as saídas desejadas for o suficiente, é possível perceber como o Perceptron apresenta uma convergência mais rápida do que outras redes, resultado que será discutido futuramente neste trabalho.

Isto ocorre pois o Perceptron não busca os pesos sinápticos que melhor atendam às condições da função, mas sim, finaliza o treinamento com quaisquer pesos que atendam o critério de conclusão do treinamento. Este fato é consequência do uso da função de ativação durante o processo de treinamento.

Além, disso, conforme os pesos são alterados com o uso da função para gerar valores aleatórios `random.rand`, é possível visualizar que o Perceptron apresenta diferentes resultados para os pesos sinápticos em P ao fim de seu treinamento.

Relembrando os resultados já obtidos neste trabalho, observa-se que os pesos sinápticos obtidos empiricamente na seção 1.2 e utilizados no exemplo referente a função booleana "E" na Tabela 2 são diferentes daqueles que foram apresentados na Equação (2.14) mas ambos satisfazem as condições de conclusão do treinamento do Perceptron, ou seja, ambas matrizes de pesos sinápticos $\begin{bmatrix} 0,5 & 0,5 & 1,5 \end{bmatrix}$ e $\begin{bmatrix} 0,8 & 0,8 & 0,9 \end{bmatrix}$ realizam a categorização de forma efetiva.

2.6 Sistemas lineares e o Perceptron

É possível estabelecer uma relação entre o funcionamento do Perceptron e o estudo de sistemas lineares. Tomando a matriz apresentada na Equação (2.9) igual a zero, tem-se um exemplo de sistema linear em que as incógnitas são os pesos sinápticos P_{11}, P_{12} e o limiar θ. Nestes sistemas, busca-se encontrar para as incógnitas uma solução, um conjunto de valores numéricos, de forma que todas as equações se tornem sentenças verdadeiras simultaneamente.

De modo mais geral, chamamos de equações lineares nas incógnitas $P_1, \cdots, P_n$, todas as equações do tipo $P_1 \cdot x_{1,1} + P_2 \cdot x_{2,1} + \cdots + P_n \cdot x_{n,1} = b$.

Todos os números $x_{1,1}, x_{2,1}, \cdots, x_{n,1}$ são valores reais e, neste caso, são as variáveis das amostras contidas na matriz de entradas. O coeficiente b é chamado de termo independente e, neste caso é o limiar θ.

Quando um conjunto com mais de uma equação é estabelecido, havendo nestas equações incógnitas em comum, determina-se um sistema linear de equações lineares, como visto na Equação (2.17) que possui n incógnitas e m equações:

$$\begin{cases} P_{1,1}x_{1,1} + P_{1,2}x_{1,2} + \cdots + P_{1,n}x_{1,n} = b_1, \\ P_{1,1}x_{2,1} + P_{1,2}x_{2,2} + \cdots + P_{1,n}x_{2,n} = b_2, \\ \vdots \\ P_{1,1}x_{m,1} + P_{1,2}x_{m,2} + \cdots + P_{1,n}x_{m,n} = b_m \end{cases} . \tag{2.17}$$

Quanto à obtenção de soluções de sistemas lineares, de modo geral, eles se enquadram em um de três tipos: sistemas possíveis e determinados (SPD), sistemas possíveis e indeterminados (SPI) e sistemas impossíveis (SI). É possível interpretar estes resultados geometricamente para até três dimensões.

No caso bidimensional, cada equação linear pode ser interpretada como a equação de uma reta no plano cartesiano e, este sistema linear apresentará solução única quando

estas retas se interceptarem. Este é o caso dos sistemas tipo SPD.

Neste tipo de sistema linear, as equações envolvidas no sistema são linearmente independentes entre si, ou seja, não podem ser transformadas em alguma outra dentre elas apenas com operações de soma e multiplicação por escalar e, além disso, apresentam um número de incógnitas igual ao número de equações. Um exemplo de um SPD é formado pelas equações na Equação (2.18), cujo conjunto solução é $S = \{(12, 8)\}$.

$$\begin{cases} x + y = 20 \\ x - y = 4 \end{cases} . \tag{2.18}$$

Quando há um número menor de equações linearmente independentes do que de incógnitas, tem-se um sistema do tipo SPI. Estes sistemas apresentam infinitas soluções e, seu conjunto solução é representado por sentenças algébricas em função de alguma incógnita que neste caso se torna uma variável. Este é o caso do sistema linear apresentado na Equação (2.19), que tem seu conjunto solução representado por: $S = \{(x, 20 - x)$, para qualquer $x \in \mathbb{R}\}$.

$$\begin{cases} x + y = 20 \end{cases} . \tag{2.19}$$

Quando em um sistema linear suas equações apresentam sentenças que se contradizem, ele é dito do tipo impossível ou simplesmente SI, como é o caso do sistema apresentado na Equação (2.20). Estes sistemas também são chamados de inconsistentes.

$$\begin{cases} 2x + 3y = 5 \\ 4x + 6y = 12 \end{cases} . \tag{2.20}$$

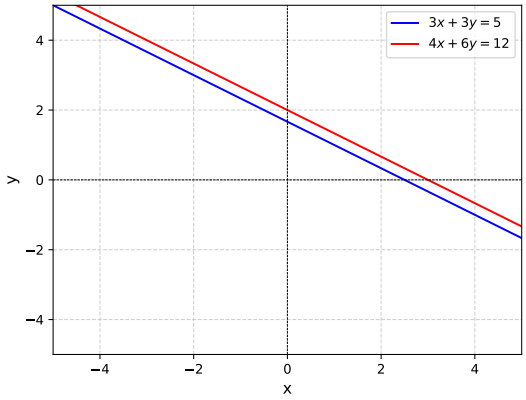
No sistema da Equação (2.20), a segunda linha pode ser dividida por 2 e transformada na primeira, de modo a obter a Equação (2.21), resultando em um absurdo, pois, pela igualdade obtida, apresenta-se $5 = 6$.

$$\begin{cases} 2x + 3y = 5 \\ 2x + 3y = 6 \end{cases} . \tag{2.21}$$

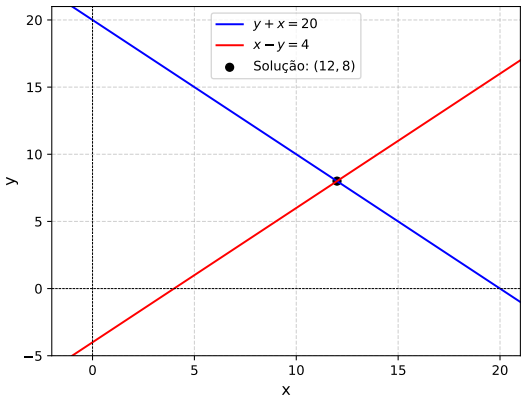
A Figura 20 apresenta as representações geométricas dos sistemas de tipo SI, SPD e SPI, com as Equações (2.20), (2.18) e (2.19) com as Figuras 20a, 20b e 20c, respectivamente.

As intersecções das retas na Figura 20 indica visualmente o número de soluções, para os sistemas SI e SPD, e a ausência de outra reta presente no gráfico significa um número indeterminado de soluções, como nos sistemas de tipo SPI.

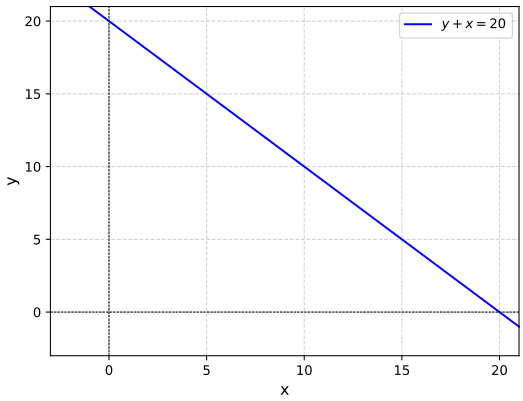
Os sistemas lineares relacionados ao Perceptron são chamados de **Sobredeterminados**, o que significa que há um número maior de equações do que incógnitas.



(a) Sistema Impossível



(b) Sistema Possível e Determinado.



(c) Sistema Possível e Indeterminado.

Figura 20 – Tipos de sistemas lineares. Elaborado pelo autor.

Há uma condição para que os sistemas Sobredeterminados possuam solução. Este é o caso em que no sistema linear existe um número de equações linearmente dependentes igual à diferença entre o número total de equações e o número de incógnitas das equações. Em outras palavras, em um sistema linear Sobredeterminado com duas variáveis, por exemplo, se houverem duas equações que possam ser utilizadas para gerar todas as outras, então este sistema possuirá solução.

A Equação (2.22) apresenta um caso de sistema Sobredeterminado com solução. Com o uso das equações, é possível perceber que, duas vezes a segunda equação, menos a primeira, resulta na terceira equação. O gráfico apresentando as três retas formadas pelas equações de (2.22) pode ser visto na Figura 21.

$$\begin{cases} x + y = 1 \\ 2x + y = 2 \\ 3x + y = 3 \end{cases} . \tag{2.22}$$

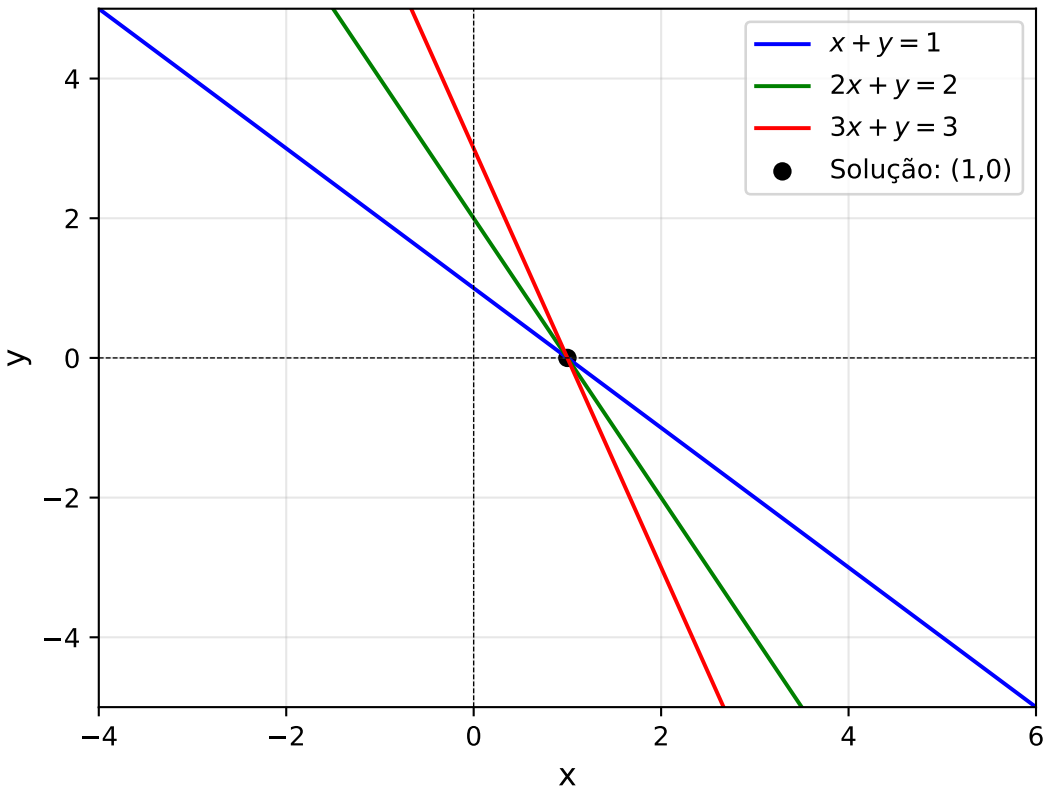


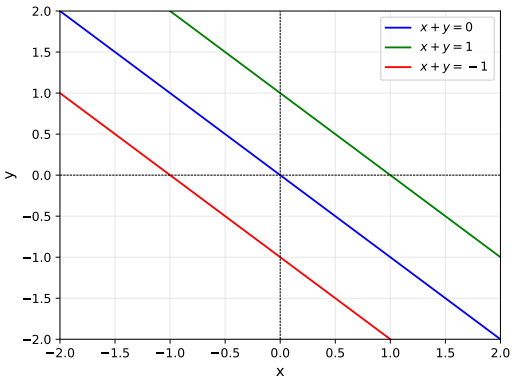
Figura 21 – Sistema Sobredeterminado com solução. Elaborado pelo autor.

Para sistemas Sobredeterminados que não possuem solução, existem duas possibilidades que podem ser exemplificadas com retas obtidas individualmente das equações do sistema como na Figura 22.

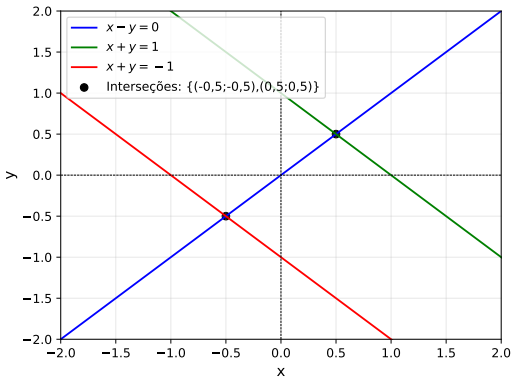
O primeiro é o caso em que todas as equações são inconsistentes e nenhuma delas se satisfazem. Este caso pode ser representado por retas que são todas paralelas, ou seja, não há interseção para nenhuma das equações, como na Figura 22a

O segundo caso é em que algumas equações satisfazem umas das outras, mas o restante não satisfaz estas condições, podendo satisfazer outras condições de outro grupo de equações. Este caso é representado pelas Figuras 22b e 22c, mas não está limitado a estas possibilidades. Um exemplo deste caso pode ser visto pela Equação (2.23), que foi representado graficamente na Figura 22c.

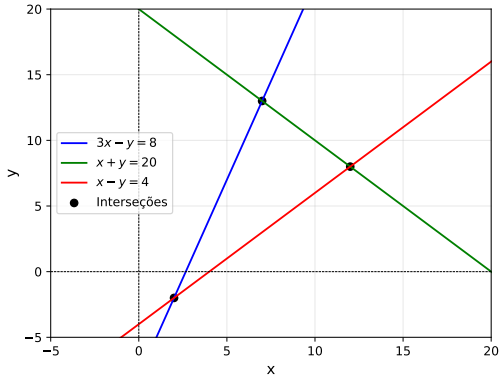
$$\begin{cases} 3x - y = 8 \\ x + y = 20 \\ x - y = 4 \end{cases} \tag{2.23}$$



(a) Todas as retas paralelas.



(b) Duas retas paralelas e duas interseções.



(c) Três interseções.

Figura 22 – Casos de sistemas Sobredeterminados. Elaborados pelo autor.

As equações apresentadas na Equação (2.23), se analisadas separadamente, duas a duas, possuem solução. A primeira e a segunda tem solução em $(7, 13)$, a segunda e a terceira em $(12, 8)$ e a primeira e terceira em $(2, -2)$. Porém, o sistema como um todo, da forma que é apresentado com as três equações simultaneamente, não possui solução.

De modo geral, ainda em duas dimensões, há uma infinidade de retas que podem se interseccionar sem apresentarem um ponto em comum a todas, e portanto há uma infinidade de equações que podem ser consistentes porém não satisfazerem a uma ou mais equações restantes de um sistema ao qual pertencem.

Mas, a busca por soluções de sistemas Sobredeterminados não é o que o objetivo do Perceptron, ou seja, os coeficientes que o Perceptron busca não são para as retas que passam pelos pontos que as variáveis dos dados representam, e sim a busca por pesos sinápticos que produzam, nas equações, sentenças que apresentem valores que estejam, comparativamente, maiores ou menores que o limiar θ.

Isto implica na possibilidade de mudar o sistema de equações para um sistema de inequações lineares, em que todas as inequações podem ser comparadas a zero, ou seja, do ponto de vista do treinamento do Perceptron, a Equação (2.17) poderia ser transformado,

a depender do exemplo, em um sistema linear como na Equação (2.24):

$$\begin{cases} P_{1,1} \cdot x_{1,1} + P_{1,2} \cdot x_{2,1} + \cdots + P_{1,n} \cdot x_{n,1} - \theta > 0 \\ P_{1,1} \cdot x_{1,2} + P_{1,2} \cdot x_{2,2} + \cdots + P_{1,n} \cdot x_{n,2} - \theta < 0 \\ \vdots \\ P_{1,1} \cdot x_{1,n} + P_{1,2} \cdot x_{2,n} + \cdots + P_{1,n} \cdot x_{n,n} - \theta > 0 \end{cases} . \tag{2.24}$$

O sistema de inequações lineares apresentado na Equação (2.24) evidencia a possibilidade de haver um conjunto de valores para os quais os pesos sinápticos satisfaçam as inequações. De fato, dada a separabilidade linear dos dados referentes às amostras, o teorema da convergência do Perceptron garante a existência destes pesos sinápticos.

A literatura apresenta diferentes métodos numéricos para determinar as soluções de sistemas Sobredeterminados, caso estas existam (Franco, 2006; Ruggiero; Rocha Lopes, 1996; Arenales; Darezzo, 2016). De um modo geral, a obtenção de soluções aproximadas para estes sistemas, sejam eles inconsistentes ou não, podem ser fornecidas com o auxílio do Método dos Mínimos Quadrados (MMQ).

Considerando uma matriz P como matriz das incógnitas, a matriz x como matriz dos coeficientes e a matriz y com os termos independentes, o MMQ apresenta soluções para o sistema $P \cdot x = y$ com o uso da Equação (2.25) (Géron, 2019).

$$P = \left[\left(x^T \cdot x \right)^{-1} \cdot x^T \right] \cdot y, \tag{2.25}$$

em que $\left(x^T \cdot x \right)^{-1}$ representa a matriz inversa resultante do produto de x^T, a matriz transposta de x, pela própria matriz x.

De acordo com Géron (2019), o MMQ pode ser encarado como um método de minimizar o custo associado a uma regressão linear que realizaria uma separação entre os dados, mas para um grande volume de dados este método demanda a análise de um número elevado de variáveis, podendo se tornar lento, principalmente se comparado ao método de aprendizado do Adaline, assunto do capítulo 3, que realiza uma regressão linear eficiente quando comparado ao Perceptron e ao MMQ.

3 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 23, em comparação com a Figura 12 apresentada no capítulo anterior.

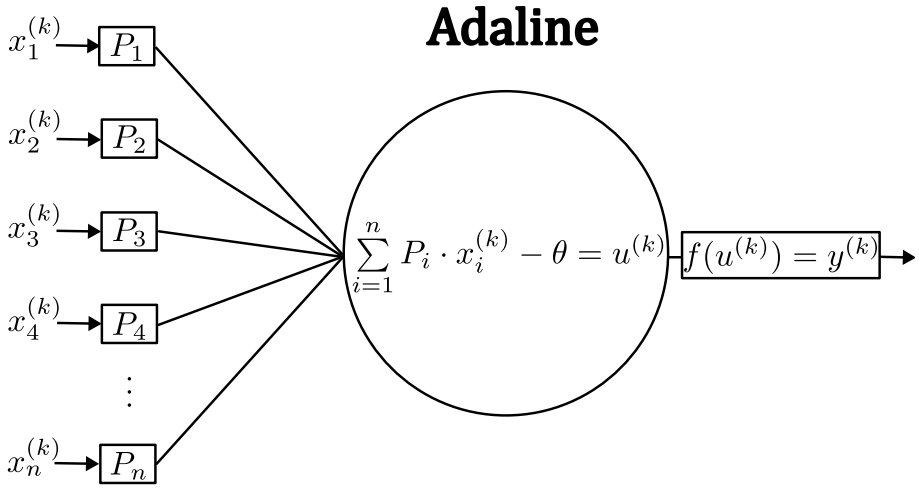


Figura 23 – Representação esquemática do Adaline. Elaborado pelo autor.

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).

3.1 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 (3.1):

$$\begin{aligned} E(P) &= \frac{1}{2}\sum_{k=1}^{m}(d^{(k)} - u^{(k)})^2 \\ &= \frac{1}{2}\sum_{k=1}^{m}\left[d^{(k)} - \left(\sum_{i=1}^{n} P_i \cdot x_i^{(k)} - \theta\right)\right]^2 . \end{aligned} \tag{3.1}$$

Na Equação (3.1): 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 (3.1) 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 (3.1) possa ser reescrita sem um dos somatórios como:

$$\begin{aligned} E(P) &= \frac{1}{2}\sum_{k=1}^{m}(d^{(k)} - u^{(k)})^2 \\ &= \frac{1}{2}\sum_{k=1}^{m}\left[d^{(k)} - \left(P \cdot x^{(k)} - \theta\right)\right]^2. \end{aligned} \tag{3.2}$$

A Equação (3.2) 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 (3.2), 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 (3.3):

$$P^{(novo)} = P^{(antigo)} + \eta \cdot \nabla E, \tag{3.3}$$

em que η é a taxa de aprendizagem do algoritmo.

O fator de correção ∇E contido no algoritmo de aprendizado apresentado na (3.3) pode ser apresentado explicitamente ao aplicar-se o gradiente do Erro Quadrático na Equação (3.2). 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 (3.4).

$$\begin{aligned}
\nabla E &= \frac{\partial E}{\partial P} \\
&= \frac{\partial E}{\partial u}\frac{\partial u}{\partial P} \\
&= \frac{\partial}{\partial u}\left[\frac{1}{2}\sum_{k=1}^{m}\left(d^{(k)}-u^{(k)}\right)^2\right]\frac{\partial}{\partial P}\left(d^{(k)}-u^{(k)}\right) \\
&= \sum_{k=1}^{m}\left(d^{(k)}-u^{(k)}\right)\frac{\partial}{\partial P}\left[d^{(k)}-\left(P\cdot x^{(k)}-\theta\right)\right] \\
&= \sum_{k=1}^{m}\left(d^{(k)}-u^{(k)}\right)\cdot\left(-x^{(k)}\right)^T. \qquad (3.4)
\end{aligned}$$

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 (3.4).

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 (3.4). Levando-se este fato em consideração e substituindo-o em (3.3) tem-se:

$$P^{(novo)} = P^{(antigo)} + \eta \cdot \sum_{k=1}^{m}(d^{(k)} - u^{(k)}) \cdot (x^{(k)})^T. \qquad (3.5)$$

A Equação (3.4) 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.

3.2 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 (3.6):

$$EQM(P) = \frac{1}{m}\sum_{k=1}^{m}\left(d^{(k)} - u^{(k)}\right)^2. \tag{3.6}$$

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 24 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:

$$\left|EQM^{(anterior)} - EQM^{(atual)}\right| < \varepsilon. \tag{3.7}$$

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 25.

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

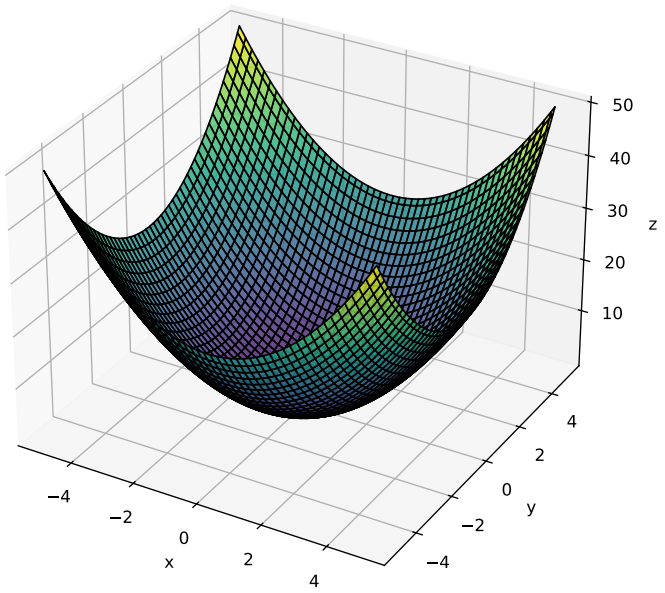


Figura 24 – Superfície de uma função Erro Quadrado Médio usual de um combinador linear. Adaptado de Widrow e Lehr (1990, p.1428).

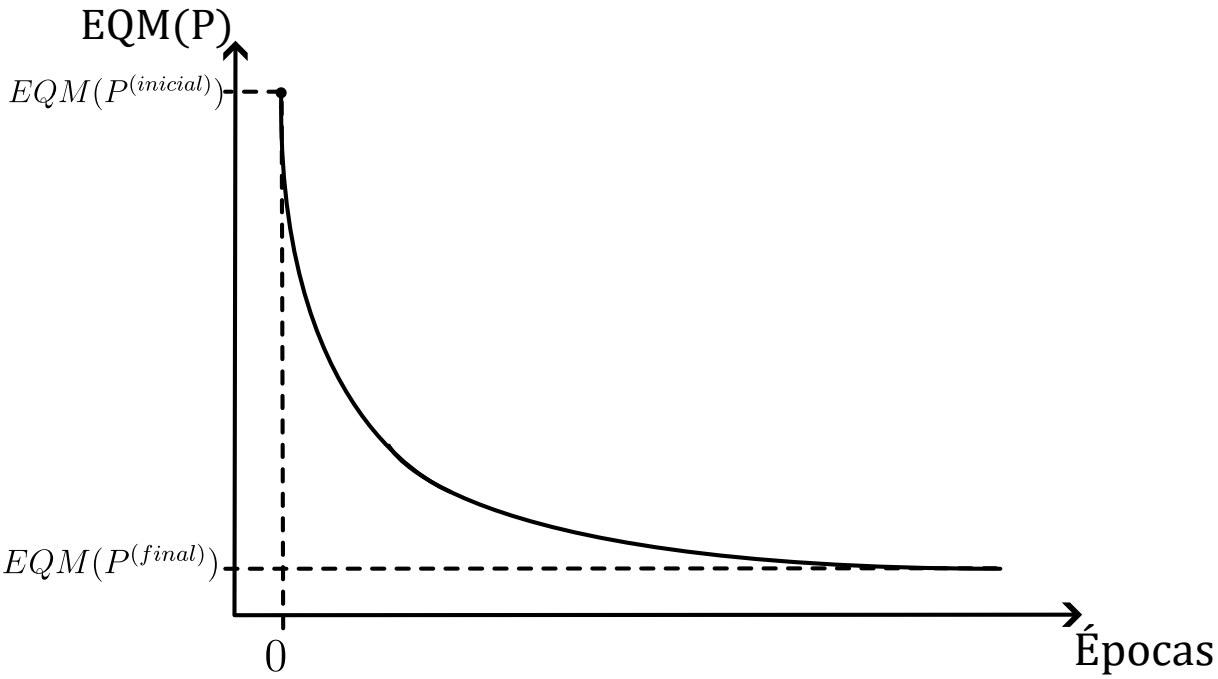


Figura 25 – Convergência da função Erro Quadrático Médio. Adaptado de Silva, Spatti e Flauzino (2016, p.83).

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 26.

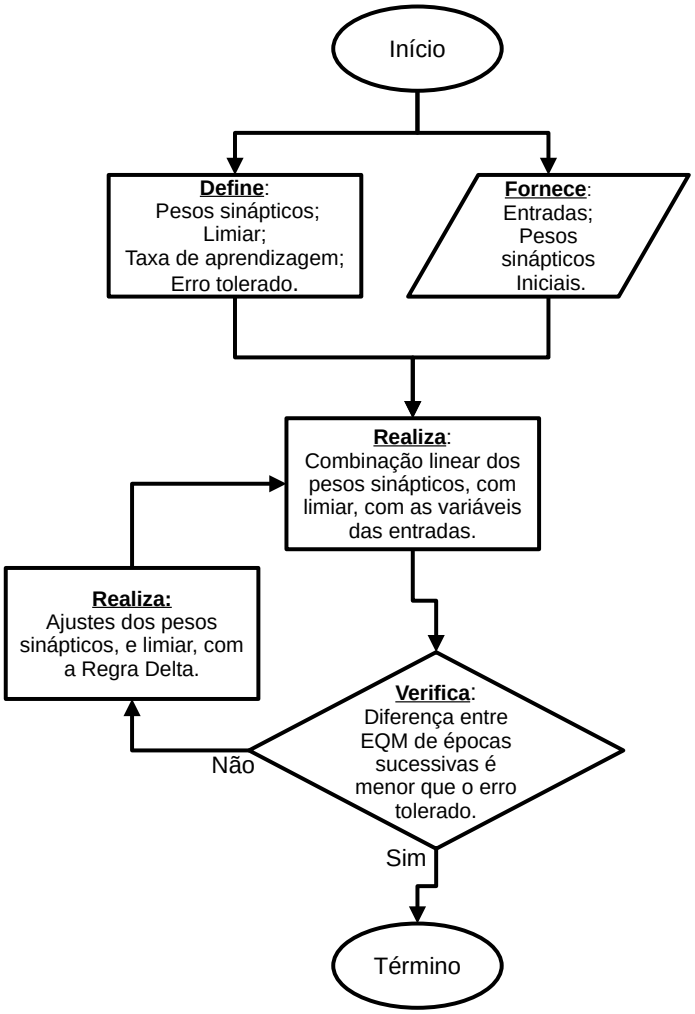


Figura 26 – Fluxograma apresentando as etapas do processo de treinamento do Adaline. Elaborado pelo autor.

3.3 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 27 apresenta o diagrama de dispersão deste caso.

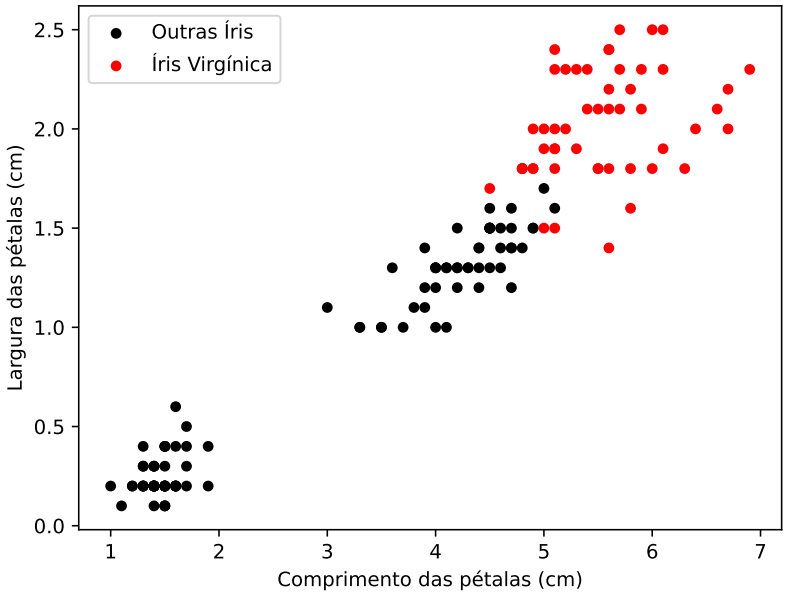


Figura 27 – Os dados dos comprimentos das pétalas das três flores Íris. Elaborado pelo autor.

Conforme pode ser observado pela Figura 27, 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 28.

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.

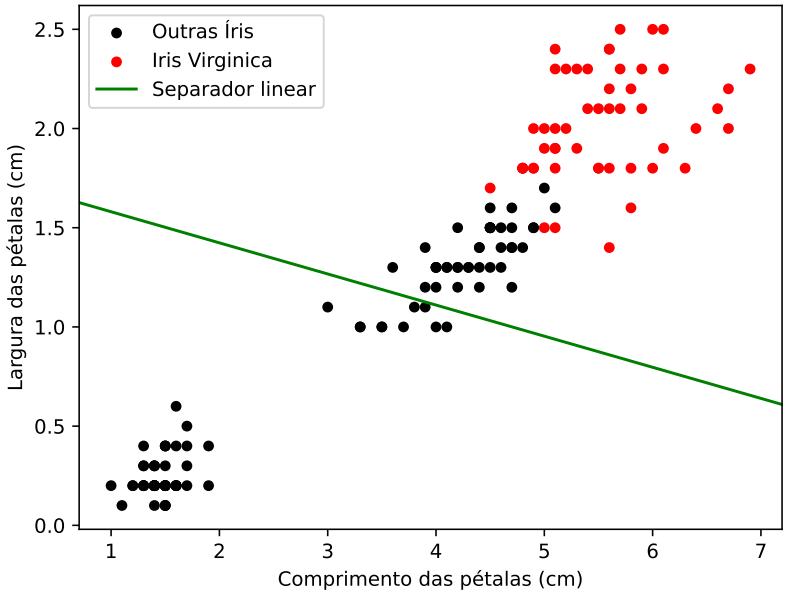


Figura 28 – Possibilidade de reta apresentada pelo Perceptron em dados não separáveis linearmente. Elaborada pelo autor.

Um exemplo desta vantagem do treinamento do Adaline frente ao do Perceptron pode ser visto na Figura 29, 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.

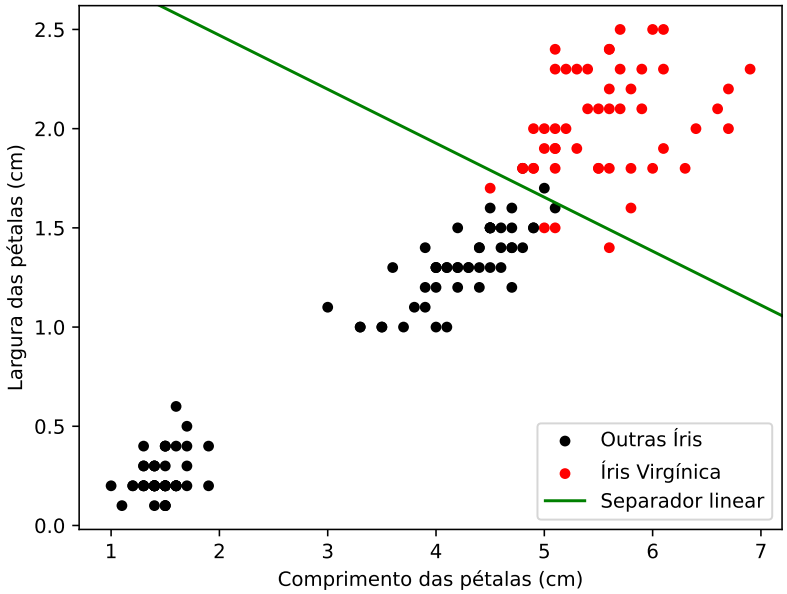


Figura 29 – Possibilidade de reta apresentada pelo Adaline em dados não separáveis linearmente. Elaborado pelo autor.

3.4 Implementando o Adaline em Python

O Algoritmo 7 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 7 – 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 1.2 e por meio da Figura 7, 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.

$$x = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \tag{3.8}$$

A disposição dos parâmetros de entrada da função Adaline é ilustrado no Algoritmo 8 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 8 – Fornecimento das entradas, saídas desejadas e pesos iniciais.
Elaborada pelo autor

A matriz de pesos P fornecida no Algoritmo 8 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 7. O Algoritmo 9, apresenta a sintaxe para este treinamento.

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

Algoritmo 9 – Função Adaline com parâmetros definidos. Elaborado pelo autor.

Ao executar a linha de código fornecida no Algoritmo 9, 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 30.

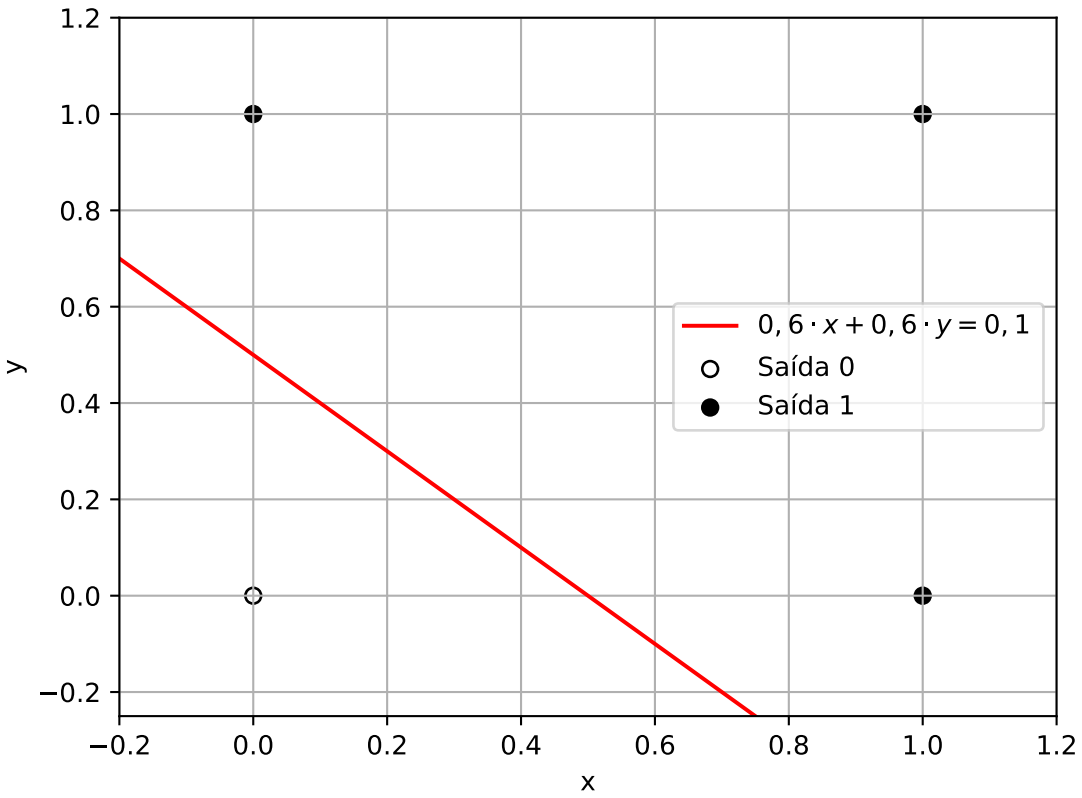


Figura 30 – Função booleana "OU" com reta obtida com treinamento. Elaborada pelo autor.

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 10 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 31.

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

Algoritmo 10 – Função Adaline com parâmetros definidos novamente. Elaborado pelo autor.

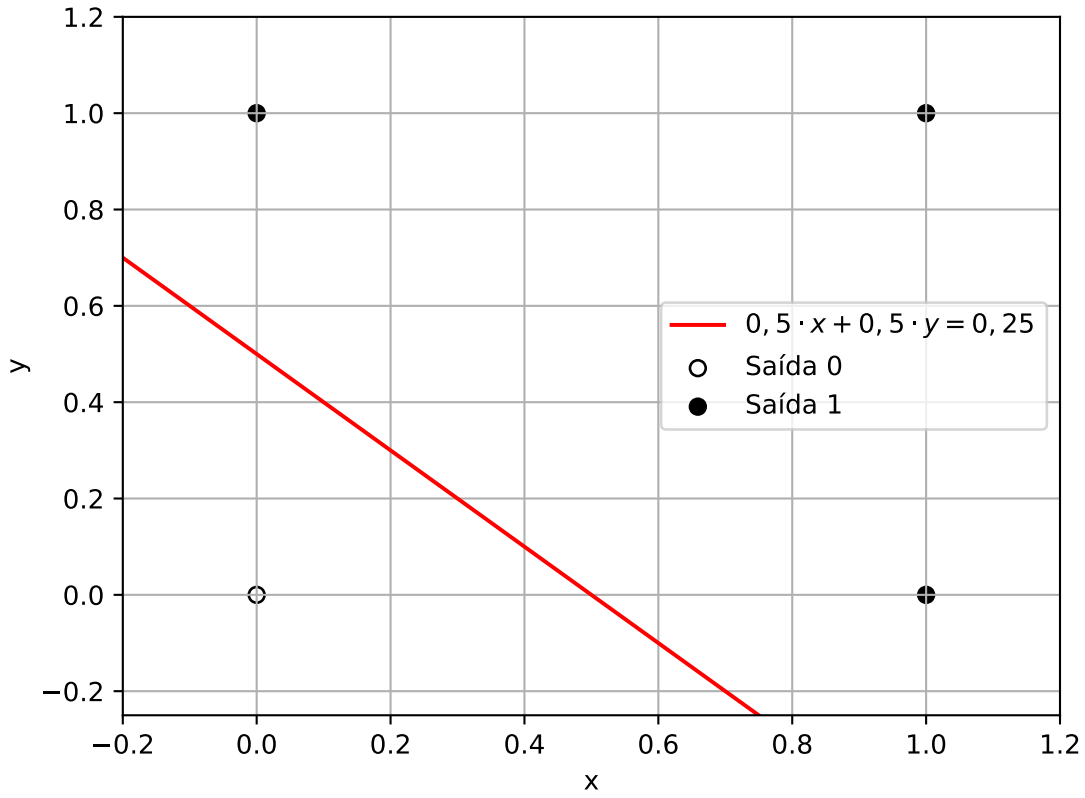


Figura 31 – Função booleana "OU" com coeficientes otimizados. Elaborado pelo autor.

A reta da Figura 31 está melhor posicionada devido ao menor erro ε admitido em relação à reta apresentada pela Figura 30, que possui um valor para ε relativamente maior. Isto implica no fato dos pesos sinápticos obtidos no treinamento realizado com o Algoritmo 10 estarem mais aptos a generalizar este Adaline para novos dados do que os pesos obtidos com o treinamento com o Algoritmo 9.

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 4 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.*

4 Perceptron Multicamadas

A composição de Perceptrons, que forma uma RNA chamada de Perceptron Multicamadas (PMC), contorna limitações que o Perceptron unitário possui em relação às condições que precisam ser satisfeitas para que ele possa ser utilizado eficientemente e às possibilidades de separação de dados que pode realizar (Bottou, 2017; Minsky; Papert, 1988).

Quando é estabelecida uma conexão direta entre a saída de um neurônio com a entrada de outro, criam-se camadas. Ao se utilizar redes neurais de várias camadas são introduzidos novos algoritmos de treinamento, cada um com seus respectivos potenciais e limitações.

Na Figura 32 é apresentado um exemplo de rede PMC. As linhas indicam as direções que os dados seguem, podendo alternar se os sentidos, da esquerda para a direita, ou o contrário, de acordo com a etapa do treinamento em questão. Os círculos são os neurônios e já na primeira camada, em vermelho, cada um dos três neurônios estão recebendo todas as variáveis x_1, x_2 e x_3 da amostra em questão.

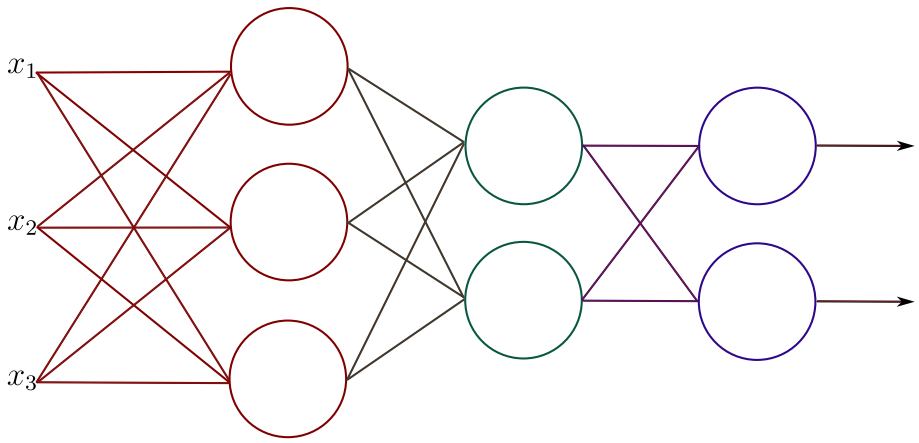


Figura 32 – Rede Perceptron Multicamadas com três variáveis, três camadas e duas saídas. Elaborada pelo autor.

Continuando com o exemplo da Figura 32, os neurônios em vermelho formam a primeira camada da RNA ou camada de entrada. As saídas dos neurônios desta camada estão diretamente ligadas com as entradas de cada um dos neurônios em verde. Isto significa que são as saídas dos neurônios da primeira camada que serão combinadas linearmente com os pesos sinápticos de cada um dos dois neurônios em verde, neurônios da segunda camada. Consequentemente, para a terceira camada, neurônios em azul, as saídas da segunda camada servirão de entradas.

A primeira camada do exemplo apresentado na Figura 32 é a camada de entrada e,

junto com a segunda camada, são conhecidas como camadas ocultas. Estas são designadas assim pois o operador/treinador da RNA não terá acesso a saída delas. A terceira camada é chamada de camada de saída e são os neurônios destas camadas que apresentarão as saídas para os dados fornecidos à RNA.

É importante evidenciar que, não só no exemplo da Figura 32, mas em todas as RNA do tipo PMC, as saídas dos neurônios serão suas saídas das funções de ativação. Outro fato importante é que a função de ativação escolhida é a adotada por todos os neurônios da rede.

Deste modo, todos os neurônios da segunda camada em diante recebem os dados da camada anterior, as ponderam com seus respectivos pesos sinápticos, comparam com os respectivos limiares e então fornecem esta combinação linear para as funções de ativação, enquanto os neurônios da primeira camada realizam estes mesmos processos mas com as variáveis dos dados das entradas.

Os pesos sinápticos de cada neurônio são independentes dos outros neurônios. Isto significa que eles devem ser treinados e devidamente responsabilizados quando a rede apresentar um erro. O algoritmo de treinamento que será estudado neste trabalho é o *backpropagation*.

Este algoritmo foi escolhido para ser estudado neste capítulo devido aos avanços que pode proporcionar em relação a outros algoritmos que existiam na época em que foi apresentado, inspirando o desenvolvimento de adaptações e melhorias para casos específicos (Widrow; Lehr, 1990).

Antes de apresentar o treinamento de uma rede neural em maiores detalhes, será implementada uma função booleana que não poderia ser implementada com o Perceptron de camada única ou com o Adaline, a função booleana ou-exclusivo ("XOU").

4.1 Implementando a função booleana ou-exclusivo "XOU"

Como visto na Figura 9, a função booleana "XOU" não é linearmente separável e portanto, não existe um modo de somente um neurônio ser capaz de implementá-la. Para que seja possível a implementação desta função, será criada uma rede neural com camadas.

Antes de definir a quantidade de neurônios e de camadas, deve-se preparar a implementação para que possa ser feita segundo os moldes do que foi realizado no capítulo 1 para as funções booleanas "OU" e "E".

Primeiramente constrói-se a tabela verdade da função "XOU":

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 5 – Tabela verdade da função booleana "XOU".

onde o sinal de $\oplus$ indica a atuação da função booleana "XOU" nas variáveis x_1 e x_2.

A arquitetura da RNA do tipo PMC que será adotada foi baseada na que foi apresentada por Kovács (2023) que utiliza a decomposição da função "XOU" com o uso de três funções booleanas. A função "XOU" pode ser expressa como combinação linear das funções "E", a função "NEGAÇÃO"($\neg$) e a função "OU".

A função booleana "NEGAÇÃO" inverte os valores binários que recebe, ou seja, se a entrada for "1" ela apresenta "0" como saída e vice-versa. A tabela verdade da função "NEGAÇÃO" para as variáveis de entrada x_1 e x_2 pode ser vista na Tabela 6.

x_1	x_2	$\neg x_1$	$\neg x_2$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	0

Tabela 6 – Tabela verdade da função "NEGAÇÃO". Elaborada pelo autor.

Recordando a notação adotada para as funções "E" e "OU", pode-se fazer, ao considerar como f a função booleana "XOU",

$$\begin{aligned} f(x_1, x_2) &= x_1 \oplus x_2 \\ &= (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2), \end{aligned}$$

e, fazendo $f_1 = x_1.(\neg x_2)$ e $f_2 = (\neg x_1).x_2$, tem-se $f = f_1 \vee f_2$.

É possível ver a equivalência do uso das funções para representar a função "XOU", com a saída desta função, na Tabela 7.

x_1	x_2	$\neg x_1$	$\neg x_2$	$x_1 \wedge \neg x_2$	$\neg x_1 \wedge x_2$	$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$	$x_1 \oplus x_2$
0	0	1	1	0	0	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	1	1
1	1	0	0	0	0	0	0

Tabela 7 – Tabela-verdade da função "XOU" $(A \oplus B)$ através de sua decomposição usando "NEGAÇÃO", "E" e "OU". Elaborada pelo autor.

Com isto, para esta RNA serão representados três neurônios: N_1, N_2 e N_3, associados às funções f_1, f_2 e f, respectivamente, conforme ilustrado na Figura 33.

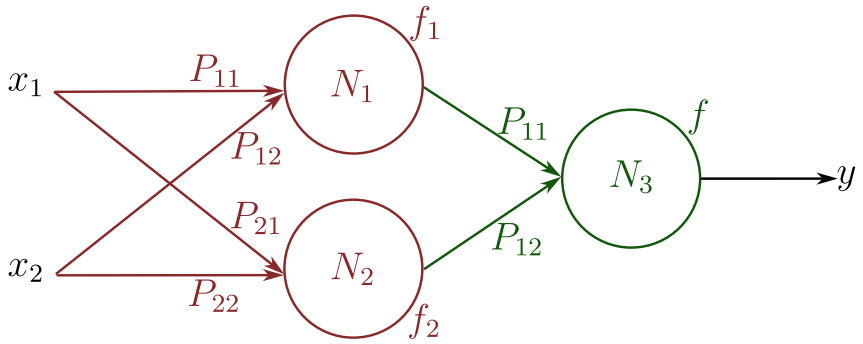


Figura 33 – Arquitetura envolvida na implementação da função booleana "XOU" com três neurônios de McCulloch-Pitts. Elaborado pelo autor.

De acordo com a decomposição apresentada, na Figura 33, a arquitetura desta RNA terá dois neurônios na primeira camada, responsáveis por receberem as entradas x_1 e x_2 e um neurônio na camada de saída que receberá as saídas dos neurônios da primeira camada e fornecerá a saída y da rede.

Na tabela verdade apresentada na Tabela 5, percebe-se que existem somente dois casos em que as saídas "1" são resultados da função "XOU". Com isso, pode-se atribuir cada um dos dois casos, respectivamente aos neurônios N_1 e N_2 e, em seguida, utiliza-se o neurônio N_3 para unir essas duas saídas em uma só.

O neurônio N_1, responsável por resolver $f_1(x_1, x_2) = x_1.(\neg x_2)$, encontra, por exemplo, uma solução tomando $P_{11}^{(1)} = \dfrac{1}{2}$ e $P_{12}^{(1)} = -\dfrac{1}{2}$, conforme ilustrado na Tabela 8.

(x_1, x_2)	$P_{11}^{(1)}.x_1 + P_{12}^{(1)}.x_2 - \theta = u_1$
$(0,0)$	$\frac{1}{2}.0 - \frac{1}{2}.0 = \ \ 0 - \theta$
$(0,1)$	$\frac{1}{2}.0 - \frac{1}{2}.1 = -\frac{1}{2} - \theta$
$(1,0)$	$\frac{1}{2}.1 - \frac{1}{2}.0 = \ \ \frac{1}{2} - \theta$
$(1,1)$	$\frac{1}{2}.1 - \frac{1}{2}.1 = \ \ 0 - \theta$

Tabela 8 – Combinação linear das variáveis com os pesos do primeiro neurônio da camada entrada. Elaborada pelo autor.

Com isso, o funcionamento do neurônio N_1 pode ser descrito pela função de ativação degrau definida pela Equação (4.1).

$$f_1(x_1, x_2) = \begin{cases} 1, \ se \ u_1 \geqslant 0 \\ 0, \ se \ u_1 < 0. \end{cases} \tag{4.1}$$

Os coeficientes fornecidos para o neurônio N_1, juntamente com um limiar $\theta = \frac{1}{4}$ que foi estipulado de acordo com a função degrau apresentada na Equação (1.5), podem

ser utilizados como coeficientes de uma reta que separa linearmente os dados, como pode ser visto na Figura 34.

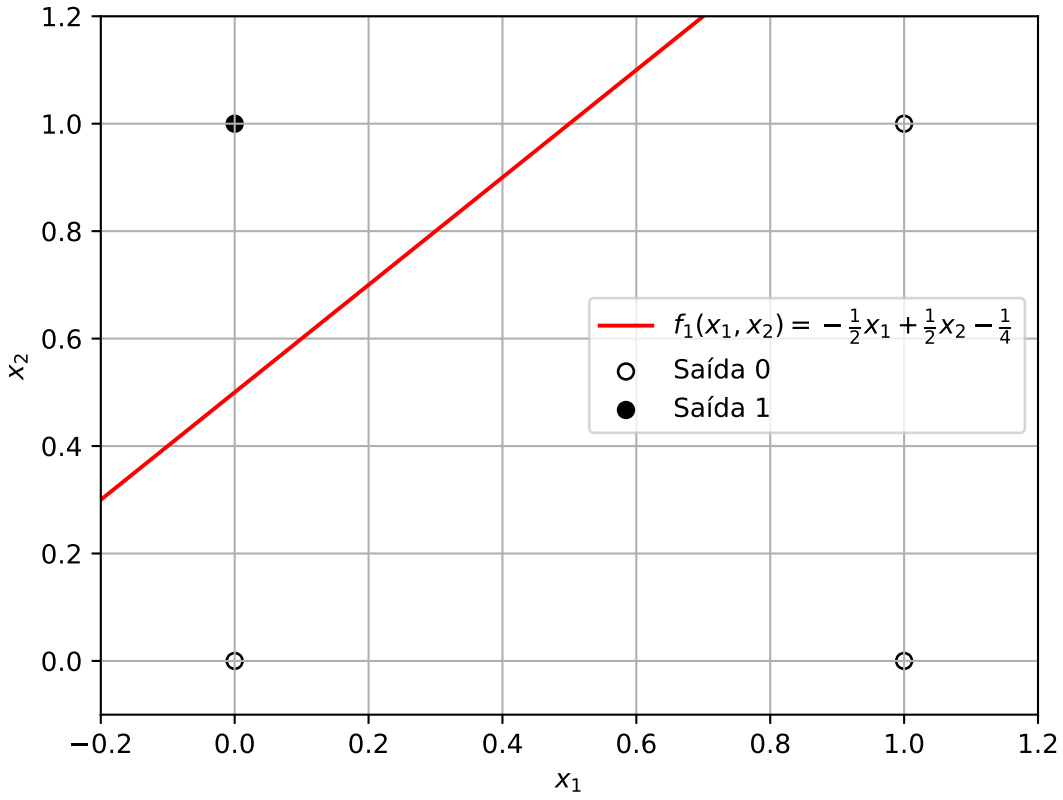


Figura 34 – Reta obtida com os pesos e limiar selecionados para f_1. Elaborado pelo autor.

De modo análogo, para N_2 com $f_2(x_1, x_2) = (\neg x_1).x_2$, pode-se fazer $P_{21}^{(1)} = -\frac{1}{2}$ e $P_{22}^{(1)} = \frac{1}{2}$ de modo a obter a Tabela 9:

(x_1, x_2)	$P_{21}^{(1)}.x_1 + P_{22}^{(1)}.x_2\theta = u_2$
$(0,0)$	$-\frac{1}{2}.0 + \frac{1}{2}.0 = 0 - \theta$
$(0,1)$	$-\frac{1}{2}.0 + \frac{1}{2}.1 = \frac{1}{2} - \theta$
$(1,0)$	$-\frac{1}{2}.1 + \frac{1}{2}.0 = -\frac{1}{2} - \theta$
$(1,1)$	$-\frac{1}{2}.1 + \frac{1}{2}.1 = 0 - \theta$

Tabela 9 – Combinação linear das variáveis com os pesos do segundo neurônio da camada de entrada. Elaborada pelo autor.

Utilizando a mesma função de ativação, definida em, (4.1), tem-se, para o neurônio

N_2:

$$f_2(x_1, x_2) = \begin{cases} 1, \ se \ u_2 \geqslant 0 \\ 0, \ se \ u_2 < 0 \end{cases} . \tag{4.2}$$

De modo análogo ao que foi feito para a função f_1, pode-se elaborar um gráfico ao utilizar os pesos sinápticos e limiar atribuído como $\theta = \frac{1}{4}$ como coeficientes de uma reta, como pode ser visto na Figura 35.

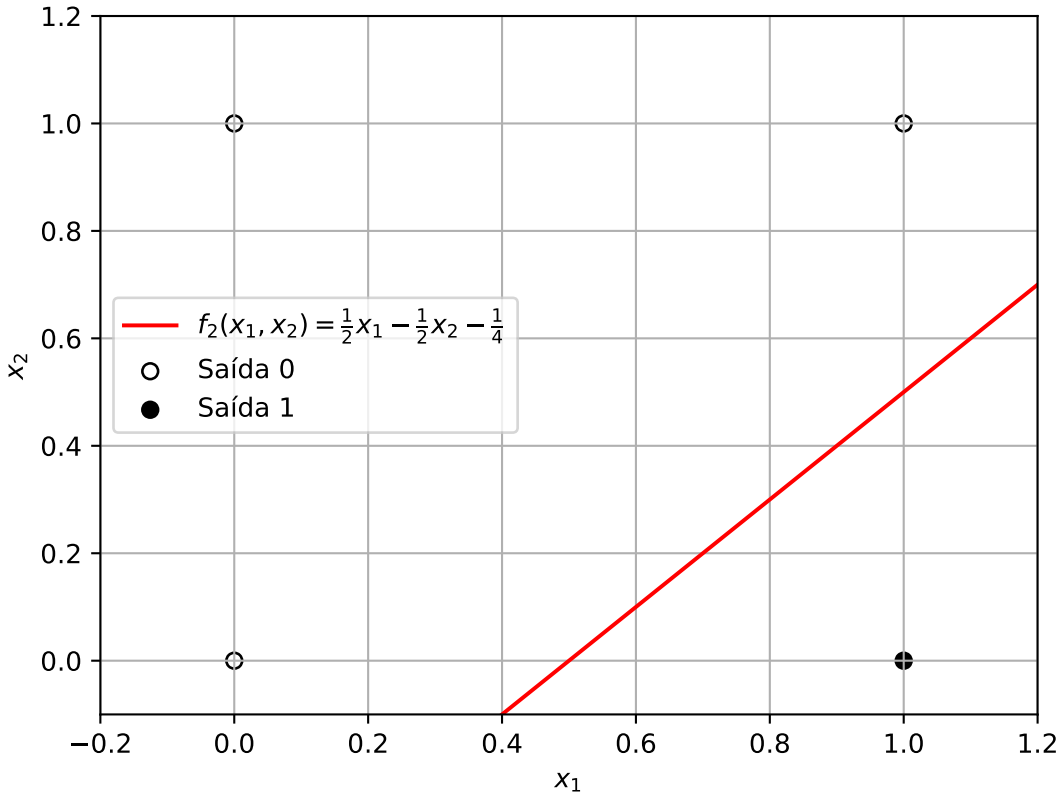


Figura 35 – Reta obtida com os pesos e limiar selecionados para f_2. Elaborado pelo autor.

Como os neurônios N_1 e N_2 pertencem a primeira camada de neurônios e N_3 à camada de saída, o neurônio N_3 não trabalhará diretamente com as entradas x_1 e x_2, mas sim receberá as respectivas saídas dos neurônios N_1 e N_2.

Estabelecidas as saídas dos neurônios da primeira camada, faz-se os ajustes necessários aos pesos da segunda camada de forma que a saída dela seja correspondente a da função que se deseja implementar.

Com base nas entradas x_1 e x_2 e nas saídas u_1 e u_2, dos neurônios da camada de entrada, é possível observar que o par ordenado $(N_1, N_2) = (1, 1)$ não fará parte do

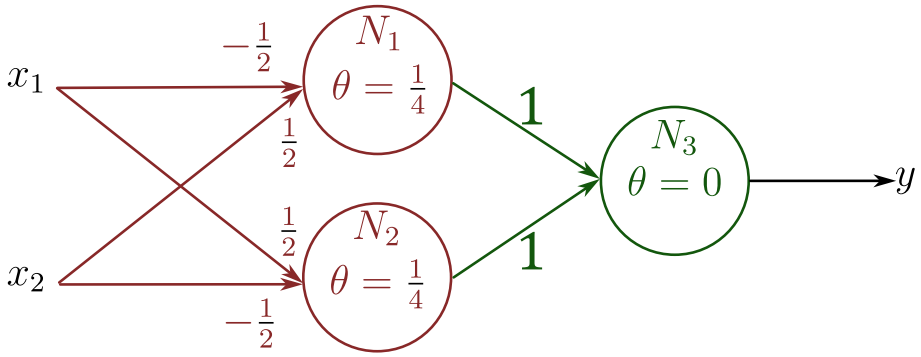


Figura 36 – Proposta de RNA para função "XOU" com pesos sinápticos atribuídos. Elaborado pelo autor.

domínio de f, pois independentemente de qualquer entrada atribuída à rede, a camada de entrada não apresentará simultaneamente o resultado mencionado. Tal fato pode ser observado na Tabela 10, adaptada da Tabela 7.

x_1	x_2	N_1	N_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

Tabela 10 – Tabela-verdade da função "XOU" ($A \oplus B$) e sua decomposição usando "NEGAÇÃO", "E" e "OU". Elaborada pelo autor.

Dessa forma, pode-se adotar $P_{11}^{(2)} = P_{12}^{(2)} = 1$, como os pesos sinápticos de N_3, e construir a tabela de saída da rede como:

(N_1, N_2)	$P_{11}^{(2)} \cdot N_1 + P_{12}^{(2)} \cdot N_3 = u_3$
$(0,0)$	$1.0 + 1.0 = 0$
$(0,1)$	$1.0 + 1.1 = 1$
$(1,0)$	$1.1 + 1.0 = 1$
$(0,0)$	$1.0 + 1.0 = 0$

Tabela 11 – Dados da primeira camada fornecidos à camada de saída.

À saída de N_3 aplica-se a função de ativação como definida da Equação (1.5), com $\theta = 0$, de modo a obter:

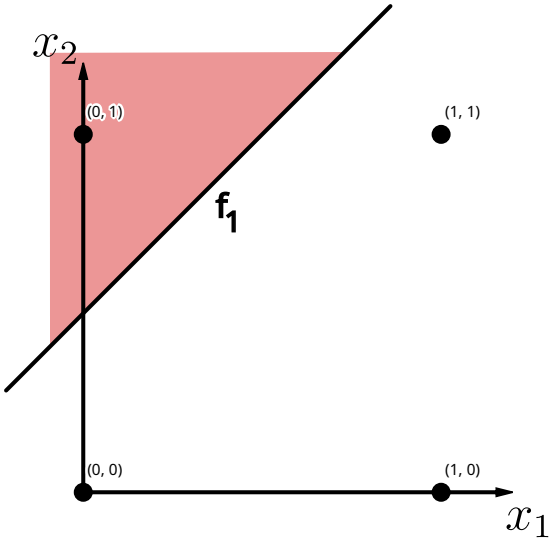
$$f(N_1, N_2) = \begin{cases} 1, \ se \ u_3 \geqslant 0 \\ 0, \ se \ u_3 < 0. \end{cases} \tag{4.3}$$

O diagrama na Figura 33, com os pesos atualizados, pode ser visto na Figura 36.

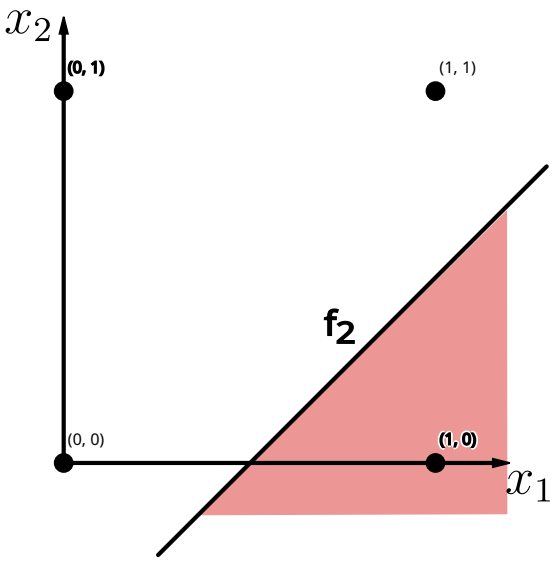
Com a atribuição dos pesos finalizada é possível escrever a saída da RNA em função dos dados das entradas, como pode ser visto na Equação (4.4).

$$f(x_1, x_2) = \begin{cases} 1, \ se \ x_1 \neq x_2 \\ 0, \ se \ x_1 = x_2. \end{cases} \tag{4.4}$$

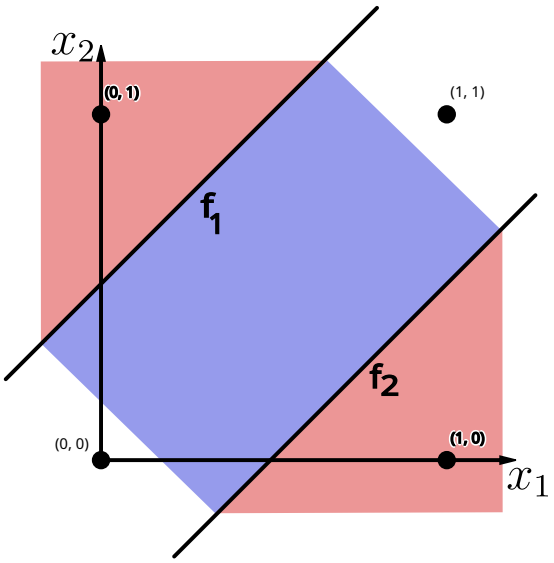
Uma representação acerca do resultado obtido por esta RNA é ilustrada na Figura 37, em que a Figura 37a apresenta isoladamente a representação geométrica da função f_1 e a Figura 37b a representação geométrica da função f_2.



(a) Função f_1.



(b) Função f_2.



(c) Função f.

Figura 37 – Composição de funções para a função booleana "XOU".
Elaborado pelo autor.

Na Figura 37c tem-se a representação geométrica da solução apresentada pela a

RNA ao problema do ou-exclusivo, onde a região em vermelho apresenta um dos resultados possíveis e a região em azul a outra possível solução.

Assim como foi feito no capítulo, 1, a atribuição dos pesos sinápticos foi realizada manualmente e de modo conveniente. Para uma RNA com maior complexidade, mais neurônios, mais camadas e com maior volume de dados de amostras, a atribuição manual de pesos sinápticos e de seus ajustes se torna inviável e, portanto, efetuar o treinamento por um algoritmo se torna necessário.

No livro Perceptrons: An Introduction to Computational Geometry (1988), os autores, apesar de incentivarem o estudo do Perceptron, estendem para as RNA do tipo PMC as limitações que trazem em seu texto, apesar de não dissertarem muito a respeito.

Este livro, segundo Bottou (2017) e Silva, Spatti e Flauzino (2016), foi um fator importante para a queda de investimentos no desenvolvimento de pesquisas e tecnologias envolvendo Redes Neurais Artificiais, que ocorreu logo após seu lançamento. Porém, ainda de acordo com Bottou (2017), os incentivos foram direcionados ao estudo de processadores de sinais adaptativos, categoria a qual o Adaline se enquadra.

Segundo Widrow e Lehr (1990), uma RNA em que vários Adaline se conectam é denominada de Madaline[1] e, tais redes detém algumas funções booleanas fixas conectando alguns neurônios, além disso, diferentes algoritmos de treinamentos foram desenvolvidos para o Madaline, porém, o algoritmo Madaline Rule III se mostrou matematicamente equivalente ao treinamento efetuado em uma rede PMC com o uso do algoritmo *backpropagation*.

4.2 O treinamento do Perceptron Multicamadas

O treinamento do PMC também é do tipo supervisionado. Os erros calculados entre as saídas geradas pela rede e as saídas desejadas constituem o papel principal do ajuste dos pesos sinápticos e, assim como no Adaline, a tentativa de minimizar este erro estabelece o caminho a ser percorrido no treinamento de uma rede PMC.

Este treinamento pode ser efetuado em *lotes*, *mini-lote*s ou *em linha*. Estas três estratégias se referem ao modo como as amostras de treinamento são utilizadas.

Segundo Géron (2019), o treinamento em *lotes* é muito lento se comparado aos outros dois. Isto se deve ao grande volume de dados que a rede deve armazenar para executar as etapas do treinamento. Por outro lado, Haykin (2001) afirma que este é o método que tem maior estabilidade na convergência da rede para valores ótimos de pesos sinápticos.

O treinamento efetuado *em linha* atualiza os pesos sinápticos após o processamento

[1] Acrônimo para *Many Adaline*, em português, Muitos Adaline.

de cada amostra que é fornecida a rede. Este método é mais econômico em relação à custos computacionais como processamento e armazenamento, porém é instável. A cada amostra inserida, a convergência dos pesos sinápticos pode se afastar do mínimo global (Haykin, 2001).

Uma alternativa aos dois seria o uso de *mini-lotes* de dados de amostras. Esta é uma estratégia que fica no meio termo entre a estratégia de *lotes* e a *em linha* em todos os quesitos, pois utiliza pequenos lotes das amostras, sendo estes partes menores do que o lote completo.

Diferentemente do que foi feito nos capítulos anteriores, para o PMC será utilizado o treinamento *em linha*. Esta estratégia de treinamento é adotada para que as passagens realizadas aqui estejam mais próximas das utilizadas por Silva, Spatti e Flauzino (2016), Haykin (2001), e Géron (2019), por exemplo. Isto é feito com a expectativa de que, utilizar uma amostra por vez, possibilite um melhor entendimento do que está sendo feito, já que a notação utilizada pode fornecer maiores detalhes no processo de treinamento.

4.3 O algoritmo *backpropagation*

O *backpropagation*, também chamado de retropropagação, recebe seu nome pois, após avaliar os erros apresentados pelas saídas da rede em relação às saídas desejadas, retorna este erro juntamente com um fator de correção a todos os neurônios.

Este algoritmo foi apresentado no fim da década de 1980 e utiliza a ideia de controle e redução dos erros entre as saídas produzidas pela RNA e as respectivas saídas desejadas, similarmente ao que é feito no Adaline.

Todos os neurônios de um PMC estão conectados seja diretamente ou indiretamente, por meio de outros neurônios. Isto significa que um neurônio com pesos mal ajustados pode acarretar no mau funcionamento de toda RNA, afetando as saídas produzidas pela rede. Segundo Haykin (2001), o algoritmo *backpropagation* consegue atribuir níveis de responsabilidades aos neurônios que mais, ou menos, desviam a rede da saída desejada e ajusta seus pesos de acordo.

O algoritmo de treinamento *backpropagation* contém duas fases. A primeira consiste na passagem dos dados das amostras até a apresentação da saída da RNA, e a segunda fase, em que um erro é calculado e retornado até os neurônios da primeira camada. Estas fases se repetem, alternadamente, até que a rede determine que os pesos estão ajustados segundo um critério adotado para o encerramento do treinamento.

Segundo Widrow e Lehr (1990), a mudança da função degrau, empregada principalmente nos exemplos envolvendo o neurônio de McCulloch-Pitts realizados neste trabalho e apresentadas, por exemplo, nas Equações (1.5) e (4.4), para as funções do tipo sigmoides,

permitiu que o algoritmo *backpropagation* pudesse ser aplicado no treinamento de um PMC.

Géron (2019, p.270), comenta que as funções sigmoides foram utilizadas durante muito tempo como analogia ao que se parecia com a curva relativa a ativação de neurônios biológicos mas, atualmente, a mais utilizada é a função ReLU[2], Figura 40.

A Tabela 12 apresenta quatro das funções de ativação amplamente utilizadas na literatura.

Função	**Fórmula**	**Conjunto imagem**
Logística	$\sigma(x) = \dfrac{1}{1+e^{-x}}$	(0, 1)
ReLU	$\text{ReLU}(x) = \begin{cases} x, & \text{se } x > 0, \\ 0, & \text{se } x \leqslant 0 \end{cases}$	$[0, +\infty)$
Tangente Hiperbólica	$\tanh(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1, 1)
Degrau	$f(x) = \begin{cases} 1, & \text{se } x \geqslant 0, \\ 0, & \text{se } x < 0 \end{cases}$	{0, 1}

Tabela 12 – Tabelas com algumas funções de ativação. Elaborado pelo autor.

Segundo Widrow e Lehr (1990), as funções de ativação adotadas ditas do tipo sigmoide são aquelas que apresentam uma curva em forma de S quando seu domínio varia de valores próximos a -1 até valores próximos a 1 como pode ser visto na Figura 39.

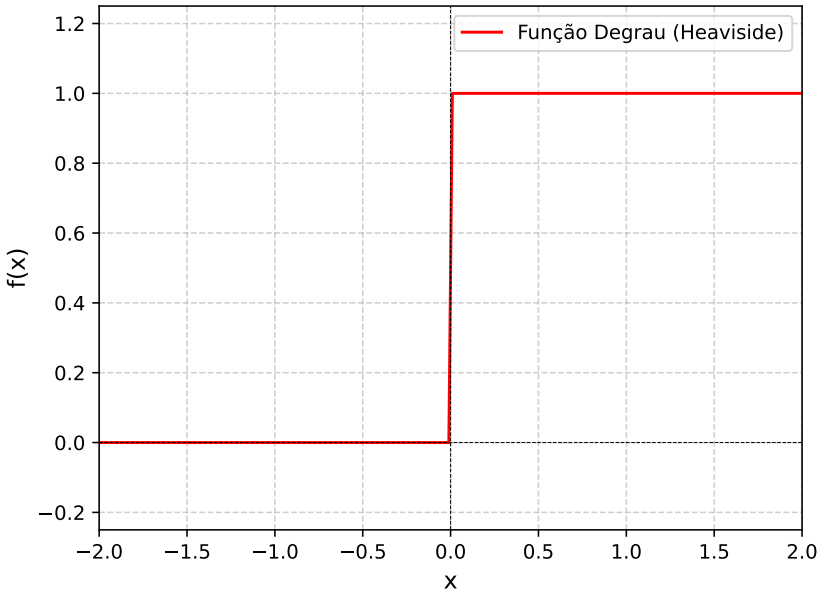


Figura 38 – Gráfico da função degrau, também conhecida como função degrau ou função de Heaviside. Elaborado pelo autor.

[2] Acrônimo para *Rectified Linear Unit*.

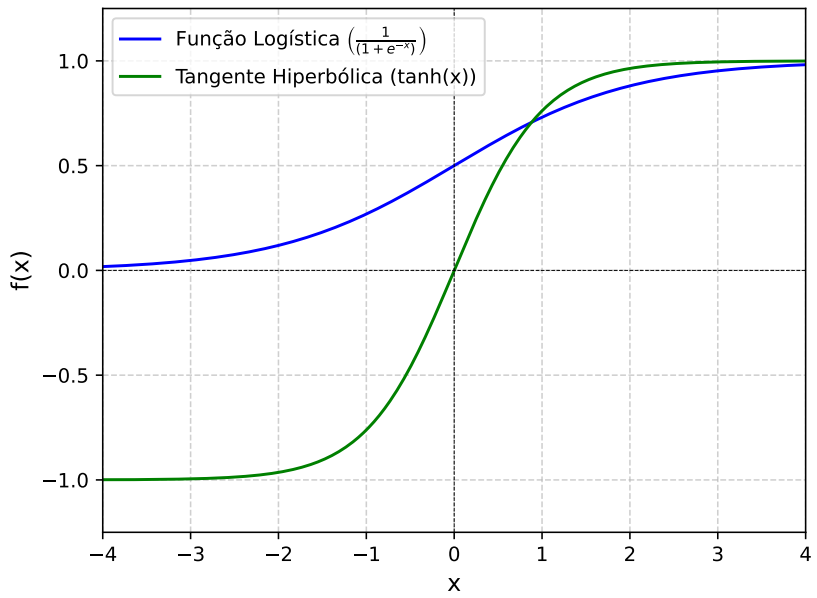


Figura 39 – Exemplos de gráficos de duas funções ditas sigmoidais, de acordo com Widrow e Lehr (1990). Elaborado pelo autor.

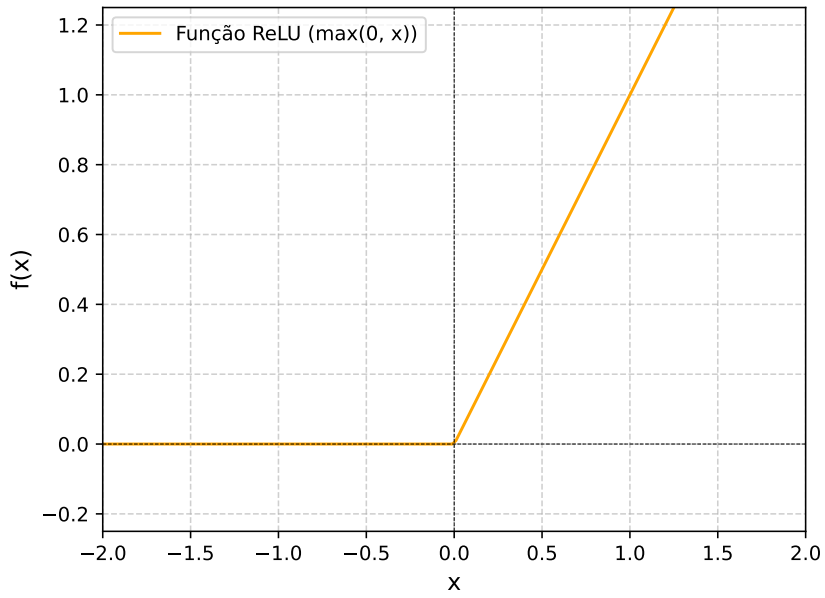


Figura 40 – Gráfico da função ReLU. Elaborado pelo autor.

4.3.1 A fase *Forward*

Devidos aos propósitos didáticos deste texto, a apresentação do algoritmo de treinamento do PMC será desenvolvida com base no modelo simplificado apresentado na Figura 42. A utilização de um exemplo genérico, como o da Figura 41, demandaria um

maior nível de abstração por parte do leitor que poderia ficar preso no entendimento do elevado número de índices que apareceriam neste exemplo generalizado.

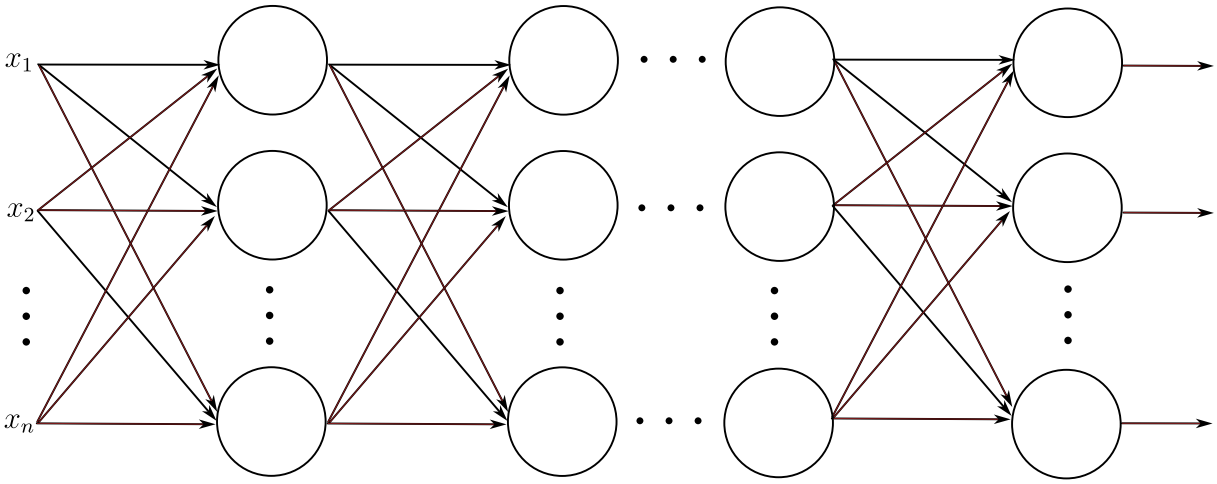


Figura 41 – Perceptron Multicamadas genérico. Elaborado pelo autor.

Caso leitor tenha interesse na versão generalizada desta demonstração, recomenda-se a leitura de Redes Neurais: Princípios e Prática (2001), que é muito bem detalhada, enquanto uma apresentação mais sucinta pode ser vista em 30 years of adaptive neural networks: perceptron, Madaline, and backpropagation (1990).

Será adotado o modo escolhido por Silva, Spatti e Flauzino (2016) ao apresentar um caso particular de PMC para ilustrar o funcionamento do algoritmo de treinamento, através da utilização de um PMC com três camadas, contendo três neurônios na primeira camada (de entrada), dois na segunda (camada oculta) e dois na terceira camada (de saída), como visto na Figura 42.

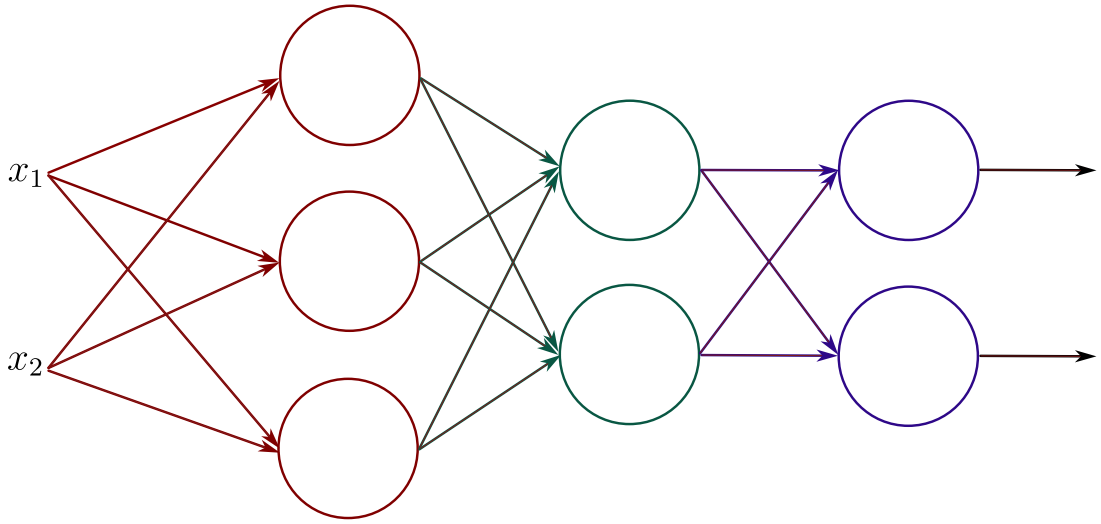


Figura 42 – PMC322 com sentidos da fase *forward* indicados pelas setas. Elaborado pelo autor.

A Figura 42 apresenta não só a rede utilizada mas também o sentido em que os dados são propagados pela rede PMC na fase *forward*.

Antes de prosseguir, deve-se identificar a quais neurônios os pesos sinápticos se referem. As camadas serão ordenadas da esquerda para a direita utilizando a ordem numérica crescente, assim como serão identificados os neurônios de cima para baixo, também em ordem crescente e em ambos começando pelo 1.

Com esta convenção estabelecida, pode-se identificar a qual camada se refere, e a qual neurônio pertencem, os pesos sinápticos. A Figura 43 apresenta os pesos sinápticos em seus respectivos neurônios e camadas.

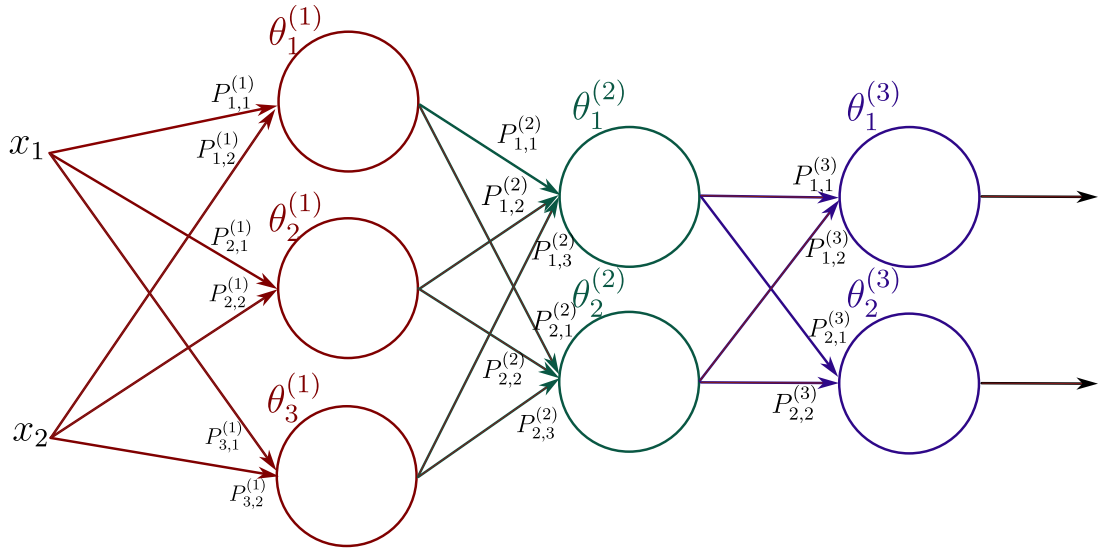


Figura 43 – PMC322 com pesos sinápticos identificados. Elaborado pelo autor.

Por meio da Figura 43 , é possível exemplificar o modo que os pesos estão identificados. O peso genérico $P_{ij}^{(c)}$ é um peso sináptico que pondera a entrada j, advinda da camada $(c-1)$, ao neurônio i da camada (c). Por exemplo, $P_{1,2}^{(2)}$ é um peso sináptico da camada 2, que pondera o sinal proveniente do 2ª neurônio da camada 1 como entrada ao 1º neurônio da camada 2.

Pode-se adotar quaisquer funções mencionadas na tabela 12, com exceção da função degrau. Porém, pretende-se seguir com a notação de função f, sem especificar uma função, com o objetivo de apresentar o modo como os dados são propagados entre as camadas.

Como mencionado anteriormente, será realizado o treinamento em linha. Isto significa que serão assumidos x_1 e x_2 como sendo variáveis de uma amostra específica do conjunto de amostras x. Estas variáveis formarão o conjunto de entradas. Os neurônios da primeira camada as combinam linearmente com seus respectivos pesos sinápticos e limiares para em seguida aplicar a função de ativação. Pode-se escrever esta etapa como

na Equação 4.5.

$$\begin{aligned} Y_1^{(1)} &= f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) \\ Y_2^{(1)} &= f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) \\ Y_3^{(1)} &= f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right). \end{aligned} \tag{4.5}$$

Na Equação (4.5), o $Y_j^{(c)}$ representa a saída do neurônio j da camada c, por exemplo, $Y_3^{(1)}$ representa a saída do terceiro neurônio da primeira camada, e, $\theta_j^{(c)}$ representa o limiar do neurônio j da camada c.

A segunda camada receberá as saídas da primeira camada e realizará com elas as combinações lineares com pesos sinápticos e limiares específicos, de modo similar ao que foi realizado para a primeira camada, como é possível ver na Equação (4.6).

$$\begin{aligned} Y_1^{(2)} &= f\left(P_{11}^{(2)} \cdot Y_1^{(1)} + P_{12}^{(2)} \cdot Y_2^{(1)} + P_{13}^{(2)} \cdot Y_3^{(1)} - \theta_1^{(2)}\right) \\ Y_2^{(2)} &= f\left(P_{21}^{(2)} \cdot Y_1^{(1)} + P_{22}^{(2)} \cdot Y_2^{(1)} + P_{23}^{(2)} \cdot Y_3^{(1)} - \theta_2^{(2)}\right). \end{aligned} \tag{4.6}$$

Para a terceira camada, a camada de saída, pode-se fazer como na Equação (4.7).

$$\begin{aligned} Y_1^{(3)} &= f\left(P_{11}^{(3)} \cdot Y_1^{(2)} + P_{12}^{(3)} \cdot Y_2^{(2)} - \theta_1^{(3)}\right) \\ Y_2^{(3)} &= f\left(P_{21}^{(3)} \cdot Y_1^{(2)} + P_{22}^{(3)} \cdot Y_2^{(2)} - \theta_2^{(3)}\right). \end{aligned} \tag{4.7}$$

Uma imagem com todas as saídas com seus respectivos neurônios pode ser vista na Figura 44.

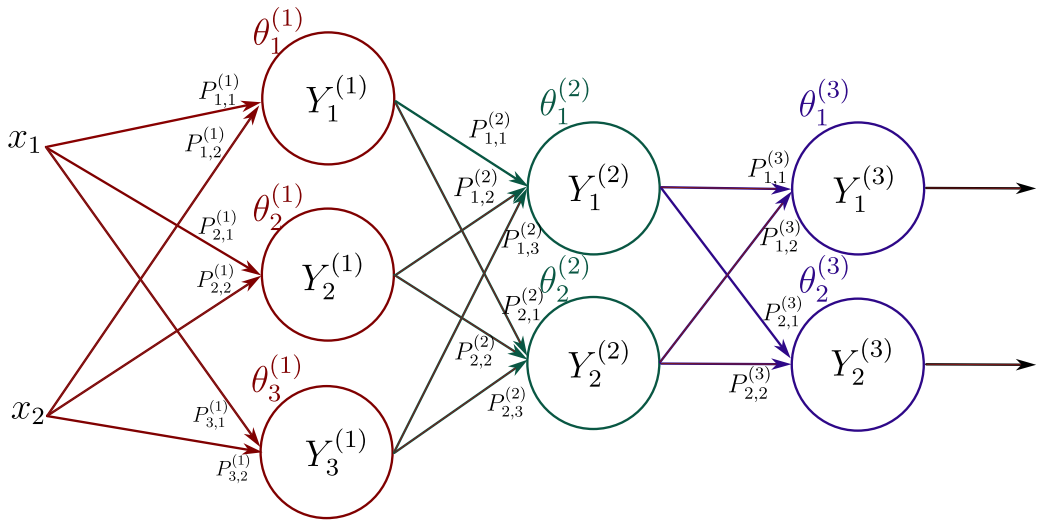


Figura 44 – PMC322 com todas as saídas identificadas com seus respectivos neurônios. Elaborada pelo autor.

As saídas $Y_1^{(3)}$ e $Y_2^{(3)}$ são as que o operador da rede tem acesso. Geralmente, segundo Silva, Spatti e Flauzino (2016), é adotada a prática conhecida como *one of c-classes*, em que, cada neurônio da saída é utilizado para indicar uma das categorias possíveis para os dados. Em outras palavras, esta prática adota para a camada de saída o mesmo número de classes a serem mapeadas com as amostras, caso o problema envolva classificação de dados.

Segundo esta estratégia, para os neurônios da camada de saída a situação específica da função "XOU", por exemplo, a RNA do tipo PMC teria dois neurônios na camada de saída em que cada um deles representaria "1" ou "0", respectivamente. Desta forma, somente um dos neurônios seria ativado como forma de apresentar a classe a qual os dados pertencem.

É possível, ainda, apresentar as saídas dos neurônios da última camada em função dos dados de entrada da primeira camada, como pode ser visto na Equação (4.8). Para facilitar a compreensão dos termos envolvidos nesta Equação, os pesos sinápticos estão coloridos de acordo com a camada a qual pertencem, em vermelhos, verdes e azuis os pesos referentes a primeira, segunda e terceira camadas, respectivamente.

$$
\begin{cases}
Y_1^{(3)} = f\left(
\begin{array}{c}
P_{11}^{(3)} \cdot f\left(
\begin{array}{c}
P_{11}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{12}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{13}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_1^{(2)}
\end{array}
\right) \\
+ \\
P_{12}^{(3)} \cdot f\left(
\begin{array}{c}
P_{21}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{22}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{23}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_2^{(2)}
\end{array}
\right) - \theta_1^{(3)}
\end{array}
\right) \\
Y_2^{(3)} = f\left(
\begin{array}{c}
P_{21}^{(3)} \cdot f\left(
\begin{array}{c}
P_{11}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{12}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{13}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_1^{(2)}
\end{array}
\right) \\
+ \\
P_{22}^{(3)} \cdot f\left(
\begin{array}{c}
P_{21}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{22}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{23}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_2^{(2)}
\end{array}
\right) - \theta_2^{(3)}
\end{array}
\right)
\end{cases}
\tag{4.8}
$$

Para condensar as informações apresentadas pelas combinações lineares, será chamado de $u_i^{(c)}$ a combinação linear referente ao neurônio i na camada c, e fazer a Equação (4.8) se tornar a Equação (4.9).

$$
\begin{cases}
Y_1^{(3)} = f\left(\begin{array}{c}
P_{11}^{(3)} \cdot f\left(\begin{array}{c} P_{11}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{12}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{13}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_1^{(2)} \end{array}\right) \\ + \\ P_{12}^{(3)} \cdot f\left(\begin{array}{c} P_{21}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{22}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{23}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_2^{(2)} \end{array}\right) - \theta_1^{(3)}
\end{array}\right), \\
Y_2^{(3)} = f\left(\begin{array}{c}
P_{21}^{(3)} \cdot f\left(\begin{array}{c} P_{11}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{12}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{13}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_1^{(2)} \end{array}\right) \\ + \\ P_{22}^{(3)} \cdot f\left(\begin{array}{c} P_{21}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{22}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{23}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_2^{(2)} \end{array}\right) - \theta_2^{(3)}
\end{array}\right)
\end{cases}
$$

$$
\Leftrightarrow
$$

$$
\begin{cases}
Y_1^{(3)} = f\left(P_{11}^{(3)} \cdot f\left(u_1^{(2)}\right) + P_{12}^{(3)} \cdot f\left(u_2^{(2)}\right) - \theta_1^{(3)}\right), \\
Y_2^{(3)} = f\left(P_{21}^{(3)} \cdot f\left(u_1^{(2)}\right) + P_{22}^{(3)} \cdot f\left(u_2^{(2)}\right) - \theta_2^{(3)}\right)
\end{cases}
$$

$$
\Leftrightarrow
$$

$$
\begin{cases}
Y_1^{(3)} = f\left(u_1^{(3)}\right), \\
Y_2^{(3)} = f\left(u_2^{(3)}\right)
\end{cases}
\tag{4.9}
$$

É importante relembrar que, como f é uma função com conjunto imagem contido em um intervalo real, as saídas são valores reais. A partir do momento em que as saídas apresentadas na Equação (4.9) são obtidas, é iniciada a próxima fase do treinamento, a fase *backward*.

4.3.2 A fase *Backward*

Ao neurônio da rede PMC é atribuída duas funções, sendo primeira delas vista na seção anterior, que não se diferencia muito daquela proposta ao neurônio quando está

atuando isoladamente em uma rede do tipo Adaline ou Perceptron: a combinação linear dos pesos sinápticos com as variáveis e o uso da função de ativação.

A segunda função atribuída ao neurônio é a de realizar os ajustes de seus pesos sinápticos de acordo com a posição que está na arquitetura da rede neural. Estes ajustes são dependentes dos ajustes realizados nos neurônios na camada seguinte ao neurônio em questão.

É nesta fase, chamada *backward*[3] , que com o uso da minimização da função Erro Quadrático, os pesos da camada de saída são ajustados e os fatores utilizados neste ajustes são propagados de trás para frente, ou seja, no sentido dos neurônios da camada de saída para os neurônios da primeira camada, como ilustrado na Figura 45.

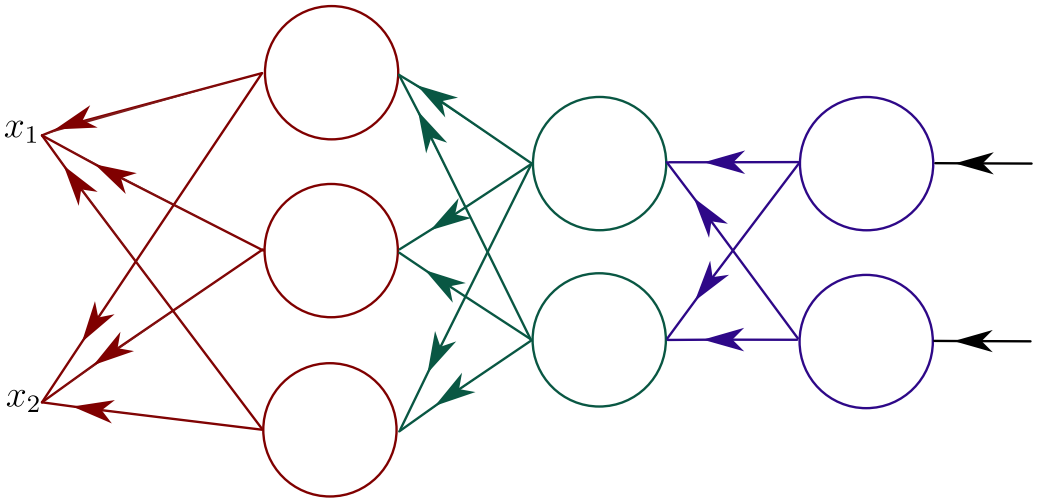


Figura 45 – PMC322 Com sentido indicado do fluxo de informações na etapa *backward* do treinamento. Elaborado pelo autor.

Esta estratégia é adotada pois, no caso do PMC não se tem acesso às saídas de todos os neurônios, mas sim somente às dos neurônios na camada de saída.

Como estas saídas são na verdade funções de outras funções, e assim sucessivamente, até uma função diretamente em função dos primeiros pesos sinápticos, como pode ser visto na Equação (4.8), pode-se basear na fundamentação relativa à composição de funções para pensar no ajuste dos pesos sinápticos.

A função Erro Quadrático que será minimizada é a mesma utilizada no capítulo 3, porém, será aplicada em mais de um neurônio caso a arquitetura do PMC em questão tenha mais de um neurônio em sua camada de saída.

Como esta função não pode ser aplicada diretamente em função dos pesos sinápticos de camadas anteriores às da saída, o processo de minimização que depende do uso de derivadas, agora também dependerá da regra da cadeia para derivadas. Aqui também

[3] Uma possível tradução para o português seria algo como “retroativo”.

deve-se recorrer ao uso do gradiente, pois, cada camada contém um conjunto de pesos que atuam ao mesmo tempo na função E, Equação (4.10).

Continuando com o exemplo de neurônio PMC322, Figura 45, serão obtidos os fatores a serem utilizados nos ajustes dos pesos sinápticos dos neurônios da camada de saída. Assim como no Adaline, com o objetivo de minimizar o Erro Quadrático, utiliza-se a ideia de otimização associada às derivadas de E em função dos pesos sinápticos da camada, e neurônio, em questão. A Equação (4.10) apresenta o erro quadrático da camada de saída:

$$E^{(3)}(P) = \frac{1}{2}\left[\left(d_1 - Y_1^{(3)}\right)^2 + \left(d_2 - Y_2^{(3)}\right)^2\right]. \tag{4.10}$$

Na Equação (4.10), d_1 e d_2 são as saídas associadas aos valores específicos da amostra que possui x_1 e x_2 como variáveis de entrada.

4.3.2.1 Ajustando os pesos dos neurônios da camada de saída (terceira camada)

O objetivo do processo de treinamento, para cada camada de neurônios, é ajustar os pesos sinápticos de modo a minimizar o erro entre a saída da rede e a saída desejada. Isto implica na minimização da função $E^{(3)}$, dada pela Equação (4.10).

Calculando a derivada de (4.10) em relação aos pesos, P_{ij} da camada de saída tem-se o gradiente do erro quadrático:

$$\nabla E^{(3)} = \frac{\partial E}{\partial P_{i,j}^{(3)}} = \frac{\partial E}{\partial Y_i^{(3)}} \frac{\partial Y_i^{(3)}}{\partial u_i^{(3)}} \frac{\partial u_i^{(3)}}{\partial P_{i,j}^{(3)}}. \tag{4.11}$$

Como está sendo utilizada uma configuração particular de RNA, pode-se atribuir valores aos índices i e j de modo a explorar o os termos da Equação (4.11). Analisando o gradiente em função do peso sináptico $P_{1,1}^{(3)}$, adotando $i = 1$ e $j = 1$, a Equação apresentada em (4.11) resulta na Equação (4.12):

$$\frac{\partial E}{\partial P_{1,1}^{(3)}} = \overbrace{\frac{\partial E}{\partial Y_1^{(3)}}}^{1)} \underbrace{\frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}}}_{2)} \overbrace{\frac{\partial u_1^{(3)}}{\partial P_{1,1}^{(3)}}}^{3)}. \tag{4.12}$$

Analisando termo a termo da regra da cadeia apresentada na Equação (4.12), tem-se que:

$$\begin{aligned} 1) \quad & \frac{\partial E}{\partial Y_1^{(3)}} = -(d_1 - Y_1^{(3)}) \\ 2) \quad & \frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}} = f'(u_1^{(3)}) \\ 3) \quad & \frac{\partial u_1^{(3)}}{\partial P_{1,1}^{(3)}} = Y_1^{(2)}. \end{aligned} \tag{4.13}$$

De (4.13) tem-se que:

$$\frac{\partial E}{\partial P_{1,1}^{(3)}} = -(d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) \cdot Y_1^{(2)}. \tag{4.14}$$

Dessa forma, o fator de correção $\Delta P_{1,1}^{(3)}$, a ser utilizado para ajustar o peso sináptico $P_{1,1}^{(3)}$, deve ser efetuado na direção oposta ao gradiente a fim de minimizar o erro, ou seja:

$$\begin{aligned} \Delta P_{1,1}^{(3)} &= -\eta \cdot \frac{\partial E}{\partial P_{1,1}^{(3)}} \\ &= \eta \cdot (d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) \cdot Y_1^{(2)} \\ &= \eta \cdot \delta_1^{(3)} \cdot Y_1^{(2)}, \end{aligned} \tag{4.15}$$

em que $\delta_1^{(3)} = (d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)})$ é definido como o gradiente local em relação ao primeiro neurônio da camada de saída.

A constante η define a taxa de aprendizagem para o treinamento e tem proposta semelhante à apresentada para as RNA anteriores ao PMC neste trabalho, ou seja, corresponde ao quanto dos fatores de correção serão incorporados ao novo peso sináptico.

Na segunda linha da Equação (4.13) aparece a derivada da função f, ou seja, a derivada da função de ativação adotada para a rede, cujos valores são apresentados na Tabela 13.

Função	**Derivada**
Logística	$\sigma'(x) = \sigma(x)(1 - \sigma(x))$
ReLU	$\text{ReLU}'(x) = \begin{cases} 1, & \text{se } x > 0, \\ 0, & \text{se } x \leqslant 0 \end{cases}$
Tangente Hiperbólica	$\tanh'(x) = 1 - \tanh^2(x)$
Degrau	$f'(x) = \begin{cases} \text{indefinido}, & \text{em } x = 0, \\ 0, & \forall x \in \mathbb{R} - \{0\} \end{cases}$

Tabela 13 – Tabela com as derivadas de algumas funções de ativação. Elaborado pelo autor.

De modo similar ao que foi realizado com o peso sináptico $P_{1,1}^{(3)}$, são obtidos os fatores de correção para todos os pesos sinápticos do primeiro neurônio da terceira camada

(camada de saída), como pode ser visto na Equação (4.16):

$$\begin{aligned} \Delta P_{1,1}^{(3)} &= \eta \cdot \delta_1^{(3)} \cdot Y_1^{(2)} \\ \Delta P_{1,2}^{(3)} &= \eta \cdot \delta_1^{(3)} \cdot Y_2^{(2)}. \end{aligned} \tag{4.16}$$

Desta forma, a atualização que os pesos sinápticos do primeiro neurônio da terceira camada será atualizado pela Equação (4.17), com o índice j representando a origem da informação de atualização.

$$P_{1,j}^{(3)(novo)} = P_{1,j}^{(3)(antigo)} + \eta \cdot \delta_1^{(3)} \cdot Y_j^{(2)}. \tag{4.17}$$

Analogamente ao que foi exposto nas Equações (4.16) e (4.17), com as devidas atualizações de índices, os pesos sinápticos para o segundo neurônio da terceira camada é atualizado.

De modo geral, é possível escrever para os neurônios da camada de saída a regra de ajustes apresentada na Equação (4.18):

$$P_{i,j}^{(3)(novo)} = P_{i,j}^{(3)(antigo)} + \eta \cdot \delta_i^{(3)} \cdot Y_j^{(2)}, \tag{4.18}$$

em que os índices i e j dizem respeito ao destino e a sua origem dos pesos em atualização, respectivamente.

Finalizados os ajustes dos pesos sinápticos de todos os neurônios da camada de saída, ou seja, da terceira camada, deve-se iniciar os ajustes dos neurônios da camada anterior a esta, ou seja, a segunda camada.

4.3.2.2 Ajustando os pesos dos neurônios da segunda camada

Como não se tem acesso às saídas destes neurônios para que se possa efetuar diretamente os ajustes, será considerado o fato das saídas da RNA do tipo PMC serem, basicamente, funções compostas por outras funções, e assim utilizar a regra da cadeia para ajustar os neurônios de camadas ocultas.

Deste modo, calculando-se gradiente da função erro quadrático E, agora em função dos pesos sinápticos da segunda camada, obtém-se a Equação (4.19):

$$\nabla E^{(2)} = \frac{\partial E}{\partial P_{i,j}^{(2)}} = \frac{\partial E}{\partial Y_i^{(2)}} \frac{\partial Y_i^{(2)}}{\partial u_i^{(2)}} \frac{\partial u_i^{(2)}}{\partial P_{i,j}^{(2)}}. \tag{4.19}$$

Utilizando a mesma estratégia utilizada anteriormente, serão adotados os valores para os índices com $i = 1$ e $j = 1$, ou seja, serão detalhados os processos para atualização do peso sináptico $P_{1,1}^{(2)}$ e a saída $Y_1^{(2)}$.

Com isto, obtém-se:

$$\frac{\partial E}{\partial P_{1,1}^{(2)}} = \underbrace{\frac{\partial E}{\partial Y_1^{(2)}}}_{1)} \overbrace{\frac{\partial Y_1^{(2)}}{\partial u_1^{(2)}}}^{2)} \underbrace{\frac{\partial u_1^{(2)}}{\partial P_{1,1}^{(2)}}}_{3)}. \tag{4.20}$$

Os termos 2) e 3) da Equação (4.20) são análogos aos da terceira camada, com os devidos ajustes de índices e, são dados pela Equação (4.21):

$$\begin{aligned} 2)\, &\frac{\partial Y_1^{(2)}}{\partial u_1^{(2)}} = f'(u_1^{(2)}) \\ 3)\, &\frac{\partial u_1^{(2)}}{\partial P_{1,1}^{(2)}} = Y_1^{(1)}. \end{aligned} \tag{4.21}$$

Como a função E não recebe diretamente a saída $Y_1^{(2)}$, a derivada parcial apresentada em 1) precisará de algumas manipulações. Primeiramente, deve-se lembrar que $u_1^{(3)}$ e $u_2^{(3)}$ recebem $Y_1^{(2)}$ e que com ele operam com seus respectivos pesos sinápticos $P_{1,1}^{(3)}$ e $P_{2,1}^{(3)}$, como pode ser visto na Equação (4.7). Desta forma, a derivada parcial 1) pode ser reescrita através da regra da cadeia dada pela Equação (4.22):

$$\begin{aligned} 1)\quad \frac{\partial E}{\partial Y_1^{(2)}} &= \frac{\partial E}{\partial u_1^{(3)}} \frac{\partial u_1^{(3)}}{\partial Y_1^{(2)}} + \frac{\partial E}{\partial u_2^{(3)}} \frac{\partial u_2^{(3)}}{\partial Y_1^{(2)}} \\ &= \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}} P_{1,1}^{(3)} + \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_2^{(3)}} P_{2,1}^{(3)}. \end{aligned} \tag{4.22}$$

Os resultados das regras das cadeias apresentadas em ambos os fatores, em preto e em azul, na Equação (4.22) podem ser obtidos por meio da aplicação do gradiente realizado na Equação (4.11). Isto significa que estes fatores são, na verdade, os mesmos que apareceram na Equação (4.13) como é possível ver na Equação (4.23):

$$\begin{aligned} \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}} &= -(d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) = \delta_1^{(3)} \\ \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_2^{(3)}} &= -(d_2 - Y_2^{(3)}) \cdot f'(u_2^{(3)}) = \delta_2^{(3)}. \end{aligned} \tag{4.23}$$

Desta forma, é possível reescrever a Equação (4.22) como a Equação (4.24).

$$1)\quad \frac{\partial E}{\partial Y_1^{(2)}} = \delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}. \tag{4.24}$$

Um detalhe importante é que os pesos sinápticos dos neurônios da terceira camada que recebem a saída do primeiro neurônio da segunda camada, pesos $P_{1,1}^{(3)}$ e $P_{2,1}^{(3)}$ respectivamente, já estão atualizados e influenciarão diretamente na atualização dos pesos sinápticos dos neurônios da segunda camada aos quais estão conectados.

Unindo os fatores 1), 2) e 3), pode-se reescrever o gradiente aplicado na função E para o peso sináptico $P_{1,1}^{(2)}$ da segunda camada como:

$$\frac{\partial E}{\partial P_{1,1}^{(2)}} = \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_1^{(1)}. \tag{4.25}$$

A manipulação foi longa, mas, realizando passos similares e alterando somente os índices envolvidos, é possível escrever as variações no erro quadrático em função dos pesos sinápticos para o primeiro neurônio da segunda camada através das equações (4.26):

$$\begin{aligned}
\frac{\partial E}{\partial P_{1,1}^{(2)}} &= \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_1^{(1)} \\
\frac{\partial E}{\partial P_{2,1}^{(2)}} &= \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_2^{(1)} \\
\frac{\partial E}{\partial P_{3,1}^{(2)}} &= \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_3^{(1)}.
\end{aligned} \tag{4.26}$$

Como o fator $\left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)})$ é comum em todas os ajustes que serão realizados, pode-se chamá-lo de $\delta_1^{(2)}$ e reescrever as equações na Equação (4.26) em função do peso a ser atualizado e renomeá-la como o fator de correção destes pesos, ponderado pela taxa de aprendizado η, como na Equação (4.27):

$$\Delta P_{1,j}^{(2)} = \eta \cdot \delta_1^{(2)} \cdot Y_j^{(1)}. \tag{4.27}$$

Seguindo os mesmos passos para o segundo neurônio da segunda camada, é possível obter o fator de ajustes de seus pesos sinápticos como na Equação (4.28):

$$\Delta P_{2,j}^{(2)} = \eta \cdot \delta_2^{(2)} \cdot Y_j^{(1)}, \tag{4.28}$$

em que $\delta_2^{(2)} = \left(\delta_1^{(3)} P_{1,2}^{(3)} + \delta_2^{(3)} P_{2,2}^{(3)}\right) \cdot f'(u_2^{(2)})$.

E assim, os pesos sinápticos dos neurônios da segunda camada serão ajustados de acordo com a Equação (4.29)

$$P_{i,j}^{(2)(novo)} = P_{i,j}^{(2)(antigo)} + \eta \cdot \delta_i^{(2)} \cdot Y_j^{(2)}. \tag{4.29}$$

4.3.2.3 Ajustando os pesos dos neurônios da primeira camada

De modo similar ao que foi feito com o primeiro neurônio da segunda camada, para o primeiro neurônio da primeira camada também serão detalhadas todas as etapas de aplicação de gradiente em função dos pesos sinápticos $P_{1,0}^{(1)}$,$P_{1,1}^{(1)}$,$P_{1,2}^{(1)}$ e $P_{1,3}^{(1)}$ ligados a este neurônio e a utilização de uma função adequada para a regra da cadeia em função destes pesos sinápticos.

Em resumo, o fator de ajustes $\delta_1^{(1)}$ para o peso sináptico $P_{1,1}^{(1)}$ do primeiro neurônio da primeira camada pode ser expresso como na Equação (4.30).

$$\delta_1^{(1)} = \left(\delta_1^{(2)} \cdot f'(u_1^{(2)}) \cdot P_{1,1}^{(2)} + \delta_2^{(2)} \cdot f'(u_2^{(2)}) \cdot P_{2,1}^{(2)}\right) f'(u_1^{(1)}) \cdot x_1, \tag{4.30}$$

que foi obtido por meio de

$$\nabla E^{(1)} = \frac{\partial E}{\partial P_{1,1}^{(1)}} = \frac{\partial E}{\partial Y_1^{(1)}} \frac{\partial Y_1^{(1)}}{\partial u_1^{(1)}} \frac{\partial u_1^{(1)}}{\partial P_{1,1}^{(1)}}. \tag{4.31}$$

Assim, estendendo para todos os pesos sinápticos do primeiro neurônio na primeira camada o que foi realizado, pode-se atualizá-los de acordo com as equações de (4.32):

$$\begin{aligned} P_{1,1}^{(1)(novo)} &= P_{1,1}^{(1)(antigo)} + \eta \cdot \delta_1^{(1)} \cdot x_1 \\ P_{1,2}^{(1)(novo)} &= P_{1,2}^{(1)(antigo)} + \eta \cdot \delta_1^{(1)} \cdot x_2. \end{aligned} \tag{4.32}$$

Generalizando para todos os pesos sinápticos de todos os neurônios da primeira camada, é possível escrever sua regra de aprendizado como na Equação (4.33):

$$P_{i,j}^{(1)(novo)} = P_{i,j}^{(1)(antigo)} + \eta \cdot \delta_i^{(1)} \cdot x_j. \tag{4.33}$$

A Equação (4.33) contém x_j, em que j varia de 1 a 2, representando as variáveis de entradas de uma amostra específica.

Construiu-se a Tabela 14 apresentando os fatores utilizados na correção dos pesos sinápticos $P_{1,1}^{(c)}$ dos primeiros neurônios em cada camada c.

Estão identificados, em azul, os fatores de correção da terceira camada, em verde, os fatores de correção na segunda camada, e em vermelho, os fatores de correção da primeira camada juntamente com suas respectivas saídas dos neurônios, ou seja, cada cor indica a camada a qual aquelas informações pertencem.

$P_{1,1}^{(c)}$		
Camada (c)	Fator de correção	Dependência
3	$\delta_1^{(3)}$	$(d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) \cdot Y_1^{(2)}$
2	$\delta_1^{(2)}$	$\left(\delta_1^{(3)} \cdot P_{1,1}^{(3)} + \delta_2^{(3)} \cdot P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_1^{(1)}$
1	$\delta_1^{(1)}$	$\left(\delta_1^{(2)} \cdot f'(u_1^{(2)}) \cdot P_{1,1}^{(2)} + \delta_2^{(2)} \cdot f'(u_2^{(2)}) \cdot P_{2,1}^{(2)}\right) f'(u_1^{(1)}) \cdot x_1$

Tabela 14 – Tabela apresentando os fatores de correção utilizados nos primeiros neurônios de cada camada. Elaborada pelo autor.

A notação matemática para todo este processo pode, em um primeiro momento, parecer excessiva e confusa. Porém, o que deve ser absorvido disto tudo é a forma como os ajustes são realizados nos pesos sinápticos e o modo que estes ajustes dependem daqueles

que já foram realizados nas camadas seguintes. A Figura 46 apresenta as relações existentes entre os fatores de correção $\delta_i^{(c)}$.

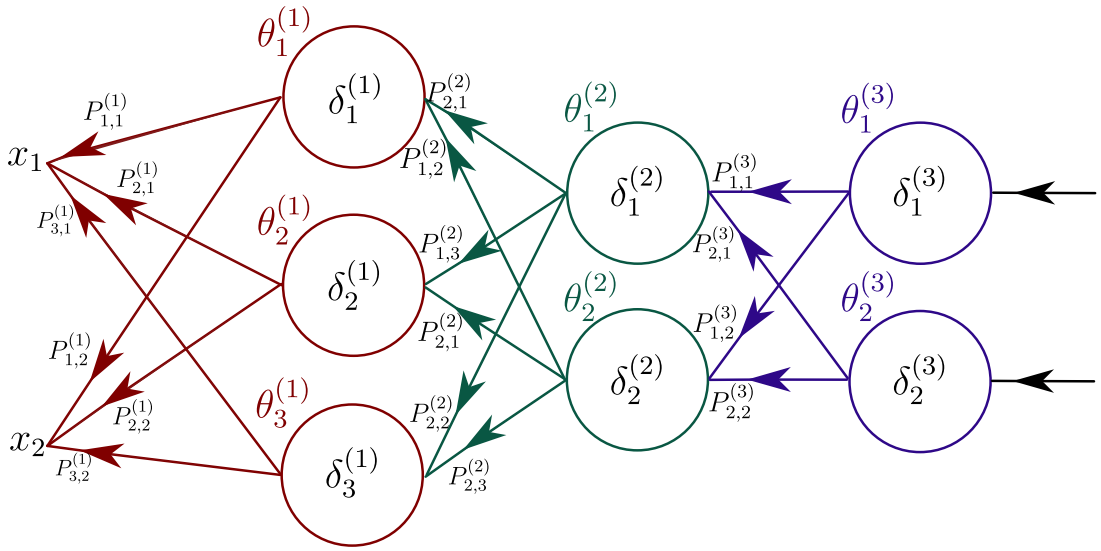


Figura 46 – Disposição dos fatores de correção obtidos na fase *backward* em seus respectivos neurônios. Elaborado pelo autor.

Por meio da Figura 46 é possível observar que, para ajustar o peso sináptico dos neurônios da primeira camada, deve-se primeiro passar pelos ajustes realizados com os pesos dos neurônios que receberão as ponderações realizadas por ele na fase *forward*.

Realizados os ajustes nos pesos sinápticos de todos os neurônios da primeira camada, é encerrada a fase *backward* e o *backpropagation* conclui uma época de treinamento. A partir de então, mantendo o treinamento em linha, outras variáveis do conjunto de entradas são fornecidas à rede e o processo todo, a fase *forward* seguida da *backward*, se repete.

O *backpropagation* não possui um critério universal de parada. Isto significa que o treinamento pode ser parado ao estabelecer-se um número finito de épocas, na observância da acurácia da rede, ou seja, de quantas saídas são apresentadas corretamente ao se comparar com as saídas desejadas, ou algum outro método escolhido.

De fato, Silva, Spatti e Flauzino (2016), Haykin (2001) e Géron (2019) apresentam várias modificações e aperfeiçoamentos para casos específicos de treinamentos envolvendo o *backpropagation*. De um modo geral, é esperado uma convergência da rede para valores ótimos dos pesos sinápticos, ou seja, que os pesos obtidos permitam que a rede efetue com excelência o papel ao qual foi aplicada.

Ainda é possível citar Widrow e Lehr (1990) em que

> "Os algoritmos iterativos descritos neste documento foram todos projetados de acordo com um único princípio subjacente. Essas técnicas (...) baseiam-se no princípio da perturbação mínima: Adaptar-se para reduzir o erro de saída do padrão de treinamento atual, com o mínimo de perturbação das respostas já aprendidas." (Widrow; Lehr, 1990, p.1423, tradução nossa).[4]

4.4 Implementando o Perceptron Multicamadas em Python

A implementação de uma rede PMC em *Python* pode ser um tanto desafiadora. As linhas de código necessárias para uma implementação de modo generalizado, permitindo ajustes na quantidade de neurônios e camadas com o uso de parâmetros e com poucas modificações no código tomaria uma boa parte desta seção. Devido a isto opta-se por utilizar uma biblioteca que contenha uma classe, ou função, que execute o treinamento.

A biblioteca `scikit-learn` (Pedregosa et al., 2011) contém a classe `MLPClassifier`. Esta classe é utilizada em situações envolvendo a classificação dos dados em categorias. Esta biblioteca também fornece a classe `MLPRegressor`, voltado para uso em problemas envolvendo a interpolação dos dados das amostras por uma superfície, pois as saídas desta classe produzem valores contínuos (ver Figura 47a).

A classe `MLPClassifier` produz saídas discretas e portanto é, dentre as duas, a mais adequada para o contexto de categorização. Esta classe não utiliza a função erro quadrático E em seu algoritmo, mas sim a função Regressão Logística[5] L . Primeiramente as saídas da função L são utilizadas como probabilidades $y_{i,j}$ que se relacionarão com as respectivas saídas desejadas $d_{i,j}$ de acordo com a Equação (4.34):

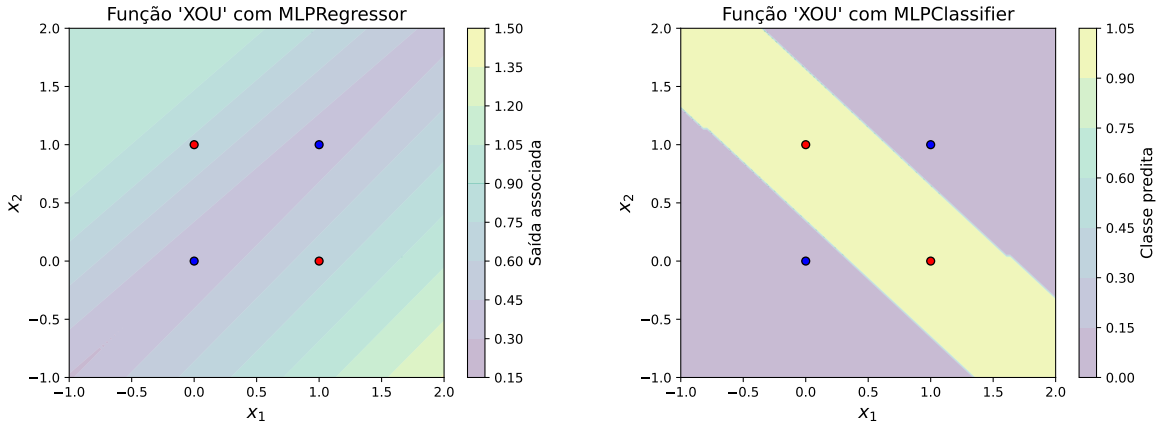
$$L(P) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\left[d_{ij}\log(y_{i,j}) + (1-d_{ij})\log(1-y_{i,j})\right]. \tag{4.34}$$

Na Equação (4.34), m é o número de amostras, n é o número de variáveis por amostras, i representa a amostra em questão e j sua respectiva saída. Mais sobre a função Regressão logística pode ser visto em Mãos à Obra Aprendizado de Máquina com Scikit-Learn & TensorFlow (2019).

Na prática, aderindo a esta substituição da função E por L, sua otimização por meio do algoritmo de treinamento *backpropagation* permitirá obter ao invés de uma superfície, fronteiras de decisão, ao se utilizar o `MLPClassifier` (ver Figura 47b).

[4] No original, sem edições, "The iterative algorithms described in this paper are all designed in accord with a single underlying principle. These techniques-the two LMS algorithms, Mays's rules, and the Perceptron procedure for training a single Adaline, the MRI rule for training the simple Madaline, as well as MRII, MRIII, and backpropagation techniques for training multilayer Madalines-all rely upon the principle of minimal disturbance: Adapt to reduce the output error for the current training pattern, with minimal disturbance to responses already learned."

[5] Também chamada de *log loss*.



(a) Função booleana "XOU" implementada com `MLPRegressor`.

(b) Função booleana "XOU" implementada com `MLPClassifier`.

Figura 47 – Função booleana "XOU" implementada com as classes `MLPRegressor` e `MLPClassifier`. Elaborados pelo autor.

4.4.1 Implementação da função booleana "XOU"

Com o uso desta biblioteca e estas duas classes implementa-se, por exemplo, a função booleana "XOU". A implementação que é realizada neste trabalho foi utilizada para gerar a Figura 47.

Primeiramente, realiza-se a importação as bibliotecas, como no Algoritmo 11:

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import accuracy_score
```

Algoritmo 11 – Importando as bibliotecas para o treinamento do PMC em Python. Elaborada pelo autor.

Em seguida, definem-se as matrizes: x das entradas e d das saídas desejadas, como no Algoritmo 12:

```
# Definindo a tabela verdade XOU em 2 dimensões
x = np.array([[0,0],[0,1],[1,0],[1,1]])
# Saída esperada: 1 quando há um número ímpar de 1 na entrada.
d = np.array([0,1,1,0])
```

Algoritmo 12 – Fornecendo as matrizes de entradas x e de saídas desejadas d para o treinamento do PMC. Elaborada pelo autor.

Utilizando inicialmente a classe `MLPRegressor`, que permite a customização de

diferentes parâmetros internos[6]. Para este estudo de caso serão configurados: a função de ativação; o estado inicial dos pesos sinápticos; como será a otimização realizada pelo *backpropagation*; e o número máximo de iterações da rede. A sequência de comandos apresentada no Algoritmo 13 resumem os parâmetros de inicialização da rede.

```
pmc = MLPRegressor(
    #uma camada escondida com 2 neurônios
    hidden_layer_sizes=(2),
    #função de ativação
    activation='relu',
    #Modo que os gradientes das funções serão utilizados
    solver='sgd',
    #Define o bloco de treinamento do tamanho das amostras
    batch_size=len(x),
    #Semente aleatória para os pesos sinápticos
    random_state=28,
    #Define o número máximo de iterações
    max_iter=20000,
    #Fornece o progresso do treinamento.
    verbose= True
                )
```

Algoritmo 13 – Configurações dos parâmetros do `MLPRegressor`. Elaborado pelo autor.

Alguns métodos e parâmetros de ajustes da rede neural são mais eficientes em determinados contextos, como pode ser visto na documentação da classe `MLPCLassifier`, e também levam em consideração a estratégia de treinamento adotada, se é em lotes, mini-lotes ou em linha, de acordo com os dados fornecidos. Aqui será adotado o treinamento em blocos, ou lotes, e para isto será utilizado o parâmetro `batch_size=len(x)`, que determina o tamanho dos blocos igual ao número de amostras fornecido por `len(x)`.

Fornecer a semente aleatória `random_state` permite que os resultados sejam reproduzíveis com os mesmos parâmetros. Mesmo em 1990, no artigo 30 years of adaptive neural networks: perceptron, Madaline, and backpropagation (1990), os autores já mencionam o quanto os parâmetros iniciais interferem no treinamento da rede como um todo. Há indicações de que determinadas características de conjuntos de pesos sinápticos iniciais favorecem ou não a convergência da RNA em questão e, adotar um parâmetro fixo para `random_state` nos permite reproduzir os resultados obtidos com os pesos iniciais em condições semelhantes todas as vezes que o código for executado.

[6] Uma descrição completa desta classe pode ser vista em https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html. Acessado em 25 de fevereiro de 2025.

De fato, o ajuste dos parâmetros, bem como o número de camadas e neurônios, ainda são objetos de estudo e a escolha destes se baseia na eficiência que apresentam para um mesmo tipo de problema em condições específicas.

O Algoritmo 14 resume a sequência de comandos para treinamento, teste e verificação da acurácia da rede neural após a implementação da rede, cujo resultado é apresentado no Algoritmo 15.

```
# Treinando a rede
pmc.fit(x, d)
# Testando a rede com as amostras
t = pmc.predict(x)
#Transformando as saídas contínuas em discretas
y=t.round()
#Apresenta as saídas obtidas com os pesos treinados
print("Saídas calculadas:", y)
# Avaliando os acertos da rede
acuracia = accuracy_score(d, y)
#Apresenta a porcentagem de acertos
print(f"Acurácia:{acuracia}*100")
```

Algoritmo 14 – Linhas de código executando o treinamento do `MLPRegressor`. Elaborado pelo autor.

```
Iteration 499, loss = 0.05097980
Training loss did not improve more than tol=0.000100 for 10
consecutive epochs. Stopping.
Saídas calculadas: [0. 1. 1. 0.]
Acurácia :100.0%
```

Algoritmo 15 – Saída apresentada ao executar o código fornecido para o treinamento com o `MLPRegressor`. Elaborada pelo autor.

Os resultados após o treinamento podem ser visualizados na Figura 47a. Esta figura apresenta uma série de curvas de nível, separando o plano que contém as entradas em diferentes regiões. Isto ocorreu devido ao fato da saída do `MLPRegressor` ser contínua.

As linhas de código utilizadas para gerar as imagens 47 não serão apresentadas no texto, porém estão disponíveis no caderno do Jupyter Notebook, fornecido em conjunto a este trabalho.

Para implementar a função "XOU", na classe `MLPClassifier`, é possível simplesmente trocar o nome da classe no algoritmo apresentado no Algoritmo 13 e escrever o código para o treinamento do PMC como no Algoritmo 16. A escolha da função de

ativação também depende da finalidade a qual a RNA será aplicada, e isto demanda uma inspeção detalhada por parte do executor (Géron, 2019). Neste caso, foi adotada a função tangente hiperbólica pois foi a que produziu melhores resultados neste treinamento, além da mudança na semente aleatória.

```
pmc = MLPClassifier(
    #uma camada escondida com 2 neurônios
    hidden_layer_sizes=(2),
    #função de ativação
    activation='tanh',
    #Modo que os gradientes das funções serão utilizados
    solver='sgd',
    #Define o bloco de treinamento do tamanho das amostras
    batch_size=len(x),
    #Semente aleatória para os pesos sinápticos
    random_state=4,
    #Define o número máximo de iterações
    max_iter=20000,
    #Fornece o progresso do treinamento.
    verbose= True
                    )
```

Algoritmo 16 – Configurações dos parâmetros do `MLPClassifier`. Elaborado pelo autor.

É possível então executar o mesmo algoritmo conforme descrito no Algoritmo 13 com uma pequena alteração no que diz respeito ao arredondamento das saídas produzidas pela rede, que não será mais necessário, e apresentar a sequência de instruções para o treinamento como no Algoritmo 17.

```
# Treinando a rede
pmc.fit(x, d)
# Testando a rede com as amostras
y = pmc.predict(x)
#Apresenta as saídas obtidas com os pesos treinados
print("Saídas calculadas:", y)
# Avaliando os acertos da rede
acuracia = accuracy_score(d, y)
#Apresenta a porcentagem de acertos
print("Acurácia:", acuracia)
```

Algoritmo 17 – Linhas de código executando o treinamento do `MLPClassifier`. Elaborado pelo autor.

O treinamento do PMC com esta classe permite gerar a imagem apresentada na Figura 47b, na qual as fronteiras são representadas por linhas delimitando as regiões de mesma cor que apresentam a mesma saída e, ao executar os códigos do Algoritmo 17, a saída apresentada será como no Algoritmo 18.

```
Iteration 3147, loss = 0.25937596
Training loss did not improve more than tol=0.000100 for 10
consecutive epochs. Stopping.
Saídas calculadas: [0 1 1 0]
Acurácia :100.0%
```

Algoritmo 18 – Saída apresentada pelo PMC que implementa a função booleana "XOU" em duas dimensões. Elaborado pelo autor.

4.4.2 Utilizando o PMC em três categorias: As flores Íris

A possibilidade de um PMC conter em sua camada de saída mais de um neurônio, implica em uma amostra poder ser classificada para uma dentre três, ou mais, categorias. Esta situação pode ser exemplificada com os dados das flores Íris, contidos em Fisher (1988).

Como foi discutido na seção 2.2, existem três categorias neste conjunto de dados, sendo duas destas não-linearmente separáveis, e é possível realizar o treinamento do PMC com os dados relativos somente às pétalas ou às sépalas das flores, de modo que se possa apresentar a visualização das fronteiras de decisão que esta RNA estabelecerá a partir dos pesos sinápticos treinados em cada caso.

Para utilizar as flores pode-se adotar a arquitetura para o PMC como no Algoritmo 19 que contém duas camadas ocultas com quatro neurônios cada. Para a camada de saída, a classe `MLPClassifier` atribuirá automaticamente três neurônios, um para cada categoria das flores.

```
#Duas camadas com 4 neurônios cada
pmc = MLPClassifier(hidden_layer_sizes=(4,4),
    #Máximo de iterações
    max_iter=10000,
    #função de ativação tangente hiperbólica
    activation='tanh',
    #Define o uso do Gradiente descendente estocástico
    solver='sgd',
    #Define o bloco de treinamento do tamanho da matriz de amostras
    batch_size=len(x),
    #Define a tolerância da diferença dos erros em épocas sucessivas
    tol=0.00001,
    #Define a semente aleatória para inicialização dos pesos sinápticos
    random_state=5
                    )
```

Algoritmo 19 – Configuração do `MLPClassifier` para treinar o PMC com os dados das flores Íris. Elaborado pelo autor.

A matriz de amostras x contendo os respectivos dados das medidas das flores, em centímetros, pode ser construída a partir dos dados fornecidos em Fisher (1988) ou por meio do comando `sklearn.datasets.load_iris()`[7]. Estes dados também são apresentados, explicitamente, no respectivo caderno interativo Jupyter Notebook que tem o link fornecido no capítulo 5, que também pode ser acessado pelo site Oliveira (2025).

A matriz de saídas desejadas d pode ser construída de forma similar à matriz de amostras x e, para a realização deste treinamento, a identificação de cada Íris Setosa, Íris Versicolor e Íris Virgínica foi alterada, respectivamente, para 0, 1 e 2. Nos dados disponibilizados, as primeiras cinquenta amostras são de Íris Setosa, seguidas por outras cinquenta de Íris Versicolor, e por último as cinquenta amostras de Íris Virgínica

Após o fornecimento da matriz x de amostras, utiliza-se o Algoritmo 20 para treinar o PMC.

[7] Como pode ser visto detalhadamente em: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html. Acessado em 25 de fevereiro de 2025.

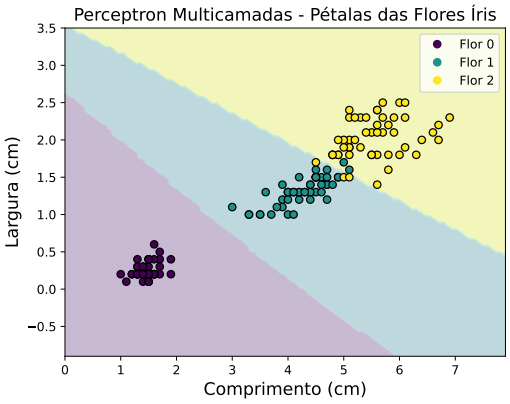
```
# Treinando a rede
pmc.fit(x,d)
# Fazendo previsões
y = pmc.predict(x)
# Avaliando a acurácia
acuracia = accuracy_score(d, y)
# Variável auxiliar 'erro' é uma matriz
erro=d-y
# Reorganização das dimensões da matriz 'erro'
erros_organizados=erro.reshape((3,50))
# Reorganização das dimensões da matriz de saídas y
y_organizados=y.reshape((3,50))
#Apresentação dos dados resultates do treinamento:
#Percentual de acertos do PMC
print(f"Acurácia:{acuracia*100}%")
#Saídas obtidas com os pesos sinápticos treinados
print("Saídas calculadas:\n", y_organizados)
#Diferença entre saídas desejadas e treinadas
print("Erros de categorização\n", erros_organizados)
```

Algoritmo 20 – Treinamento do PMC com o `MLPClassifier` para os dados das flores Íris. Elaborado pelo autor.

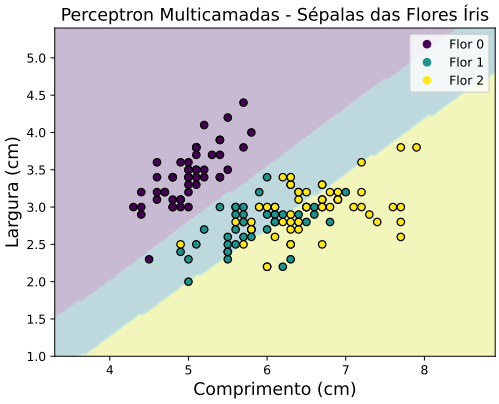
Utilizando somente os dados relativos às medidas de comprimento e largura das pétalas das flores, o PMC como configurado no Algoritmo 19 com as saídas apresentadas pela rede de acordo com o Algoritmo 20, permite que a rede atinja 96% de acurácia. Ao fornecer somente as medidas das sépalas, é alcançada uma acurácia de, aproximadamente, $76,67\%$. Utilizando como amostras as medidas das sépalas e pétalas, a rede apresenta 98.0% de acurácia. Em todos os casos é possível perceber que os equívocos que a rede comete são em relação às classificações das flores Virgínica e Versicolor, que não são linearmente separáveis.

Com os pesos sinápticos treinados, é possível ainda apresentar uma visualização das fronteiras de decisão para cada um dos três casos, como apresentado na Figura 48.

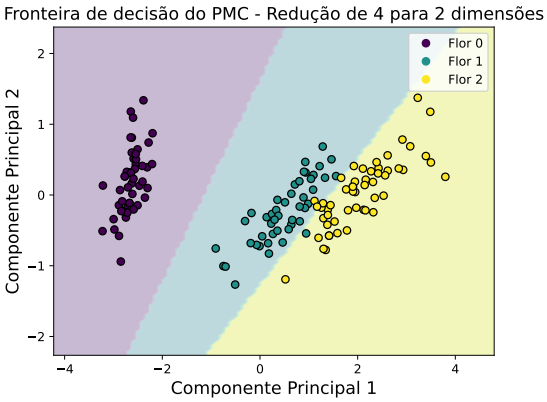
As vantagens em se utilizar somente as pétalas ou sépalas residem no fato dos dados poderem ser dispostos diretamente num plano e a visualização destes ser interpretada diretamente com o modo em que são apresentados, como pode ser visto nas Figuras 48a e 48b.



(a) PMC treinado somente com as pétalas.



(b) PMC treinado somente com as sépalas.



(c) PMC treinado com as pétalas e sépalas.

Figura 48 – PMC treinado com os dados das flores Íris. Elaborados pelo autor.

Ao se utilizar todos os dados de todas as amostras, perde-se o poder de visualização devido ao número de dimensões associada a eles. Em uma proposta para representar os dados em duas dimensões, é possível utilizar como auxílio da Análise dos Componentes Principais (ACP), um recurso que também é fornecido pela biblioteca `scikit-learn`[8] e utilizado para visualização de dados multidimensionais em dimensões reduzidas, como pode ser visto na Figura 48c.

As duas classes e a biblioteca utilizada nestes exemplos não são as únicas possibilidades para implementar um PMC em Python. Além da implementação manual, existem outras bibliotecas como o `Keras` ou o `PyTorch`, que cada uma com seu modo, permitem outras implementações dos parâmetros de configuração do PMC.

[8] "Principal Component Analysis". https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html#sphx-glr-auto-examples-decomposition-plot-pca-iris-py. Acessado em 25 de fevereiro de 2025.

5 Descrição dos materiais produzidos

De modo a complementar este texto teórico, foram elaboradas videoaulas que reapresentam os exemplos discutidos no decorrer nos capítulos desta dissertação, visando suprir as limitações de um texto estático ao serem adotados recursos pertinentes às tecnologias digitais dos vídeos e trazendo mais dinamismo aos conceitos abordados, e também cadernos IPython contendo os códigos de programação em linguagem Python, desenvolvidos para que haja uma maior interação, por parte do leitor, com as redes neurais mencionadas neste trabalho.

Todos os materiais elaborados estão disponibilizados na página do autor deste trabalho, que pode ser acessada em Oliveira (2025). Este site existe com o único propósito de organizar as videoaulas, os capítulos da dissertação e os cadernos de acordo com o assunto abordado, facilitando uma integração entre estes formatos apresentados. Nada será adicionado a este site além das descrições realizadas neste trabalho e, portanto, o conteúdo deste está completamente de acordo e limitado com o que foi escrito e descrito aqui.

5.1 Descrição dos vídeos

Além dos exemplos apresentados no decorrer deste texto, encontram-se também exemplos exclusivos a serem explorados em conjunto com os recursos visuais que os vídeos possam permitir, possibilitando uma abordagem facilitadora ao entendimento, por parte do leitor e agora também ouvinte, destes conceitos e assuntos.

Os vídeos estão listados na ordem em que devem ser assistidos, juntamente com sua duração e uma breve descrição de seu conteúdo, resumidos na Tabela 15, onde encontram-se também os links para que possam ser assistidos online.

nº	Título	Tempo	Descrição	Link
1º	Introdução às Redes Neurais Artificiais: O Perceptron, o Adaline e o Perceptron Multicamadas	4:58	Apresentação do autor e dos temas que serão abordados neste minicurso.	https://drive.google.com/file/d/1pqOgKQh7MFR3lfiCUthUn9KhnoTUSG40/preview
2º	Perceptron - A1: Introdução ao funcionamento do Perceptron	4:04	Implementação do Neurônio de McCulloch-Pitts e caracterização do Perceptron como classificador linear que pode ter seus pesos sinápticos treinados e ajustados automaticamente por um algoritmo de treinamento.	https://drive.google.com/file/d/1PkcLqaQjKD8HcxNbkrjedx6UPDt8uL8f/preview
3º	Perceptron - A2: O treinamento do Perceptron	5:51	Apresentação da regra de treinamento a qual realiza os ajustes dos pesos sinápticos do Perceptron e de seus critérios de convergência.	https://drive.google.com/file/d/19ulrDMglVpTabFiP3aOQNX7_JUR_la2y/preview
4º	Perceptron - A3: Os fundamentos matemáticos do funcionamento do Perceptron	4:47	Consolidação da notação matemática que será utilizada no decorrer deste minicurso.	https://drive.google.com/file/d/1SJSuAuP8JTfvpY25taAp351j6VcMCJ82/preview
5º	Perceptron - A4: Implementando o Perceptron em Python	14:37	Implementação e execução dos exemplos apresentados neste texto, e de outros, em linguagem de programação Python e visualização do treinamento efetuado pelo Perceptron por meio de gráficos interativos.	https://drive.google.com/file/d/1mb4pY0cdCkhnzq3AVWPYkriRulVozybY/preview
6º	Perceptron - A5: Sistemas Lineares e o Perceptron	5:39	Neste vídeo é realizada uma pequena revisão sobre sistemas lineares para que se possa explorar os sistemas lineares sobredeterminados.	https://drive.google.com/file/d/1JK4lStI3VmjJebv2yw3Ms0sw9fgjFM0r/preview

nº	Título	Tempo	Descrição	Link
7º	Adaline - A1: Introdução ao Adaline	3:16	Apresentação do Adaline e de sua regra de aprendizagem, a regra Delta, ou Gradiente descendente.	https://drive.google.com/file/d/1G9-xjpGXnB_PE8KFIT84UzNSPaX7yGaY/preview
8º	Adaline - A2: O treinamento do Adaline	2:29	Apresentação do processo de treinamento do Adaline e de seu critério de convergência.	https://drive.google.com/file/d/1mP08CG4o-ygxewPuqMfep4r93F7zZ5-o/preview
9º	Adaline - A3 : Implementando o Adaline em Python	12:42	Implementação do Adaline em linguagem de programação Python e execução do treinamento com exemplos.	https://drive.google.com/file/d/1_2dxOkUHtXkCAongyt0WgehkwYGQaTRJ/preview
10º	Adaline - A4: Comparando o Perceptron e o Adaline	3:44	Neste vídeo compara-se o Perceptron e o Adaline evidenciando suas diferenças, vantagens e desvantagens um em relação ao outro, em determinados contextos.	https://drive.google.com/file/d/1Mch4hXYj1C1joazgVtABlCv3oyJq0vqa/preview
11º	Perceptron Multicamadas (PMC): Uma introdução ao funcionamento do PMC	3:29	Apresentação do Perceptron Multicamadas e de sua estrutura interna de funcionamento.	https://drive.google.com/file/d/1In64kPmpgaIwuJ-8tJ8dc-sKRiLmzBTF/preview
12º	Perceptron Multicamadas (PMC): Uma introdução ao treinamento do PMC	4:36	Apresentação do algoritmo de treinamento *backpropagation* com o uso da função Erro Quadrático Médio.	https://drive.google.com/file/d/10P-dSXdOCsDtPq3l20nT2pqc9Odiiazb/preview
13º	Perceptron Multicamadas (PMC): Aplicações do PMC	8:11	Implementação do PMC em Python e utilização em exemplos já apresentados no texto.	https://drive.google.com/file/d/1JRNVjuR_wtWzx6QjKjpnhUAMQEb7vZH_/preview

Tabela 15 – Tabela listando a ordem das videoaulas, título, duração, conteúdo e links para assisti-las. Elaborada pelo autor.

5.2 Cadernos interativos "Jupyter Notebooks"

Com a possibilidade de interação com o treinamento de redes neurais, todos os códigos que estão descrito neste trabalho, juntamente com os que foram apresentados nos vídeos, estão disponibilizados para execução e modificação por parte do leitor. Estes códigos estão organizados em arquivos `IPyhton`, com extensão de arquivo `.ipynb`, que é lido localmente, de modo offline, pelo software Jupyter Notebook. Este método de acessar os cadernos necessita que já estejam instalados o Python em seu computador e também o próprio Jupyter Notebook[1].

De modo alternativo, estes cadernos podem ser executados e modificados de modo online. Uma das plataformas disponíveis para este fim é o Google Colab que pode ser acessado no site oficial[2]. Se o leitor desejar manter uma cópia deste caderno para realizar edições online e salvá-las, basta que possua uma conta Google e copiar os arquivos fornecidos para ela.

Ainda se tratando de ferramentas online, é possível que o leitor utilize estes cadernos nos navegadores de internet Firefox (versões 90 em diante) e Google Chrome (Chromium versões 89 em diante). Dos vários sites existentes que implementam o Jupyter Notebook, recomenda-se um em que foi desenvolvido pela própria equipe responsável pelo desenvolvimento do software em seu site oficial[3]

Foram desenvolvidos três cadernos, de modo que cada um deles aborda uma das três Redes Neurais Artificiais discutidas neste texto. Estes cadernos também foram utilizados para a gravação das videoaulas: "Perceptron - A4: Implementando o Perceptron em Python", "Adaline - A3: Implementando o Adaline em Python" e "Perceptron Multicamadas (PMC): Aplicações do PMC". Um quarto caderno foi utilizado na videoaula: "Adaline - A4: Comparando o Perceptron e o Adaline", mas este foi apenas uma modificação temporária no caderno que implementava o Adaline, realizada apenas para gravação do vídeo. Mesmo assim, este caderno também está disponível para download.

O exemplo das flores Íris, que é mencionado nos exemplos deste trabalho, também foram estudados nas videoaulas com o uso de seu conjunto de dados obtidos em Fisher (1988). Devido a sua popularidade, existem modos de importar estes dados com bibliotecas do `Scikit-learn` e do comando `sklearn.datasets.loadiris`. Entretanto, optou-se pela importação manualmente dos dados e, apresentá-los integralmente como forma de deixar evidente o volume de informações que uma rede neural pode processar.

---

[1] A instalação pode ser realizada seguindo as instruções do site oficial que pode ser visitado em `https://jupyter.org/install`. Acessado em 25 de fevereiro de 2025

[2] `https://colab.google.com`. Acessado em 25 de fevereiro de 2025

[3] O site que permite a execução e edição destes cadernos implementado pela equipe do Jupyter é o `https://jupyter.org/try`. Acessado em 25 de fevereiro de 2025.

6 Considerações finais

Este trabalho foi desenvolvido para ser utilizado como uma introdução aos estudos na área de Redes Neurais Artificiais (RNA). Para isto, foram elaboradas 13 videoaulas e quatro cadernos interativos, com o propósito de estabelecer um percurso no qual o leitor, ou ouvinte, poderá conhecer e estudar sobre este modelo computacional amplamente utilizado nas mais diversas áreas da sociedade.

Adicionalmente, as videoaulas foram disponibilizadas na página Oliveira (2025) juntamente com os links para os cadernos Jupyter Notebook utilizados nos vídeos e este material escrito. A adoção desta estratégia se deu com o propósito de relacionar de maneira orgânica as etapas discutidas nos vídeos, sua fundamentação teórica e a possibilidade de interação, por parte do estudante, ao editar e executar os códigos apresentados nos vídeos.

A apresentação dos materiais relacionados a este trabalho, de forma interativa permitiu discutir sobre assuntos que demandariam a ampliação do corpo deste texto, implicando no desvio do assunto principal, sem perder os detalhes que seriam omitidos caso estivessem contidos aqui. Um exemplo deste caso foi o estudo de como os pesos sinápticos iniciais influenciam, ou não, o treinamento das RNA discutidas e a obtenção dos pesos sinápticos finais.

A ampliação do número de exemplos em que as RNA foram aplicadas, por meio do uso dos materiais interativos, foram os principais ganhos do uso de videoaulas. Aliados com a argumentação inicial e apresentação em linguagem simplificada, as videoaulas permitiram a observação, de forma atrativa, dos temas discutidos neste trabalho de forma a incentivar a leitura do texto para a fundamentação teórica.

É esperado que o leitor, e ouvinte, não se contente somente com o que foi abordado neste minicurso como um todo, muito menos que se limite ao consumo somente dos vídeos ou somente do texto, pois estes se complementam. Além disso, algumas discussões omitidas neste trabalho podem surgir naturalmente por questionamentos relacionados ao modo em que as propostas foram realizadas aqui.

Um exemplo disto é a discussão relativa à divisão das amostras de treinamento das flores Íris de 90% para o treinamento e 10% para verificação da rede. A análise desta divisão pode abrir espaço para a discussão de quais seriam as divisões adequadas a outros contextos de treinamento de RNAs, ou mesmo se existe a possibilidade de uma divisão adequada, bem como quais os efeitos da adoção destas divisões no treinamento das redes em se tratando de *overfitting* ou *underfitting*.

Outro exemplo de estudos que podem ser realizados a partir de como foram

abordados os elementos deste trabalho são os tipos de treinamento, em linha, em blocos ou semi-blocos, bem como outras possibilidades, e suas vantagens e desvantagens em frente aos tipos de amostras, à proposta de uso para a rede e à forma como a rede pode convergir para valores que são mínimos locais ou globais.

O próprio uso das diferentes funções utilizadas para calcular, e consequentemente minimizar, o erro da rede, as funções de custo, também é um outro aspecto que não é aprofundado neste trabalho como um todo, e é um tema que deve ser estudado por parte do leitor, e ouvinte, que pretende seguir com estudos nesta área.

A omissão de alguns temas relacionados ao desenvolvimento das RNA ao longo da história, incluindo avanços mais recentes como Redes Neurais Convolucionais, Aprendizado Profundo e inclusive aplicações mais famosas como reconhecimento facial e de escrita, por exemplo, ocorre de forma que estes assuntos se tornem extensões naturais para quem tiver interesse em seguir nesta área de pesquisa, com fontes mais adequadas aos propósitos específicos aos quais o leitor e ouvinte desejar cumprir.

Dito isto, pretende-se deixar evidente que todos os esforços deste trabalho tiveram como objetivo fundamentar temas considerados básicos, na esperança que se tornassem acessíveis e interessantes a todos que tiverem contato com os materiais que produzimos, e que sirvam como forma de ingresso de estudos nesta área.

Bibliografia

ANTON, Howard; RORRES, Chris. **Álgebra Linear com aplicações**. 10. ed. Porto Alegre: Bookman, 2012. P. 576.

ARENALES, Selma Helena Vasconcelos; DAREZZO, Artur. **Cálculo Numérico: aprendizagem com apoio de software**. 2. ed. São Paulo: Cengage Learning, 2016.

BONJORNO, José Roberto; GIOVANNI, José Ruy. **Matemática**: Uma nova abordagem. São Paulo: FTD, 2000. v. 2.

BOTTOU, Léon. **Foreword**. Set. 2017. Perceptrons. Reissue of the 1988 expanded edition. By Marvin L. Minsky and Seymour A. Papert. MIT Press. Cambridge, MA. Disponível em: http://leon.bottou.org/papers/bottou-foreword-2017. Acesso em: 20 nov. 2023.

BRASIL. **Base Nacional Comum Curriular**: A área de Matemática. 2017. Disponível em: http://basenacionalcomum.mec.gov.br/abase/#fundamental/a-area-de-matematica. Acesso em: 11 out. 2023.

_____. **Base Nacional Comum Curriular**: Computação na Educação Básica: Complemento à BNCC. 2022. Disponível em: http://portal.mec.gov.br/docman/fevereiro-2022-pdf/236791-anexo-ao-parecer-cneceb-n-2-2022-bncc-computacao/file. Acesso em: 11 out. 2023.

FISHER, R. A. **Iris**. 1988. DOI: 10.24432/C56C76. Disponível em: https://archive.ics.uci.edu/dataset/53/iris. Acesso em: 25 mar. 2024.

FRANCO, Neide Maria Bertoldi. **Cálculo Numérico**. 8. ed. São Paulo: Pearson Prentice Hall, 2006.

GAD, Ahmed Fawzy; JARMOUNI, Fatima Ezzahra. **Introduction to Deep Learning and Neural Networks with Python™: A Practical Guide**. Academic Press, 2021. ISBN 9780323909334.

GÉRON, Aurélien. **Mãos à Obra Aprendizado de Máquina com Scikit-Learn & TensorFlow**: Conceitos, Ferramentas e Técnicas Para a Construção de Sistemas Inteligentes. 4. ed. Rio de Janeiro: Alta Books, 2019. P. 576.

GOOGLE INC. **Olá, este é o Colaboratory**. 20 nov. 2023. Disponível em: https://colab.research.google.com/. Acesso em: 20 nov. 2023.

HAYKIN, Simon. **Redes Neurais: Princípios e Prática**. Bookman Editora, 2001. ISBN 9788577800865.

IEZZI, Gelson; HAZZAN, Samuel. **Fundamentos de Matemática Elementar**: Sequências, Matrizes, Determinantes e Sistemas Lineares. 8. ed. São Paulo: Atual, 2013. v. 4.

KINSLEY, Harrison; KUKIEŁA, Daniel. **Neural Networks from Scratch in Python**. Harrison Kinsley, 2020.

KOVÁCS, Zsolt László. **Redes Neurais Artificiais: Fundamentos e Aplicações**. 4. ed.: Livraria da Física, 2023. P. 176.

MARTINS, André Oliveira. **Redes Neurais no Ensino Básico**. Jan. 2021. Dissertação – Universidade Federal de Alagoas - UFAL.

MCCULLOCH, Warren S.; PITTS, Walter. **A logical calculus of the ideas immanent in nervous activity**. **The Bulletin of Mathematical Biophysics**, Springer Science e Business Media LLC, v. 5, n. 4, p. 115–133, dez. 1943. ISSN 1522-9602. DOI: 10.1007/bf02478259.

MINSKY, Marvin; PAPERT, Seymour Aubrey. **Perceptrons: An Introduction to Computational Geometry**. Expanded: The MIT Press, 1988.

OLIVEIRA, Victor Rodrigues de. **Descobrindo Redes Neurais Artificiais**: Minicurso sobre os modelos: Perceptron, Adaline e Perceptron Multicamadas. Disponível em: https://oliveiravictor2.github.io/Minicurso_RNAs/. Acesso em: 22 jan. 2025.

PEDREGOSA, F. et al. **Scikit-learn: Machine Learning in Python**. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PYTHON SOFTWARE FOUNDATION. **Python**: Welcome to Python.org. Disponível em: https://www.python.org/. Acesso em: 11 out. 2023.

RIZZATTI, Ivanise Maria et al. **Os produtos e processos educacionais dos programas de pós-graduação profissionais: proposições de um grupo de colaboradores**. **ACTIO: Docência em Ciências**, Universidade Tecnologica Federal do Parana (UTFPR), v. 5, n. 2, p. 1, ago. 2020. ISSN 2525-8923. DOI: 10.3895/actio.v5n2.12657.

RUGGIERO, Márcia Aparecida Gomes; ROCHA LOPES, Vera Lúcia da. **Cálculo Numérico: Aspectos Teóricos e Computacionais**. 2. ed. São Paulo: Pearson Education do Brasil, 1996.

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.

SILVA, Lenardo Chaves e et al. **Applications of convolutional neural networks in education: a systematic literature review**. **Expert Systems with Applications**, 2023. DOI: 10.1016/j.eswa.2023.120621.

SPECTRUM, IEEE. **Top Programming Languages 2024**. 12 nov. 2024. Disponível em: https://spectrum.ieee.org/top-programming-languages-2024. Acesso em: 12 nov. 2024.

TAPPERT, Charles C. **Who Is the Father of Deep Learning? International Conference on Computational Science and Computational Intelligence**, 2019.

TELLES, Eduardo Santos; BARONE, Dante Augusto Couto;
SILVA, Alexandre Moraes da. Inteligência Artificial no Contexto da Indústria 4.0. In: ANAIS do Workshop sobre as Implicações da Computação na Sociedad (WICS 2020). Sociedade Brasileira de Computação, jun. 2020. (WICS). DOI: 10.5753/wics.2020.11044.

UNWIN, Antony; KLEIMAN, Kim. **The Iris Data Set: In Search of the Source of Virginica**. **Significance**, v. 18, n. 6, p. 26–29, nov. 2021. ISSN 1740-9705. DOI: 10.1111/1740-9713.01589. eprint: https://academic.oup.com/jrssig/article-pdf/18/6/26/49188885/sign_18_6_26.pdf. Disponível em: https://doi.org/10.1111/1740-9713.01589.

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.