Victor Rodrigues de Oliveira

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

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

Instituto Federal de Educação Ciência e Tecnologia de São Paulo

Campus São Paulo

Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT

Orientador: Prof. Dr. Marco Aurélio Granero Santos

São Paulo - SP

2025

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Sumário

1 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 1 e 2), 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").

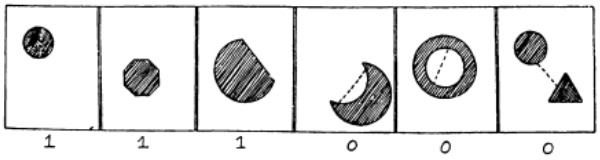


Figura 1 – Classificação em figuras planares convexas e não-convexas que um neurônio de McCulloch-Pitts é capaz de realizar: 1 é para convexas e 0 é para não-convexas.
Fonte: Minsky e Papert (1988, p.6).

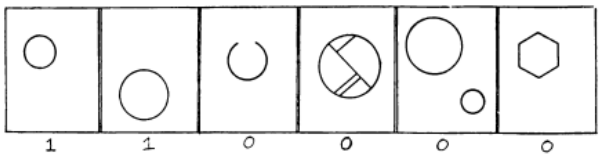


Figura 2 – Classificação que um neurônio McCulloch-Pitts é capaz de realizar para figuras planares como sendo circulares ou não.
Fonte: Minsky e Papert (1988, p.6).

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 3, 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).

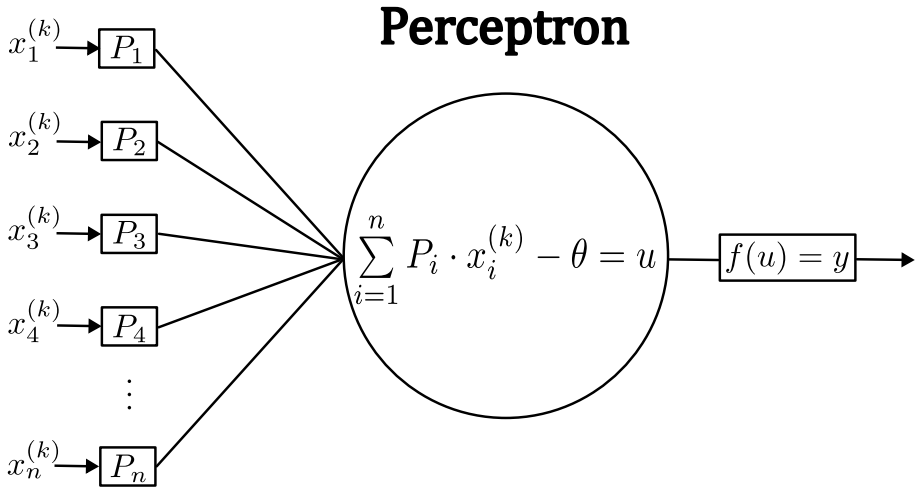


Figura 3 – Representação esquemática em linha do funcionamento do Perceptron. Adaptado de Silva, Spatti e Flauzino (2016).

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.

1.1 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 1.2 apresenta um estudo de caso em que é possível visualizar um conjunto específico de dados.

1.2 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 4) 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 5.

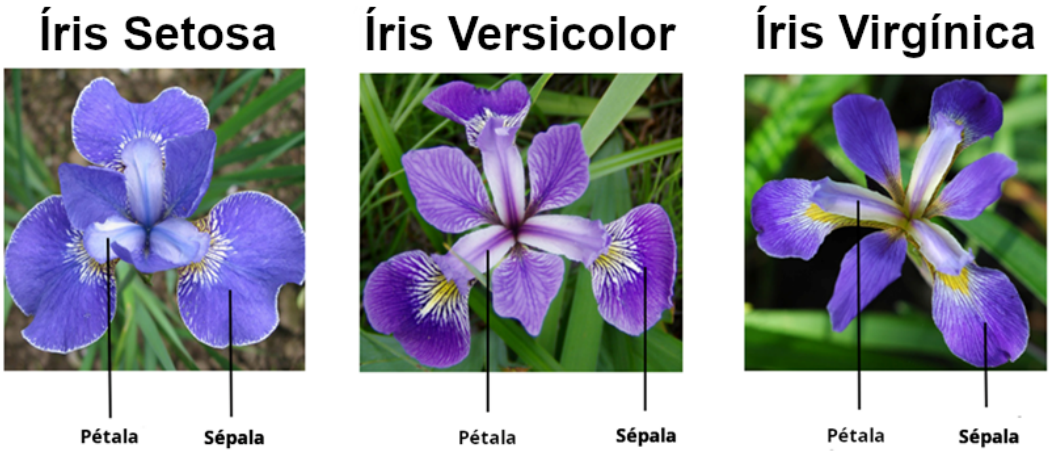


Figura 4 – Os três tipos de flores Íris que foram catalogadas nos dados apresentados em Fisher (1988). Adaptado de https://miro.medium.com/v2/resize:fit:1000/1*Hh53mOF4Xy4eORjLilKOwA.png

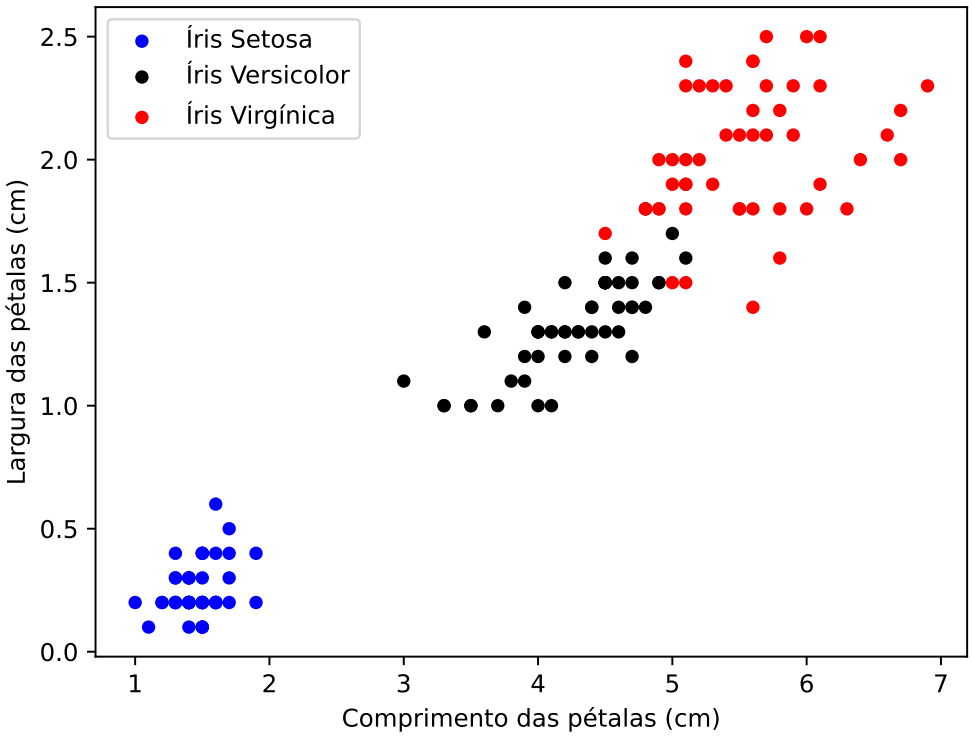


Figura 5 – Gráfico de dispersão da Íris Setosa, Íris Versicolor e Íris Virgínica. Elaborado pelo autor com base nos dados obtidos de Fisher (1988).

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 5 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 6.

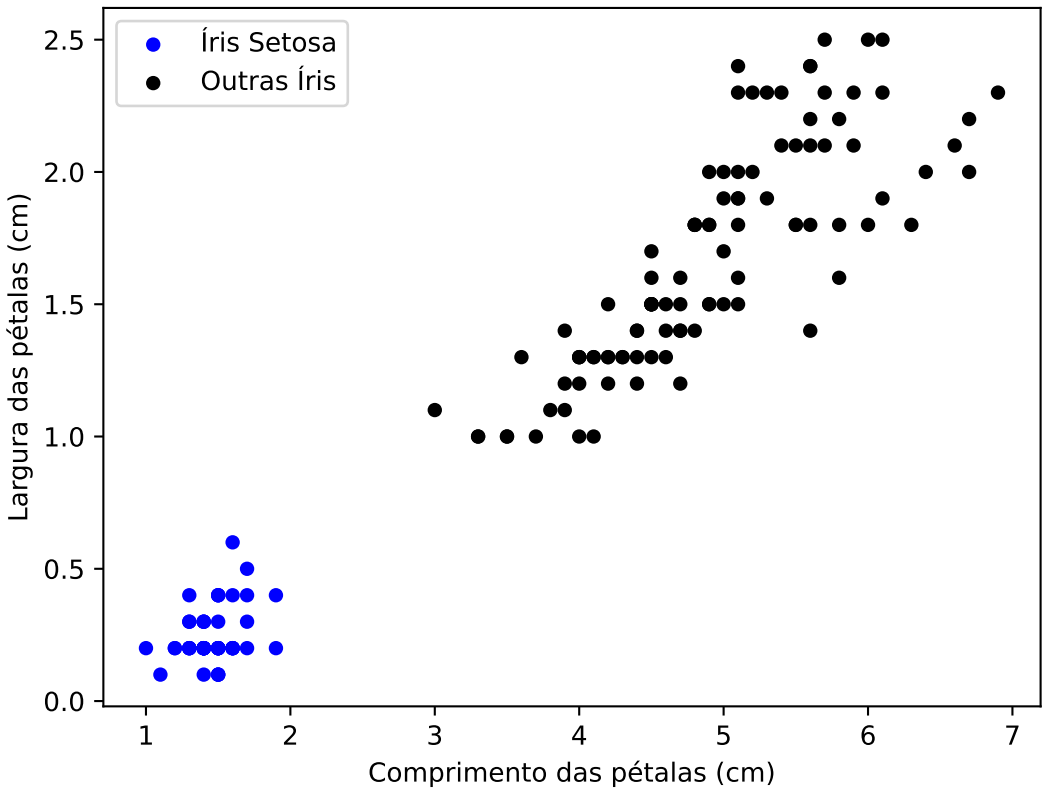


Figura 6 – Gráfico de dispersão da Íris Setosa em relação às outras duas espécies. Elaborado pelo autor.

As Figuras 5 e 6 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 7.

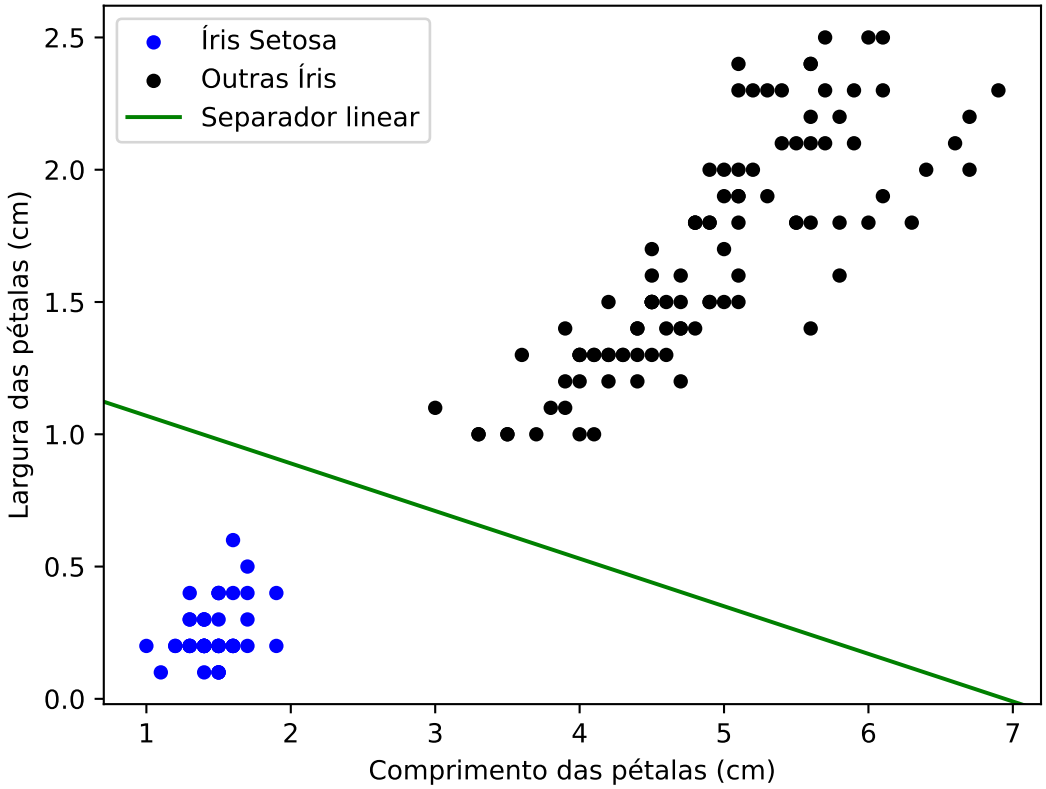


Figura 7 – Reta separando as flores com comprimentos linearmente separáveis. Elaborada pelo autor.

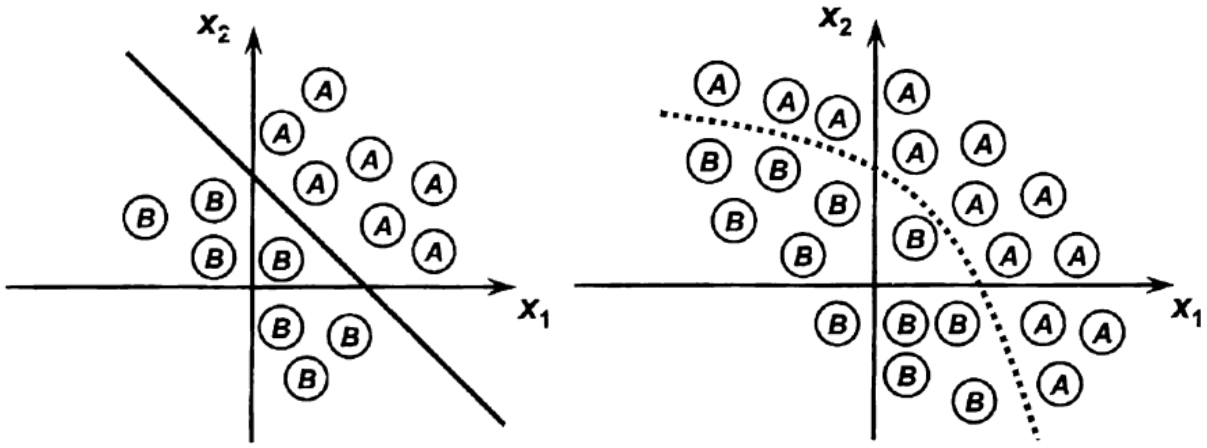
O conjunto de dados das flores apresenta para cada entrada quatro variáveis, as larguras e comprimentos das sépalas e pétalas. A utilização de um número maior de variáveis impossibilita a visualização dos dados da forma como foram coletados e, além disso, se houver uma separabilidade linear, o Perceptron 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 **??**, esta técnica será utilizada para apresentar graficamente os dados do treinamento juntamente com este mesmo conjunto de dados das flores.

1.3 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 8, 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 8a apresenta

uma possibilidade de fronteira linear enquanto a Figura 8b 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 8 – 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 **??** 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 8a 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 8b 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.

1.4 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:

$$P \cdot x, \tag{1.1}$$

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,

$$P \cdot x = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & \cdots & P_{1,n} \end{bmatrix} \cdot \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \cdots & x_{n,m} \end{bmatrix}. \tag{1.2}$$

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 (1.2)

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:

$$P \cdot x - L = \tag{1.3}$$
$$\begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,n} \end{bmatrix}_{1\times n} \cdot \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{bmatrix}_{n\times m} - \begin{bmatrix} \theta & \theta & \cdots & \theta \end{bmatrix}_{1\times m},$$

que resulta em

$$P \cdot x - L = \tag{1.4}$$
$$\begin{bmatrix} (P_{1,1}x_{1,1} + \cdots + P_{1,n}x_{n,1}) - \theta & \cdots & (P_{1,1}x_{1,m} + \cdots + P_{1,n}x_{n,m}) - \theta \end{bmatrix}_{1\times m}.$$

Em cada elemento da matriz apresentada na Equação (1.4) 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 (1.3), podem ser realizadas graças às dimensões compatíveis com o produto matricial. De modo reduzido, tem-se o sistema:

$$\begin{cases} u = P \cdot x - L \\ y = f(u). \end{cases} \tag{1.5}$$

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:

$$P^{(novo)} = P^{(antigo)} + \eta \cdot (d - y) \cdot x^T \tag{1.6}$$

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 9.

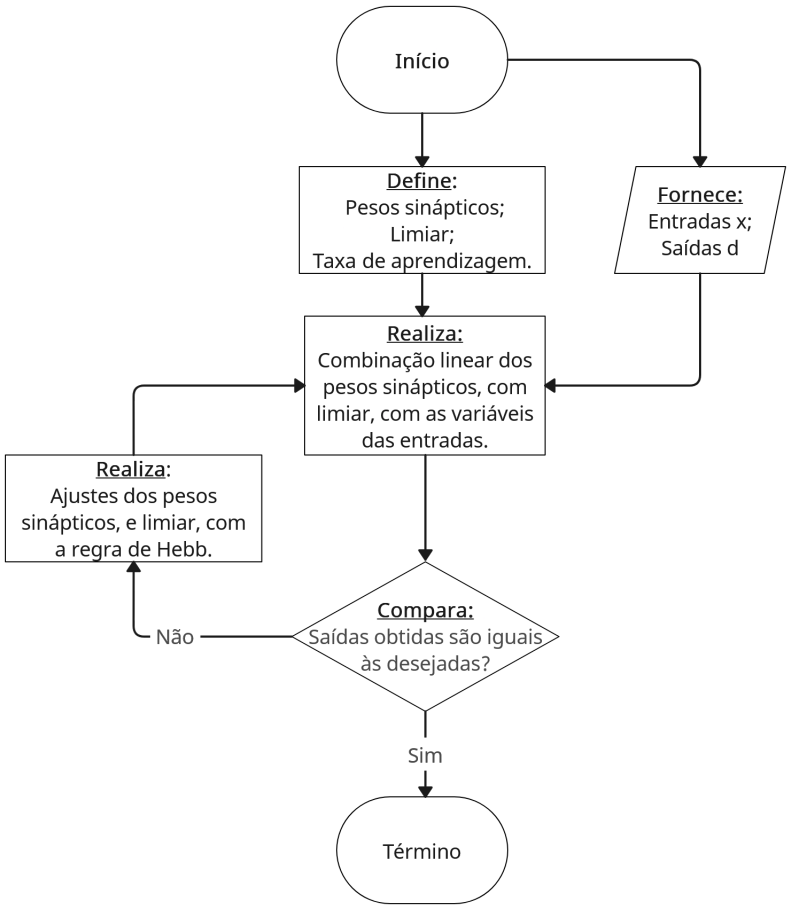


Figura 9 – Fluxograma do treinamento do Perceptron. Elaborado pelo autor.

Será apresentado na seção 1.4.1 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.

1.4.1 Implementando a função booleana "E"

A seção **??** 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 **??**. Observe que os dados para a construção da matriz x são acondicionados de modo a preservar as operações matriciais definidas nas Equações (1.1) a (1.4).

$$x' = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{1.7}$$

A matriz das entradas apresentada na Equação (1.7) 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:

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

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:

$$\begin{aligned} u &= P \cdot x - L \\ &= \begin{bmatrix} P_{11} & P_{12} & \theta \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \end{aligned} \tag{1.9}$$

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 **??**.

Atualizando a Equação (1.9) tem-se:

$$\begin{aligned} u &= \begin{bmatrix} 0,9 & 0,9 & 0,7 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} -0,7 & 0,2 & 0,2 & 1,1 \end{bmatrix}. \end{aligned} \tag{1.10}$$

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:

$$f(u) = \begin{cases} 1, & \text{se } u \geqslant 0, \\ 0, & \text{se } u < 0 \end{cases}, \tag{1.11}$$

com u sendo cada um dos elementos da matriz obtida na Equação (1.10). 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:

$$\begin{aligned} y &= f(u) \\ &= f\left(\begin{bmatrix} -0,7 & 0,2 & 0,2 & 1,1 \end{bmatrix}\right) \\ &= \begin{bmatrix} f(-0,7) & f(0,2) & f(0,2) & f(1,1) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} \end{aligned} \tag{1.12}$$

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 (1.6) para calcular os novos pesos sinápticos, tem-se:

$$P^{(novo)} = \begin{bmatrix} 0,9 & 0,9 & 0,7 \end{bmatrix} + 0,1 \cdot \begin{bmatrix} 0 & -1 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}. \tag{1.13}$$

Realizando as operações apresentadas na Equação (1.13), obtém-se:

$$P^{(novo)} = \begin{bmatrix} 0,8 & 0,8 & 0,9 \end{bmatrix}, \tag{1.14}$$

e, com esta nova matriz de pesos, atualizam-se as Equações (1.10) e (1.12), por:

$$u^{(novo)} = \begin{bmatrix} 0,8 & 0,8 & 0,9 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -0,9 & -0,1 & -0,1 & 0,7 \end{bmatrix} \tag{1.15}$$

e

$$\begin{aligned} y &= f(u) \\ &= f\left(\begin{bmatrix} -0,9 & -0,1 & -0,1 & 0,7 \end{bmatrix}\right) \\ &= \begin{bmatrix} f(-0,9) & f(-0,1) & f(-0,1) & f(0,7) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{1.16}$$

A matriz das saídas calculadas y, obtida na Equação (1.16), 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 10.

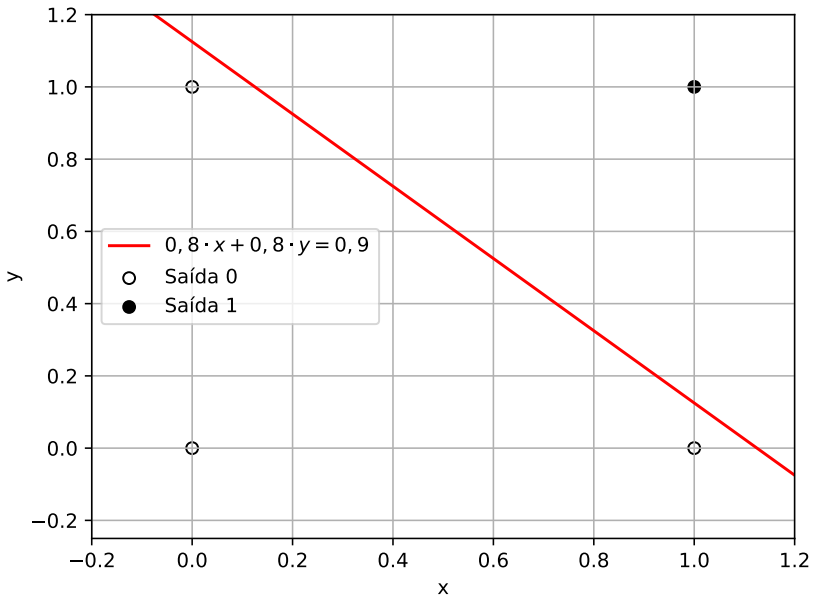


Figura 10 – Gráfico da função booleana "E" com pesos treinados. Elaborado pelo autor.

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.

1.5 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 1.4.1, 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 1.4.1, 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 **??** e utilizados no exemplo referente a função booleana "E" na Tabela **??** são diferentes daqueles que foram apresentados na Equação (1.14) 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.

1.6 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 (1.9) 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 (1.17) que possui n incógnitas e m equações:

$$\begin{cases} P_{1,1}x_{1,1} + P_{1,2}x_{1,2} + \cdots + P_{1,n}x_{1,n} = b_1, \\ P_{1,1}x_{2,1} + P_{1,2}x_{2,2} + \cdots + P_{1,n}x_{2,n} = b_2, \\ \vdots \\ P_{1,1}x_{m,1} + P_{1,2}x_{m,2} + \cdots + P_{1,n}x_{m,n} = b_m \end{cases} . \tag{1.17}$$

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 (1.18), cujo conjunto solução é $S = \{(12, 8)\}$.

$$\begin{cases} x + y = 20 \\ x - y = 4 \end{cases} . \tag{1.18}$$

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 (1.19), que tem seu conjunto solução representado por: $S = \{(x, 20 - x)$, para qualquer $x \in \mathbb{R}\}$.

$$\begin{cases} x + y = 20 \end{cases} . \tag{1.19}$$

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 (1.20). Estes sistemas também são chamados de inconsistentes.

$$\begin{cases} 2x + 3y = 5 \\ 4x + 6y = 12 \end{cases} . \tag{1.20}$$

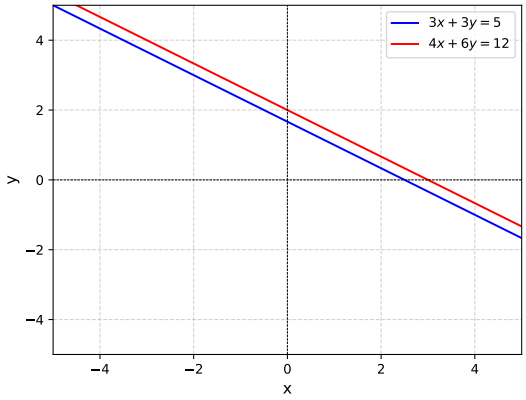
No sistema da Equação (1.20), a segunda linha pode ser dividida por 2 e transformada na primeira, de modo a obter a Equação (1.21), resultando em um absurdo, pois, pela igualdade obtida, apresenta-se $5 = 6$.

$$\begin{cases} 2x + 3y = 5 \\ 2x + 3y = 6 \end{cases} . \tag{1.21}$$

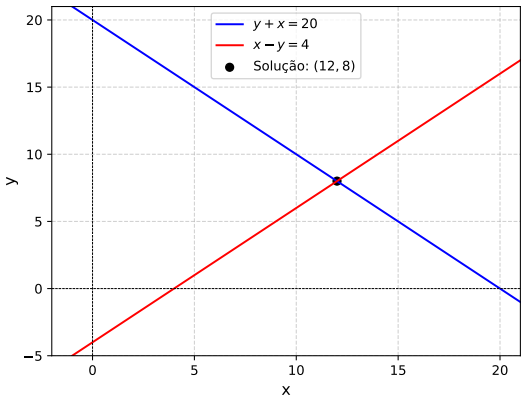
A Figura 11 apresenta as representações geométricas dos sistemas de tipo SI, SPD e SPI, com as Equações (1.20), (1.18) e (1.19) com as Figuras 11a, 11b e 11c, respectivamente.

As intersecções das retas na Figura 11 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.

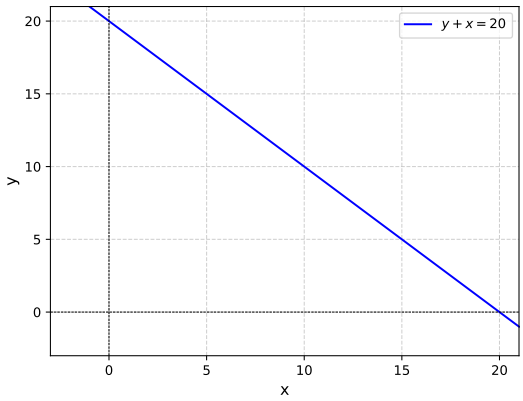
Os sistemas lineares relacionados ao Perceptron são chamados de **Sobredeterminados**, o que significa que há um número maior de equações do que incógnitas.



(a) Sistema Impossível



(b) Sistema Possível e Determinado.



(c) Sistema Possível e Indeterminado.

Figura 11 – 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 (1.22) 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 (1.22) pode ser visto na Figura 12.

$$\begin{cases} x + y = 1 \\ 2x + y = 2 \\ 3x + y = 3 \end{cases} . \tag{1.22}$$

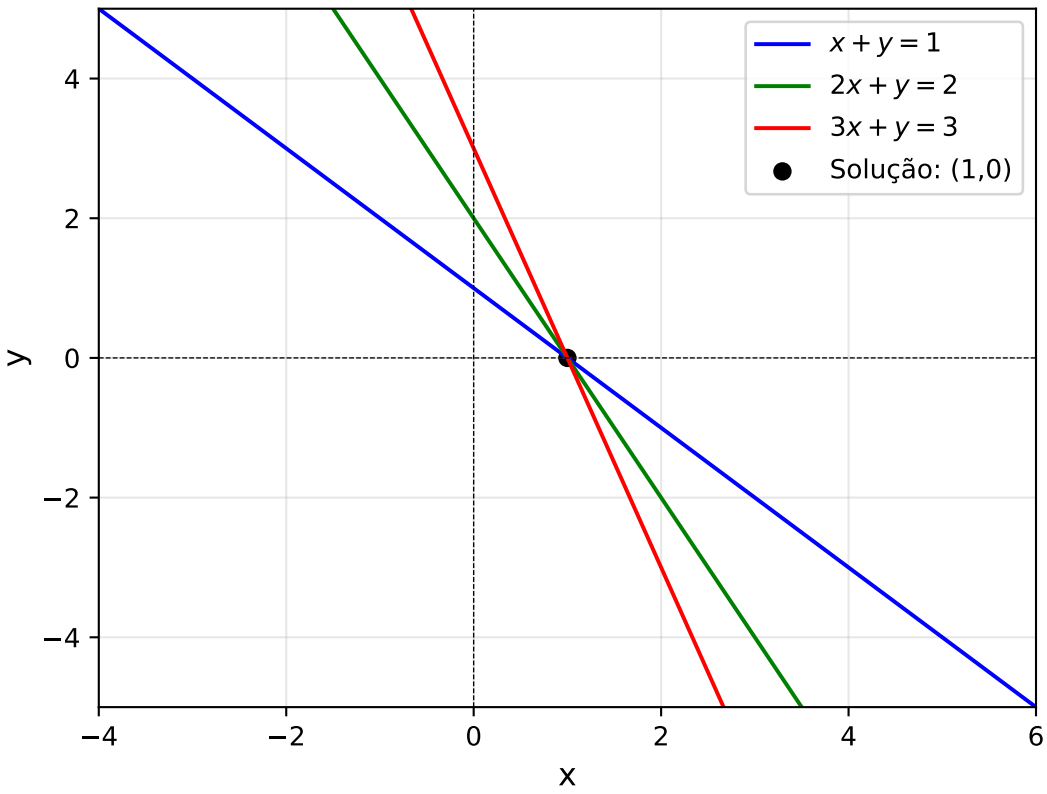


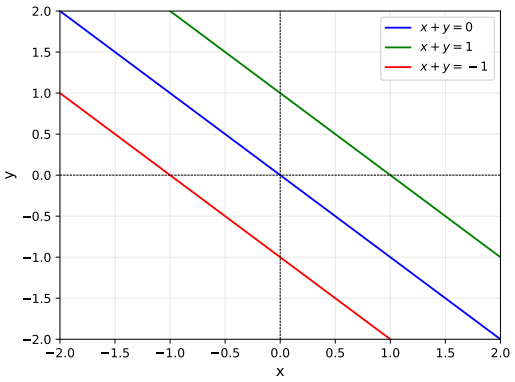
Figura 12 – Sistema Sobredeterminado com solução. Elaborado pelo autor.

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 13.

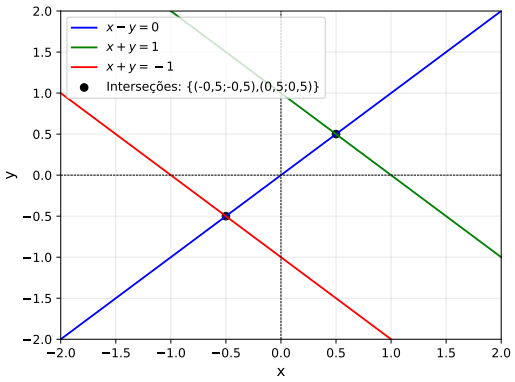
O primeiro é o caso em que todas as equações são inconsistentes e nenhuma delas se satisfazem. Este caso pode ser representado por retas que são todas paralelas, ou seja, não há interseção para nenhuma das equações, como na Figura 13a

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 13b e 13c, mas não está limitado a estas possibilidades. Um exemplo deste caso pode ser visto pela Equação (1.23), que foi representado graficamente na Figura 13c.

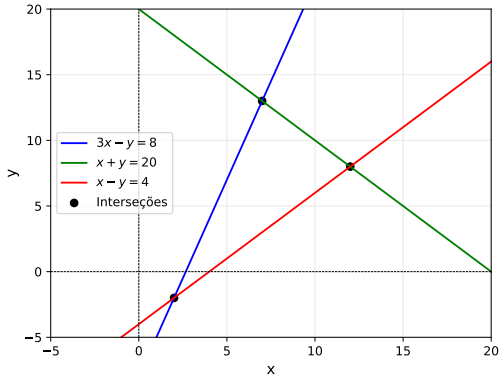
$$\begin{cases} 3x - y = 8 \\ x + y = 20 \\ x - y = 4 \end{cases} \tag{1.23}$$



(a) Todas as retas paralelas.



(b) Duas retas paralelas e duas interseções.



(c) Três interseções.

Figura 13 – Casos de sistemas Sobredeterminados. Elaborados pelo autor.

As equações apresentadas na Equação (1.23), 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 (1.17) poderia ser transformado,

a depender do exemplo, em um sistema linear como na Equação (1.24):

$$\begin{cases} P_{1,1} \cdot x_{1,1} + P_{1,2} \cdot x_{2,1} + \cdots + P_{1,n} \cdot x_{n,1} - \theta > 0 \\ P_{1,1} \cdot x_{1,2} + P_{1,2} \cdot x_{2,2} + \cdots + P_{1,n} \cdot x_{n,2} - \theta < 0 \\ \vdots \\ P_{1,1} \cdot x_{1,n} + P_{1,2} \cdot x_{2,n} + \cdots + P_{1,n} \cdot x_{n,n} - \theta > 0 \end{cases} . \tag{1.24}$$

O sistema de inequações lineares apresentado na Equação (1.24) 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 (1.25) (Géron, 2019).

$$P = \left[\left(x^T \cdot x \right)^{-1} \cdot x^T \right] \cdot y, \tag{1.25}$$

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 **??**, que realiza uma regressão linear eficiente quando comparado ao Perceptron e ao MMQ.

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.

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.

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.

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.

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

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

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.

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

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.