Representações de um Sistema de Equações Lineares

Prof. Dr. Guilherme de Alencar Barreto

Outubro/2015

Departmento de Engenharia de Teleinformática
Universidade Federal do Ceará (UFC), Fortaleza-CE

gbarreto@ufc.br

1 Objetivo

O objetivo maior destanota de aula é reforçar o entendimento de diversas representações matemáticas de um sistema de equações lineares, dada a importância deste tópico não só para a teoria de redes neurais artificiais de modo mais específico; como também para a teoria de reconhecimento de padrões, de um modo mais geral.

As primeiras arquiteturas de redes neurais artificiais que iremos estudar são modelos lineares e o entendimento do funcionamento destes é extremamente facilitado pelas representações matemáticas de sistemas de equações lineares que iremos apresentar ao longo deste documento. Além disso, arquiteturas neurais mais complexas, ou seja, aquelas de natureza não-linear, também usam tais representações, de modo que o entendimento do conteúdo deste capítulo é fundamental para grande parte das arquiteturas de redes neurais artificiais que iremos estudar em nossa disciplina.

Abordaremos a seguir a representação clássica de um sistema de equações lineares, chamada aqui de representação escalar. Em seguida, partiremos para uma representação vetor-matriz, que é bem mais compacta, facilitando a escrita de um sistema com muitas variáveis. Também apresentaremos uma representação gráfica de um sistema de equações lineares, que será interpretada mais adiante como uma arquitetura de rede neural linear de uma única camada com n entradas e m neurônios de saída. Por fim, discutiremos as intepretações geométricas decorrentes das diversas representações apresentadas.

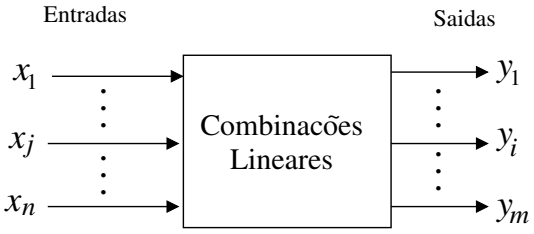


Figura 1: Representação do sistema de equações lineares mostrado em (1) como um diagrama de bloco.

2 Representação Clássica

Considere o seguinte sistema de equações lineares com m variáveis de saída, $y_i \in \mathbb{R}$, $i = 1, 2, \ldots, m$, e com n variáveis de entrada, $x_j \in \mathbb{R}$, $j = 1, 2, \ldots, n$:

$$\begin{aligned}
y_1 &= a_{10} + a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = a_{10} + \sum_{j=1}^{n} a_{1j}x_j \\
y_2 &= a_{20} + a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = a_{20} + \sum_{j=1}^{n} a_{2j}x_j \\
&\vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \\
y_m &= a_{m0} + a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = a_{m0} + \sum_{j=1}^{n} a_{mj}x_j
\end{aligned} \tag{1}$$

em que os coeficientes $a_{ij} \in \mathbb{R}$, são chamados de coeficientes (ou parâmetros) do sistema e, para os nossos interesses, devem ser calculados a partir dos dados disponíveis usando algum método de estimação de parâmetros.

No jargão da área de redes neurais, é o processo de determinação destes coeficientes que chamamos de *aprendizado*, enquanto o método de estimação dos parâmetros em si é chamado de *regra* ou *algoritmo de aprendizado*.

Podemos representar o sistema de equações lineares mostrado em (1) por meio do diagrama de bloco mostrado na Figura 1, em que à esquerda do bloco indicamos as n variáveis de entrada e à direita, as m variáveis de saída.

O que deve ser entendido na Figura é que as n variáveis de entrada são combinadas linearmente, ou seja, são multiplicadas por constantes e depois somadas, para formar as m variáveis de saída.

A título de exemplo, vamos considerar um sistema de equações lineares com $n = 1$ (apenas uma variável de entrada) e $m = 2$ (duas variáveis de saída). Este sistema é escrito como

$$\begin{aligned}
y_1 &= a_{10} + a_{11}x_1 \\
y_2 &= a_{20} + a_{21}x_1
\end{aligned}$$

Note que a expressão $y_1 = a_{10} + a_{11}x_1$ é equivalente à equação de uma reta no plano cartesiano $X_1 \times Y_1$, em que a constante a_{10} define o intercepto (*bias*) e a constante a_{11} é chamada de coeficiente angular ou inclinação (*slope*) da reta. Raciocínio similar se aplica à equação $y_2 = a_{20} + a_{21}x_1$.

Vamos considerar agora um sistema de equações lineares com $n = m = 2$ (duas variáveis de entrada e duas de saída). Este sistema é escrito como

$$\begin{aligned} y_1 &= a_{10} + a_{11}x_1 + a_{12}x_2 \\ y_2 &= a_{20} + a_{21}x_1 + a_{22}x_2 \end{aligned}$$

Desta vez, podemos interpretar a expressão para y_1 como a equação de um plano no $\mathbb{R}^3$, representando pontos no espaço cartesiano $X_1 \times X_2 \times Y_1$. O mesmo raciocínio se aplica à equação para y_2.

De um modo mais geral, a i-ésima linha do sistema de equações lineares em (1), representada como

$$y_i = a_{i0} + a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n, \tag{2}$$

pode ser interpretada como a equação de um *hiperplano*[1] no $\mathbb{R}^{n+1}$.

3 Representação Vetor-Matriz

Uma forma muito útil de escrever o sistema de equações mostrado em (1) faz uso de uma notação vetor-matriz. Para este fim, precisamos antes fazer as seguintes definições:

1. Vetor de entrada $\mathbf{x} \in \mathbb{R}^{n+1}$, contendo as n variáveis de entrada.

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{(n+1)\times 1} \tag{3}$$

2. Vetor de saída $\mathbf{y} \in \mathbb{R}^m$, contendo as m variáveis de saída.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_{m\times 1} \tag{4}$$

3. Matriz de transformação $\mathbf{A} \in \mathbb{R}^{m\times n}$, que "opera" sobre o vetor de entrada $\mathbf{x}$ para gerar o vetor $\mathbf{y}$.

$$\mathbf{A} = \begin{bmatrix} a_{10} & a_{11} & a_{12} & \cdots & a_{1n} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m0} & a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}_{m\times n} \tag{5}$$

Lembre-se que operações ou transformações lineares sobre um vetor podem alterar a sua norma (i.e. o comprimento) e/ou sua orientação (i.e. direção). Obviamente, se o sistema for quadrado (i.e. $n = m$) e a matriz $\mathbf{A}$ for a matriz identidade de ordem adequada, então nem

[1] O prefixo *hiper-* é usado para designar generalizações de figuras geométricas no $\mathbb{R}^3$ que ocorreriam em espaços de dimensão maior que 3, ou seja, $\mathbb{R}^4$, $\mathbb{R}^5$ e assim por diante.

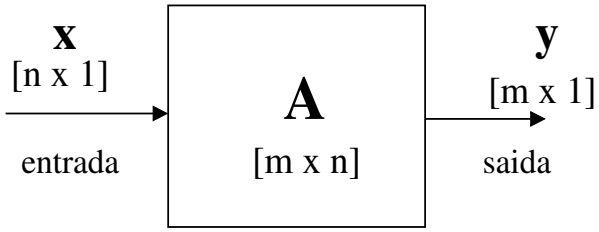


Figura 2: Representação do sistema linear $\mathbf{y} = \mathbf{A}\mathbf{x}$ em um diagrama de bloco.

a norma do vetor, nem sua orientação serão alteradas, pois a matriz identidade é o elemento neutro da multiplicação de matrizes.

Assim, o sistema de equações mostrado em (1) pode escrito como

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{6}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{10} & a_{11} & a_{12} & \cdots & a_{1n} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m0} & a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \tag{7}$$

Com isto, o sistema $\mathbf{y} = \mathbf{A}\mathbf{x}$ passa a ser representada em diagrama de bloco conforme mostrado na Figura 2.

Prosseguindo com a notação vetor-matriz $\mathbf{y} = \mathbf{A}\mathbf{x}$, é possível perceber que cada linha da matriz $\mathbf{A}$ é formada por um conjunto de coeficientes que multiplicam cada componente do vetor $\mathbf{x}$. Assim, podemos escrever a i-ésima saída y_i, $i = 1, \ldots, m$, como

$$\begin{aligned} y_i &= [a_{i0} \;\; a_{i1} \;\; a_{i2} \;\; \cdots \;\; a_{in}]^T \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \\ &= a_{i0} + a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n, \\ &= \mathbf{a}_i^T \mathbf{x}, \end{aligned} \tag{8}$$

em que o vetor de coeficientes $\mathbf{a}_i \in \mathbb{R}^{n+1}$ tem a mesma dimensão do vetor de entrada $\mathbf{x}$. O sobrescrito T denota o vetor transposto.

Importante!! - Perceba que a notação $y_i = \mathbf{a}_i^T \mathbf{x}$ permite entender a i-ésima linha do vetor $\mathbf{y}$ como o resultado do *produto escalar* entre o vetor de coeficientes $\mathbf{a}_i$ e o vetor de entrada $\mathbf{x}$.

4 Representação Arquitetural

Vimos nas seções anteriores que a i-ésima linha do sistema de equações em (1) pode ser escrita matematicamente de três maneiras diferentes, porém equivalentes.

1. **Forma expandida da equação do hiperplano**:

$$y_i = a_{i0} + a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \tag{9}$$

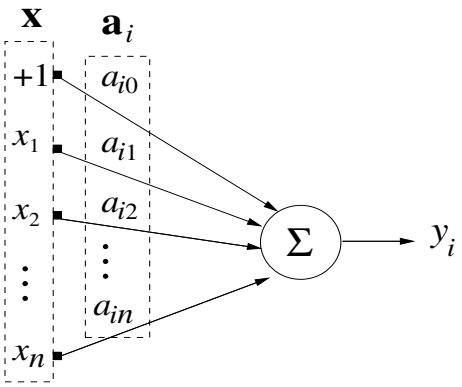


Figura 3: Representação arquitetural da Eq. (2) conhecida como *combinador linear*.

2. **Forma compacta da equação do hiperplano**:

$$y_i = a_{i0} + \sum_{j=1}^{n} a_{ij} x_j \tag{10}$$

3. **Produto-escalar entre $\mathbf{a}_i$ e $\mathbf{x}$**:

$$y_i = \mathbf{a}_i^T \mathbf{x} \tag{11}$$

Para finalizar esta nota de aula, vamos mostrar a chamada *representação arquitetural* da i-ésima linha do sistema de equações mostrado em (1). Esta representação, chamada genericamente de *combinador linear*, está mostrada na Figura 3

Em outras notas de aula, veremos que o combinador linear será usado como base teórica para um dos modelos mais usados até hoje de um neurônio artificial: o neurônio de McCulloch-Pitts [1].

A seguir vamos usar os conhecimentos de Álgebra Linear discutidos até agora nesta nota de aula para projetar um sistema computacional capaz de aprender as funções lógicas AND e OR.

5 Portas AND/OR e o Neurônio de McCulloch-Pitts

Em geral, nos cursos de Engenharia[2], o primeiro contato do estudante com os princípios da lógica booleana dá-se na disciplina de eletrônica digital ou equivalentes. Neste sentido, é de amplo conhecimento deste estudante as tabelas-verdades das funções lógicas AND e OR.

Isto posto, cabe aqui a seguinte pergunta:

> Qual a relação entre portas lógicas e Inteligência Computacional?

Talvez uma olhada no título do livro de George Boole[3] (vide Figura 4) mostrado a seguir possa ajudar a responder tal questão.

> George Boole (1854). *An investigation into the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities*. Cambridge: MacMillan and Co.

[2]Refiro-me especificamente às Engenharias Elétrica, Eletrônica, de Telecomunicações, de Computação, Automação e Controle, Mecatrônica e Biomédica

[3]`https://archive.org/details/investigationofl00boolrich`

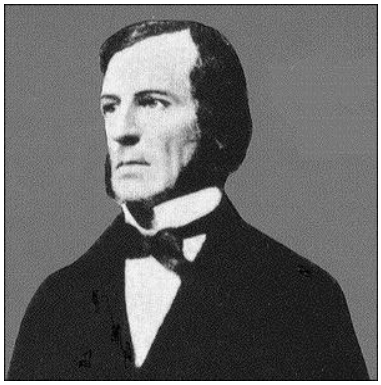


Figura 4: George Boole (02/11/1815 - 08/12/1864). Matemático e filósofo britânico. É o criador da Álgebra Booleana, base da atual aritmética computacional.

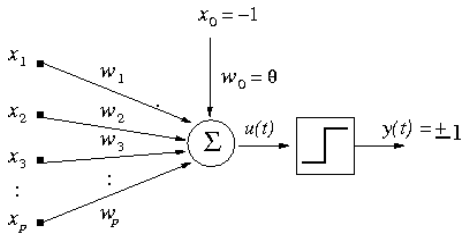


Figura 5: Representação esquemática moderna do modelo do neurônio de McCulloch-Pitts.

Note que Boole estava interessado em estabelecer uma lógica formal para as leis do pensamento. Em outras palavras, estava usando a lógica como linguagem para descrever (i.e. modelar) o processo de raciocínio humano. Porém, ele não estava interessado em mostrar como tal lógica poderia ser efetivamente implementada no cérebro humano, até porque naquela época ainda não se tinha conhecimento de que a unidade individual do sistema nervoso é o neurônio.

Tal conhecimento só ficou claro nos trabalhos do médico e histologista espanhol, Santiago Ramón y Cajal, ganhador do prêmio Nobel de Medicina em 1906 juntamente com o também médico e histologista italiano Camillo Golgi.

Somente em 1943, com o trabalho de Warren McCulloch (neurocientista) e Walter Pitts (logicista), é que surgiu uma primeira hipótese de como tais portas lógicas poderiam ser implementadas no cérebro, dando origem ao modelo matemático conhecido hoje como *neurônio artificial de Mc-Culloch-Pitts* [1].

O modelo do neurônio de McCulloch-Pitts é um combinador linear com uma função limitadora (e.g. função sinal) na sua saída, tal como ilustrado na Figura 5. Compare esta figura com a Figura 3.

Bom, agora ainda permanece em aberto a seguinte questão:

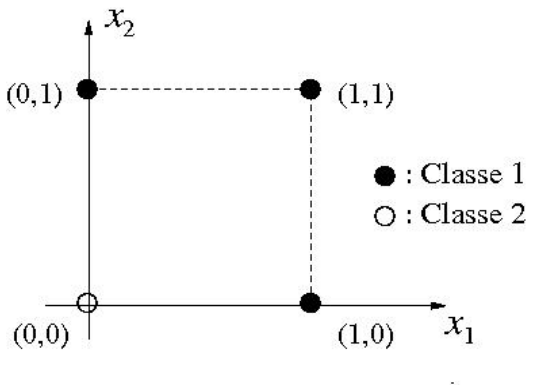
> É possível construir/implementar/projetar uma porta lógica usando o modelo do neurônio de McCulloch-Pitts (M-P)?

A resposta é sim! Na análise que se segue, vamos considerar apenas a parte linear do neurônio M-P. Usaremos a função sinal apenas para limitar/quantizar a saída para os valores ± 1. Considere agora a tabela verdade da função lógica OR para duas variáveis de entrada x_1 e x_2, mostrada na Figura 6a, e sua representação no plano cartesiano $X_1 \times X_2$, mostrada na Figura 6b.

Porta OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

(a)



(b)

Figura 6: Duas representações diferentes da função lógica OR. (a) Tabela-verdade. (b) Plano cartesiano.

Note que se representarmos no plano cartesiano as coordenadas (x_1, x_2) da tabela-verdade da função lógica OR que produzem saída "1" como um pequeno círculo cheio e as coordenadas que produzem "0" com um pequeno círculo vazio, obteremos a Figura 6b. Esta figura nos permite formular o problema de implementar uma porta lógica como um problema de classificação de padrões!

Na classe 1 devem ficar os pontos de coordenadas $\{(1,0);(0,1);(1,1)\}$, enquanto na classe 2 deve ficar apenas o ponto de coordenada (0,0). É possível notar também que a representação no plano cartesiano nos faz perceber que uma simples reta pode ser usada para separar as duas classes. Deste modo, podemos dizer que o problema de classificação que conduz à implementação da função lógica OR é um *problema linearmente separável*.

A reta em questão será implementada pelo neurônio de McCulloch-Pits por meio da seguinte expressão:

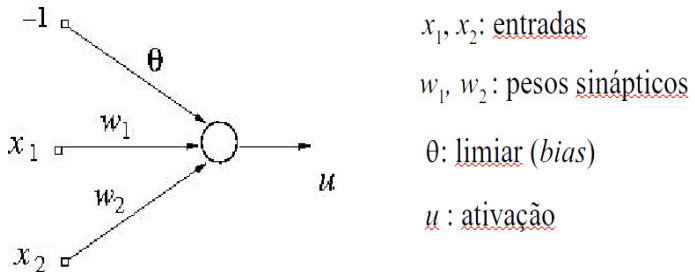
$$\begin{aligned} u &= w_0 x_0 + w_1 x_1 + w_2 x_2, \\ &= -\theta + w_1 x_1 + w_2 x_2, \end{aligned} \tag{12}$$

para a qual assumimos $x_0 = -1$ e $w_0 = \theta$. A representação arquitetural desta reta como um neurônio está representada na Figura 7a e graficamente no plano cartesiano na Figura 7b.

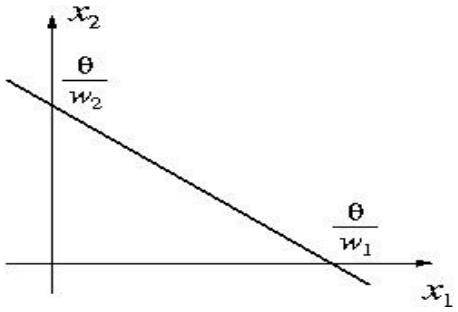
Das aulas de Geometria Analítica lembraremos que uma reta divide o plano cartesiano em duas regiões chamadas semiplanos. Em um dos semiplanos, teremos os pontos que satisfazem $u > 0$, ou seja, são os pontos localizados acima da reta. No outro semiplano, teremos os pontos que satisfazem $u < 0$ (pontos abaixo da reta). Para os pontos na reta teremos $u = 0$.

Nosso objetivo ao desenvolver um modelo matemático que funcione como a função lógica OR consiste em posicionar uma reta que coloque os pontos $\{(1,0);(0,1);(1,1)\}$ no semiplano em que $u > 0$, enquanto o ponto $(0,0)$ fique o semiplano em que $u < 0$. Para este fim, não usaremos a representação binária da função lógica OR mostrada na Figura 6, mas sim a representação *bipolar* mostrada na Figura 8. A única diferença entre as duas representações é o fato de usarmos -1 em vez de 0 (zero) na tabela-verdade correspondente.

Assim, a partir da representação bipolar da função lógica OR, podemos usar a Eq. (12) para,



(a)



(b)

Figura 7: Representação do neurônio MP para implementar a função lógica OR com 2 entradas. (a) Representação arquitetural. (b) Reta no plano cartesiano $X_1 \times X_2$.

linha a linha da tabela-verdade, construir o seguinte sistema de equações lineares:

$$\begin{aligned} -\theta + w_1(-1) + w_2(-1) &= -1 \\ -\theta + w_1(+1) + w_2(-1) &= +1 \\ -\theta + w_1(-1) + w_2(+1) &= +1 \\ -\theta + w_1(+1) + w_2(+1) &= +1 \end{aligned} \tag{13}$$

em que as incógnitas são os parâmetros do modelo do neurônio MP: o limiar (θ) e os pesos sinápticos w_1 e w_2. Vamos reescrever o sistema acima na forma matricial $\mathbf{Aw} = \mathbf{b}$, ou seja

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & +1 \end{bmatrix} \begin{bmatrix} \theta \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}, \tag{14}$$

em que

$$\mathbf{A} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & +1 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \end{bmatrix} \quad \text{e} \quad \mathbf{b} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}. \tag{15}$$

Note que o vetor $\mathbf{w}$ contém os parâmetros ajustáveis do neurônio MP. A matriz de coeficientes $\mathbf{A}$ e o vetor de saídas $\mathbf{b}$ são construídos a partir da tabela-verdade da função lógica OR. Por fim, a primeira coluna da matriz $\mathbf{A}$, formada somente de -1's surge por termos usado a entrada fixa $x_0 = -1$.

Portanto, se conseguirmos resolver o sistema linear acima, ou seja, obter uma solução numérica para $\mathbf{w}$, teremos nosso neurônio MP que implementa a função lógica OR. Surge então a seguinte questão:

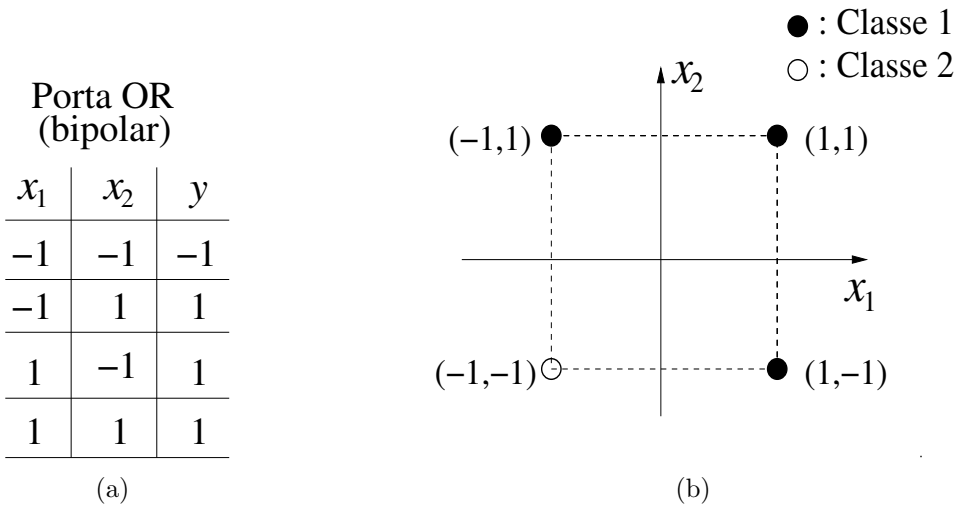


Figura 8: Representação bipolar da função lógica OR. (a) Tabela-verdade. (b) Plano cartesiano.

> Como saber se o sistema em questão admite solução e, em caso afirmativo, saber se ela é única?

Motivados por esta questão, vamos revisar rapidamente a seguir alguns conceitos importantes de Álgebra Linear.

5.1 Dicas de Álgebra Linear

Sempre que um sistema de equações do tipo $\mathbf{A}_{m\times(n+1)}\mathbf{w}_{(n+1)\times 1} = \mathbf{b}_{m\times 1}$ aparecer em um problema de modelagem de um fenômeno, é interessante que analizemos algumas propriedades da matriz $\mathbf{A}$ a fim de verificar se o sistema tem solução. Das aulas de Álgebra Linear sabemos que a existência de solução do sistema está associada ao valor do *posto* da matriz $\mathbf{A}$.

Para quem não lembra, o posto fornece uma medida do grau de *independência linear* das linhas (ou colunas) da matriz. Grosso modo, o posto nos dá o número de linhas (ou colunas) linearmente independentes da matriz, ou seja, quantas linhas (ou colunas) existem que não podem ser escritas como uma combinação linear das outras linhas (ou colunas). Em qualquer matriz o número de linhas linearmente independentes coincide com o número de colunas linearmente independentes.

Formalmente, o posto da matriz $\mathbf{A}_{m\times(n+1)}$ é definido como

$$\rho(\mathbf{A}) = posto(\mathbf{A}) \leq \min(m, n+1), \tag{16}$$

ou seja, o maior valor que o posto de uma matriz pode assumir é igual à menor das dimensões desta matriz. Por exemplo, se uma matriz tem dimensão 4×7, o valor máximo que pode alcançar o posto desta matriz é 4, visto que este é o mínimo dentre 4 e 7.

Uma forma prática de se calcular o posto de uma matriz é colocá-la na forma escalonada reduzida por meio de operações elementares sobre as linhas desta matriz. O posto da matriz será igual ao número de linhas não-nulas da forma escalonada reduzida.

Sabendo como calcular o posto de uma matriz, o nosso interesse maior está em como determinar se um sistema linear do tipo $\mathbf{Aw} = \mathbf{b}$ tem ou não solução. E se existir solução, se ela é única ou se possui infinitas soluções. Existe um resultado clássico em Álgebra Linear, na forma de um teorema, que nos fornece esta resposta. Não iremos demonstrar o teorema, pois nos interessa aqui apenas seu enunciado.

Teorema: Considere o sistema de equações lineares $\mathbf{A}_{m\times(n+1)}\mathbf{w}_{(n+1)\times 1} = \mathbf{b}_{m\times 1}$.

Existência de solução - Tal sistema admite solução se, e somente se, o posto da matriz ampliada $\tilde{\mathbf{A}} = [\mathbf{A} \mid \mathbf{b}]$ é igual ao posto da matriz dos coeficientes $\mathbf{A}$.

Unicidade da solução - Se as duas matrizes têm o mesmo posto p e $p = n + 1$, a solução será única.

Infinidade de soluções - Se as duas matrizes têm o mesmo posto p e $p < n + 1$, podemos escolher $n - p + 1$ incógnitas e as outras p incógnitas serão dadas em função destas. Neste caso, dizemos que o grau de liberdade do sistema é $n - p$, o que significa que o sistema possui $n - p + 1$ variáveis livres.

Inexistência de solução - Por exclusão, se os postos das duas matrizes diferem o sistema não possui solução exata.

Usaremos este teorema para avaliar a existência de solução para o sistema mostrado na Equação (14).

6 Solução pelo Método dos Mínimos Quadrados

Para saber se o sistema tem solução ou não, vamos determinar o posto das matrizes $\mathbf{A}$ e $\tilde{\mathbf{A}} = [\mathbf{A} \mid \mathbf{b}]$. para o problema da implementação da função lógica OR, estas são dadas respectivamente por

$$\mathbf{A} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & +1 \end{bmatrix} \quad \text{e} \quad \tilde{\mathbf{A}} = \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & +1 \\ -1 & +1 & +1 & +1 \end{bmatrix}. \tag{17}$$

Usando o Matlab ou o Octave, podemos facilmente determinar o posto dessas duas matrizes usando o comando `rank`. Os resultados obtidos são os seguintes:

```
>> A=[-1 -1 -1;-1 1 -1;-1 -1 1;-1 1 1]; % matriz A
>> b=[-1;1;1;1];
>> AA=[A b]; % matriz ampliada
>> pa=rank(A) % posto da matriz A
pa =
       3
>> paa=rank(AA) % posto da matriz ampliada
paa =
       4
```

Como `pa` $\neq$ `paa`, podemos concluir, sem resolver o sistema, que ele não tem solução exata. Isto significa dizer que não há um vetor $\mathbf{w}$ que ao ser multiplicado pela matriz $\mathbf{A}$ produza *exatamente* o vetor $\mathbf{b}$. Em outras palavras, quando não o sistema não tem solução, sempre teremos $\mathbf{Aw} \neq \mathbf{b}$, $\forall \mathbf{w}$.

Matematicamente, isto é equivalente à seguinte expressão:

$$\mathbf{b} - \mathbf{Aw} \neq \mathbf{0}, \tag{18}$$

que pode ser transformada em uma igualdade e, então, ser escrita como

$$\mathbf{b} - \mathbf{Aw} = \mathbf{e}, \tag{19}$$

com o vetor $\mathbf{e} \in \mathbb{R}^m$ tendo pelo menos uma componente não-nula.

O vetor $\mathbf{e}$ pode ser entendido como o vetor de erros, ou seja, o vetor cujas componentes são os valores que faltaram para o sistema ter solução. Se todas as componentes do vetor $\mathbf{e}$ fossem nulas, não haveria erros e o sistema teria uma solução exata!

Não temos uma solução exata, mas podemos encontrar uma solução aproximada que produza o menor erro possível. Neste caso, demos expressar esta intenção através de uma função-custo ou função-objetivo que dependa do erro. Esta função é a norma quadrática do erro, dada por

$$J(\mathbf{w}) = \|\mathbf{e}\|^2 = \mathbf{e}^T\mathbf{e} = \sum_{i=1}^{n} e_i^2 = (\mathbf{b} - \mathbf{A}\mathbf{w})^T(\mathbf{b} - \mathbf{A}\mathbf{w}), \tag{20}$$

em que e_i denota a i-ésima componente do vetor $\mathbf{e}$ e $\|\cdot\|$ simboliza a norma euclidiana do argumento.

A função-custo $J(\mathbf{w})$ pode ser entendida como uma função que busca encontrar o vetor de parâmetros $\hat{\mathbf{w}}$ do neurônio MP que produz o vetor $\mathbf{e}$ de menor norma quadrática.

A Eq. (20) pode ser decomposta em

$$\begin{aligned} J(\mathbf{w}) &= \mathbf{b}^T\mathbf{b} - \mathbf{b}^T\mathbf{A}\mathbf{w} - \mathbf{w}^T\mathbf{A}^T\mathbf{b} + \mathbf{w}^T\mathbf{A}^T\mathbf{A}\mathbf{w} \\ &= \mathbf{b}^T\mathbf{b} - 2\mathbf{w}^T\mathbf{A}^T\mathbf{b} + \mathbf{w}^T\mathbf{A}^T\mathbf{A}\mathbf{w} \end{aligned} \tag{21}$$

uma vez que $\mathbf{w}^T\mathbf{A}^T\mathbf{b} = \mathbf{b}^T\mathbf{A}\mathbf{w}$ resulta no mesmo escalar.

Daí, para encontrar a estimativa de mínimos quadrados para o vetor de pesos $\mathbf{w}$ temos que determinar o vetor gradiente de $J(\mathbf{w})$ e igualá-lo ao vetor nulo de dimensão adequada, ou seja,

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{A}^T\mathbf{b} + 2\mathbf{A}^T\mathbf{A}\mathbf{w} = \mathbf{0}, \tag{22}$$

em que $\mathbf{0}$ é um vetor de zeros de dimensão $m \times 1$. Rearranjando os termos da Eq. (22) resulta em

$$\mathbf{A}^T\mathbf{A}\mathbf{w} = \mathbf{A}^T\mathbf{b}. \tag{23}$$

Note que a matriz $\mathbf{A}^T\mathbf{A}$ é quadrada, de dimensão $(n+1) \times (n+1)$). Portanto, para isolar o vetor $\mathbf{w}$ na Eq. (23) basta multiplicar ambos os lados da Eq. (23) pela inversa de $\mathbf{A}^T\mathbf{A}$. Desta forma, a estimativa de mínimos quadrados de $\mathbf{w}$ é dada por

$$\mathbf{w} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}. \tag{24}$$

Portanto, a ativação do neurônio de McCulloch-Pitts, para um dado vetor de entrada $\mathbf{x}$, é calculada por meio da seguinte expressão:

$$\mathbf{u} = \mathbf{A}\mathbf{w}, \tag{25}$$

de modo que a saída quantizada pela função `sign`[4], é dada por

$$\mathbf{y} = \text{sign}(\mathbf{u}), \tag{26}$$

em que a função sinal é aplicada individualmente a cada componente do vetor $\mathbf{u}$.

De posse da matriz $\mathbf{A}$ e do vetor $\mathbf{b}$, podemos estimar os parâmetros do neurônio MP facilmente usando os seguintes comandos no Matlab/Octave.

[4]Em português, chamada de função sinal, sendo definida como $y = 1$, se $u > 0$; $y = -1$, se $u \leq 0$.

```
>> w=inv(A'*A)*A'*b
w =
       -0.50000
        0.50000
        0.50000
```

Note que se multiplicarmos $\mathbf{A}$ e $\mathbf{w}$ não resultará em $\mathbf{b}$:

$$\mathbf{Aw} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & +1 \end{bmatrix} \begin{bmatrix} -0.5 \\ +0.5 \\ +0.5 \end{bmatrix} = \begin{bmatrix} -0.5 \\ +0.5 \\ +0.5 \\ +1.5 \end{bmatrix} \neq \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix} \tag{27}$$

Note, contudo, que o produto $\mathbf{Aw}$ refere-se à parte linear do neurônio MP. Se aplicarmos a função limitadora (i.e. função sinal) às componentes do vetor $\mathbf{u}$ obteremos o vetor de saídas correto, ou seja

$$\text{sign}(\mathbf{Aw}) = \text{sign}\left(\begin{bmatrix} -0.5 \\ +0.5 \\ +0.5 \\ +1.5 \end{bmatrix} \right) = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}. \tag{28}$$

6.1 Comandos Alternativos no Matlab/Octave

A solução de mínimos quadrados no exemplo da implementação da função lógica OR pelo neurônio MP foi obtida através da implementação direta da Eq. (24), que chamaremos aqui de equação de *livro-texto* do método dos mínimos quadrados. Em função do seu elevado custo computacional, principlamente para vetores/matrizes de elevada dimensão, e de uma maior susceptibilidade a erros numéricos, tal implementação não é recomendada.

Para fins mais profissionais, sugere-se que seja dada preferência a um dos seguintes comandos: barra invertida ('\') ou `pinv`. Vamos testá-los abaixo.

```
>> w=A\b
w =
       -0.50000
        0.50000
        0.50000
>> w=pinv(A)*b
w =
       -0.50000
        0.50000
        0.50000
```

Podemos notar que os resultados são os mesmos para os três comandos (incluindo o do livro-texto). Contudo, é importante frisar que há diferenças na forma em que são implementados no Matlab/Octave. O comando da barra invertida ('\') utiliza a decomposição QR, enquando o comando `pinv` usa a técnica de decomposição em valores singulares (SVD, sigla em inglês). Mais detalhes sobre estas técnicas podem ser encontradas em qualquer bom livro de Álgebra Linear. Recomendo o livro do Davyd Lay [2].

Fica a minha recomendação para uso da função `pinv` caso o Matlab/Octave esteja sendo usado ou, de modo mais geral, a técnica baseada em SVD para obtenção da solução de mínimos

quadrados para um sistema linear. Esta técnica pode ser usada mesmo que a matriz $\mathbf{A}$ não tenha posto completo, o que não acontece para o caso do comando baseado na barra invertida (que usa decomposição QR).

Um comentário final: o nome do comando `pinv` se baseia no termo *pseudoinversa*, pois este é o termo pelo qual a matriz $\mathbf{A}^{\dagger} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T$ é conhecida em Álgebra Linear.

Exercício Computacional: Com base na teoria exposta nesta nota de aula, implemente a função lógica AND usando o neurônio de McCulloch-Pitts e o método dos mínimos quadrados.

Referências

[1] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[2] D. C. Lay. *Álgebra Linear e suas Aplicações*. Editora LTC, 2013.