# Introdução à Engenharia de Dados
# Uma Perspectiva de Redes Neurais Artificiais e de Reconhecimento de Padrões

Prof. Antônio de Pádua Braga
Departamento de Engenharia Eletrônica
Escola de Engenharia da UFMG

18 de setembro de 2020

# Sumário

# Capítulo 1

# Conceitos Básicos

## 1.1 Introdução

Vimos nos últimos anos uma grande transformação no nosso dia-a-dia, proporcionada pelo surgimento de novas tecnologias de comunicação de dados e de eletrônica embarcada, incorporadas massivamente a dispositivos de nosso uso diário. O exemplo mais típico destes novos dispositivos é o aparelho de telefone celular, cujas funções atuais vão muito além da simples comunicação de voz. Dotados de sensores dos mais diversos tipos, estes dispositivos possuem uma grande capacidade de coleta e de armazenamento de dados, proporcionando também ao usuário final um portal de conexão a serviços, a outros usuários e dispositivos conectados em rede. A chamada Internet das Coisas (*Internet of Things* - IoT) [XHL14], caracterizada pela conexão massiva em rede de dispositivos com algum grau de autonomia, vem determinando novas tendências e modificando padrões atuais de comportamento. Exemplos destes novos dispositivos, caracterizados conceitualmente como unidades físico-cibernéticas (*Cyber-Physical Systems Units* - CPSU) [KM15, Wol09], podem ser encontrados nas indústrias, nas residências, nos automóveis, ou mesmo na forma de agentes de *software* conectados à Internet. Um diagrama esquemático de uma CPSU é apresentado na Figura 1.1.

Conceitualmente, uma CPSU envolve basicamente uma plataforma de Computação (*Cyber*), elementos físicos com os quais a plataforma de computação interage (*Physical*) e uma conexão entre os dois primeiros elementos. CPSU não é um conceito novo [SGLW08, Wol09], pois envolve a conexão entre programas de computador e elementos físicos, porém, nos últimos anos a unidade cibernética tem sua implementação frequentemente associada aos métodos de aprendizado de máquina [SCAN16, LXZ$^+$17, WTH$^+$17, WSZ$^+$17]. Na representação da Figura 1.1, a máquina de aprendizado, implementada por um programa de computador ou circuito dedicado, faz o papel do elemento cibernético, que tem capacidade de amostrar o estado do mundo externo por meio da leitura de sinais contínuos e discretos. Conforme representação da Figura 1.1, a partir da leitura do estado do sistema, representado pelas entradas $\mathbf{x}$ e $\mathbf{y}$ na figura, a máquina de aprendizado tem capacidade para atuar no mundo externo por meio da saída $u(\hat{\mathbf{y}})$ e alterar o seu estado. A saída $\hat{\mathbf{y}}$ é uma estimativa da resposta do sistema ao vetor de entradas $\mathbf{x}$. Assim, a função aprendida pela unidade

cibernética envolve tipicamente uma estimativa do comportamento da unidade física.

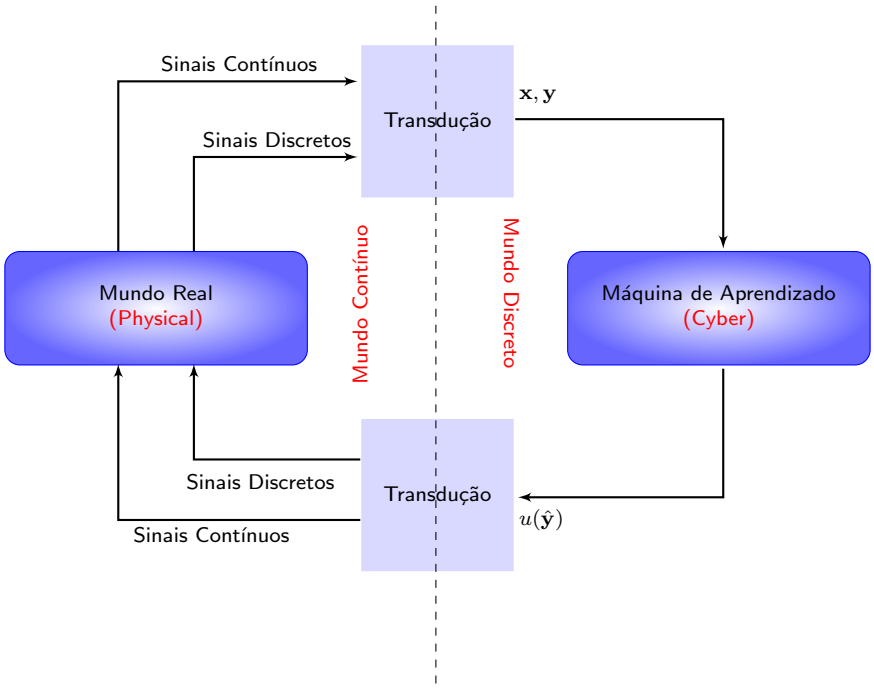



Figura 1.1: Representação esquemática de uma CPSU (Cyber-Physical System Unit), sendo a unidade cibernética implementada por meio de uma Máquina de Aprendizado.

Nos últimos anos, fatores como o baixo custo do *hardware* embarcado, maior escala de integração de circuitos integrados, baixo custo de armazenamento e novos paradigmas como a chamada Indústria 4.0 [Lu17], levaram a expectativas quando à implementação massiva de CPSUs autônomas. Esta nova tendência está intrinsecamente associada à ideia de que veículos, indústrias e residências, por exemplo, se tornarão cada vez mais autônomos e livres da intervenção humana. Assim, o estudo de métodos, modelos e algoritmos de aprendizado que sejam menos dependentes do usuário e de processamento local mais intensivo torna-se extremamente relevante. Idealmente, espera-se que estes novos métodos sejam também passíveis de implementação ao nível de circuito integrado e que demandem uma menor interação com o usuário.

Nos capítulos seguintes será dada ênfase a problemas de aprendizado em que a função principal da unidade cibernética é a de estimar o comportamento da unidade física. Representando o problema de maneira genérica, considere que a função da unidade física, ou do sistema a ser modelado, seja representada por $f_g(\mathbf{x})$. Considerando que a função executada pela unidade cibernética, ou modelo, seja representada de maneira genérica como $f(\mathbf{x}, \mathbf{w})$, onde $\mathbf{w}$ é o vetor de parâmetros aprendidos, espera-se que, por meio de um processo de adaptação de $\mathbf{w}$, $f(\mathbf{x}, \mathbf{w})$ se aproxime gradualmente de $f_g(\mathbf{x})$. Assim, os problemas de aprendizado estudados ao longo deste livro envolverão basicamente tarefas de aproximação de funções, ou seja, tarefas que envolvam aproximar $f_g(\mathbf{x})$ por meio

de uma função $f(\mathbf{x}, \mathbf{w})$ resultante da adaptação de seus parâmetros $\mathbf{w}$.

Dentro deste novo cenário, nas páginas que se seguem serão apresentados conceitos fundamentais da grande área de Ciência de Dados, com ênfase particular em Redes Neurais Artificiais (RNAs) e Reconhecimento de Padrões (RP). Esta forma de organização do texto, que agrega RNAs e RP no mesmo contexto, se justifica, já que as RNAs se apresentam como uma das possíveis e mais populares abordagens para a resolução de problemas de RP. Não obstante, as RNAs se aplicam a uma gama maior de problemas e de áreas do conhecimento, não estando a mesma restrita a problemas de RP. Por serem aproximadores universais de funções [Cyb89] as RNAs podem ser aplicadas também a problemas de regressão e de previsão.

A Figura 1.2 mostra de maneira esquemática um diagrama que representa o relacionamento entre estas três grandes áreas citadas no parágrafo anterior. Os problemas de regressão, previsão e classificação podem ser também tratados por um grande número de abordagens, entre elas as RNAs, representada na região de intersecção entre as três áreas. Por sua vez, há também uma gama variada de métodos para a resolução de problemas de classificação, ou RP, os quais serão também abordados nos capítulos seguintes. Não obstante, qualquer que seja o método adotado, a resolução destes problemas envolverá a construção de uma função $y = f(x)$, $f : A \rightarrow B$, em que A e B são, respectivamente, o domínio e o contra-domínio de $f$, $x \in A$ e $y \in B$. Assim, $f(x)$ expressa a lei que mapeia os elementos de A em B, podendo a mesma representar de maneira geral qualquer um dos três problemas acima. Portanto, os problemas abordados ao longo deste livro envolverão a construção de funções visando à resolução de problemas nas três grandes áreas representadas de maneira esquemática na Figura 1.2. A escolha da função que representará a solução do problema será realizada por meio de um conjunto de amostras $x \in A$ e de seus valores correspondentes $y \in B$. Serão apresentados, nas subseções seguintes, exemplos de aplicações nestras três grandes áreas.

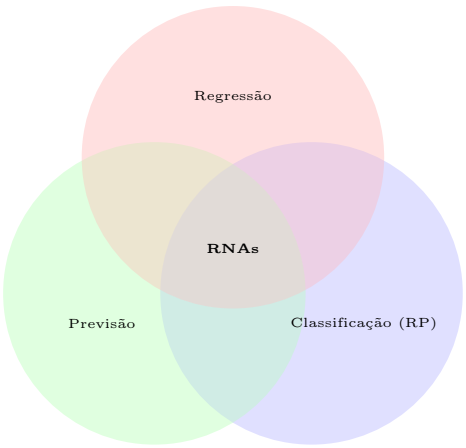



Figura 1.2: Principais áreas de aplicação das RNAs, entre elas o reconhecimento, ou classificação, de padrões.

### 1.1.1 Problemas de Regressão

A regressão envolve a estimativa da relação entre variáveis por meio de métodos de indução de funções a partir de amostras destas variáveis. A estimativa da função aproximadora, ou regressora, pode ser realizada de várias formas, entre elas por meio de RNAs. No entanto, a estimativa por meio de RNAs tem algumas características interessantes, que as tornam particularmente atrativas para esta categoria de problemas, entre elas, a capacidade de aproximação não-linear com um número de parâmetros que cresce apenas linearmente com o número de variáveis.

Como exemplo, considere um processo de combustão industrial, para o qual deseja-se estimar o valor esperado de uma variável dependente, como, por exemplo, a pressão interna de uma caldeira, a partir de variáveis independentes, como aquelas relacionadas à combustão propriamente dita, como as vazões de combustível e de ar nos queimadores. O objetivo do regressor seria, neste caso, estimar a pressão, dados os valores das vazões. A construção, ou indução, do regressor seria realizada por meio de exemplos de amostras das vazões e da pressão. Assim, o problema de regressão envolve estimar uma função ou, em outras palavras, os seus parâmetros, que represente a relação entre variáveis dependentes e independentes.

Um exemplo de regressão de uma única variável é apresentado na Figura 1.3, em que os círculos representam os dados amostrados de $x$ (variável independente) e $y$ (variável dependente) e a linha contínua representa a resposta da função aproximadora $\hat{y} = f(x)$ em um determinado intervalo. Como pode ser observado na figura, a função obtida parece aproximar bem a relação entre as variáveis $x$ e $y$.

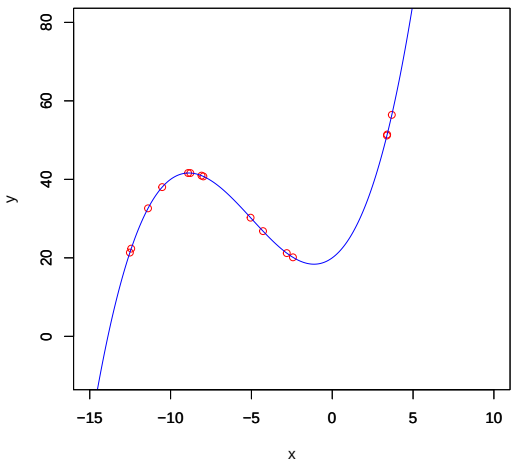



Figura 1.3: Exemplo de regressão. Círculos representam as amostras de $x$ e $y$ e a linha contínua a resposta da função estimada no intervalo de $-15$ a $+5$.

### 1.1.2 Problemas de Previsão

De maneira análoga à regressão, os problemas de previsão também visam a estimar uma relação entre variáveis, porém, neste caso há uma relação temporal entre as variáveis dependentes e independentes. Considere, por exemplo, o problema de prever o valor de fechamento da bolsa de valores de São Paulo, o IBOVESPA, que, neste exemplo, é a variável dependente. Quais são os fatores, ou variáveis, que influenciam o IBOVESPA? Podemos pensar em vários, por exemplo os preços das *commodities* que influenciam os índices majoritários que compõem o IBOVESPA, como as ações da Petrobras, Vale e as outras chamadas *blue chips*. Assim, o valor do barril de petróleo, chamado de Brent, negociado na bolsa de Londres pode ser considerado uma variável independente para a construção do estimador que terá como objetivo prever valores futuros do IBOVESPA, como o valor de fechamento do dia seguinte. O modelo de previsão poderá, portanto, ser composto por esta e por outras variáveis que influenciem o índice. Porém, em problemas de previsão deve-se considerar também os atrasos (*lags*) de tempo em que uma variável influencia a outra, ou seja, uma variação no preço do Brent hoje, ou de qualquer outra variável independente, pode levar algum tempo para ser incorporado ao IBOVESPA. Portanto, em problemas de previsão, as variáveis de entrada são incorporadas ao modelo com atrasos de tempo diferentes, de acordo com o tempo que cada uma delas leva para influenciar a saída. Modelos de previsão podem ser descritos de maneira genérica como na expressão da Equação 1.1.

$$\hat{y}(t+1) = \hat{f}(y(t), y(t-1), \cdots, x_1(t), x_1(t-1), \cdots, x_n(t), x_n(t-1), \cdots) \quad (1.1)$$

em que $t, t-1, \cdots$, indicam os instantes de tempo em que a variável é amostrada, $y(\cdot)$ é a variável de saída e $x_1(\cdot), \cdots, x_n(\cdot)$ as variáveis de entrada.

A função aproximadora, ou modelo, composta conforme conforme Equação 1.1, as variáveis de entrada e a saída realimentada serão consideradas naqueles instantes de tempo em que têm maior influência no valor seguinte da saída. A identificação dos *lags* entre as entradas e a saída pode ser realizada por meio de algum conhecimento prévio sobre o problema, por meio de experimentos ou através de análise dos dados, utilizando técnicas como a correlação cruzada [MdCT06].

### 1.1.3 Problemas de Classificação

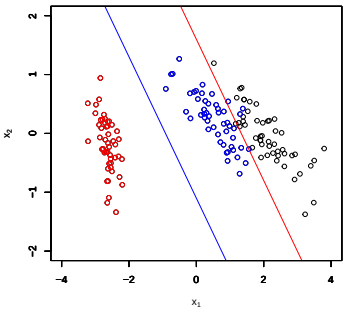
Os problemas de classificação envolvem a associação de uma amostra de entrada a uma classe conhecida. Da mesma forma que nos problemas de regressão e de previsão um vetor de entrada é associado a um valor da variável dependente, a resposta da função classificadora também resulta em um valor de saída que, neste caso, indica a classe resultante. De maneira análoga à previsão, o problema de classificação, ou RP, é também essencialmente um problema de regressão, já que envolve a busca por uma função discriminante, usualmente obtida por meio da partição do espaço de entrada em regiões distintas para cada uma das classes que definem o problema. Assim, em problemas de classificação a função a ser aproximada é discreta, em contraste com problemas de regressão e previsão que tratam tipicamente de aproximação de funções contínuas. Como amostras da mesma classe devem estar espacialmente próximas, estas caracterizarão regiões

para cada uma das classes. O objetivo da funçao discriminante é, portanto, identificar estas regiões, delimitá-las e identificar a região do espaço correspondente a uma nova amostra a ser classificada, indicando a classe correspondente.

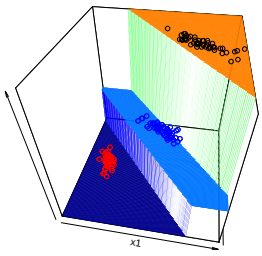
Um exemplo de classificação envolvendo três classes distintas é apresentado na Figura 1.4. As amostras que caracterizam as classes são apresentadas em cores distintas no gráfico, sendo vermelho para a classe 1, azul para a classe 2 e preto para a classe 3. Pode ser observado também que a classe 1 está mais afastada das demais, havendo alguma superposição entre as classes 2 e 3, o que pode levar a uma melhor discriminação da classe 1. Isto pode ocorrer em vários problemas reais e pode ser explicado pelo fato de as variáveis independentes, atributos $x_1$ e $x_2$, selecionadas para representar o problema são mais discriminativos em relação à classe 1. A superposição entre as amostras das classes 1 e 2 indica que poderá haver um maior erro na discriminação das amostras destas classes, especialmente na região de fronteira. Muitas vezes não é possível eliminar esta superposição, já que ela pode ser inerente ao problema e à sua representação, o que pode levar o classificador a ter um erro intrínsico ao separar amostras destas duas classes.

Com base, portanto, nas amostras apresentadas na Figura 1.4 deseja-se inicialmente identificar as regiões de cada classe. Uma análise simples da distribuição das amostras sugere que as classes podem ser separadas por retas, também indicadas na figura, e a classificação poderia ser feita identificando em qual das três regiões delimitadas foi mapeada a amostra a ser classificada. A função discriminante, neste caso, seria obtida pela composição das funções discriminantes correspondentes a cada uma das retas. Estas funções individuais deverão indicar se uma amostra está localizada acima ou abaixo da reta, respondendo com 1, por exemplo, caso esteja acima e com 0 caso contrário. Assim, para a classe 1 as respostas das funções das retas resultarão na tupla 00, já que as amostras desta classe estão abaixo de ambas as retas. De maneira análoga as tupas 01 e 11 indicarão amostras das classes 2 e 3. O classificador deverá, portanto, gerar como resposta um número indicando a classe da amostra de entrada com base nas tuplas acima. Isto pode ser feito simplesmente convertendo para decimal as respostas individuais das tuplas por meio da seguinte expressão: $\hat{y} = 2h_2(x_1, x_2) + h_1(x_1, x_2)$, em que $h_2(x_1, x_2)$ e $h_1(x_1, x_2)$ são as funções discriminantes correspondentes às duas retas e $x_1$ e $x_2$ seus argumentos. As respostas do classificador serão 0, 1 e 3 para as classe 1, 2 e 3, respectivamente, conforme indicado na superfície de resposta da Figura 1.4b.

A função das duas variáveis $x_1$ e $x_2$ apresentada na Figura 1.4b representa, portanto, a resposta do classificador e reforça a ideia de que o problema de classificação é essencialmente um problema de regressão, que visa a encontrar a função $\hat{y} = f(x_1, x_2)$ que seja capaz de discriminar as classes do problema. Neste caso particular, a função $f(x_1, x_2)$, obtida por inspeção da Figura 1.4b, é uma combinação linear das funções discriminantes $h_2(x_1, x_2)$ e $h_1(x_1, x_2)$, podendo a mesma ser representada de maneira genérica como $f(x_1, x_2) = w_2 h_2(x_1, x_2) + w_1 h_1(x_1, x_2)$, onde os parâmetros $w_2 = 2$ e $w_1 = 1$ foram obtidos experimentalmente.




(a) Amostras de três classes distintas, indicadas por cores diferentes e retas de separação que resultam na superfície de resposta da Figura 1.4b.




(b) Superfície resultante de um classificador que é capaz de discriminar as 3 classes da Figura 1.4.

Figura 1.4: Exemplo de um problema de classificação (RP) de três classes.

## 1.2 Redes Neurais Artificiais

As RNAs são formadas por elementos básicos, os neurônios artificiais, os quais executam funções matemáticas que representam modelos de neurônios biológicos. Dependendo de quais características reais tenham sido incorporadas ao modelo, estes podem variar quanto à complexidade e quanto à demanda por recursos computacionais para sua implementação. Não obstante, estruturas de RNAs utilizando o modelo simplificado de McCulloch e Pitts (modelo MCP [MP43]) são capazes de representar funções matemáticas bastante complexas, apesar de estes modelos serem bastante simples do ponto de vista matemático. Os modelos neurais artificiais partem, portanto, de conhecimentos do comportamento dos neurônios biológicos e da representação do mesmo na forma de expressões matemáticas.

Espera-se, no entanto, independemente da plausibilidade biológica, que as propriedades emergentes dos modelos neurais artificiais sejam capazes de reproduzir comportamentos característicos dos animais, os quais, além de serem dotados da capacidade de aprender de forma interativa, são também capazes de prever situações futuras, de classificar eventos, de agrupar informações, de induzir comportamentos e de lidar com informações parciais, distorcidas ou incompletas. Estas são algumas das capacidades dos animais que podem ser também reproduzidas pelas RNAs. Ao mesmo tempo em que a reprodução destas características emergentes pode ajudar no melhor entendimento de processos biológicos e cognitivos, a resolução de problemas associados à classificação de padrões, à previsão e ao agrupamento de dados por meio de *aprendizado* pode também proporcionar novas perspectivas para a solução de problemas práticos do nosso dia-a-dia.

Modelos biologicamente inspirados formam a base para a pesquisa em muitas áreas em torno da Neurociência, cujos avanços nos últimos anos proporcionaram grandes progressos em várias áreas do conhecimento. Há, no entanto, um compromisso entre a plausibilidade biológica de um modelo neural artificial e a sua

complexidade computacional. É claro que, quanto maior a fidelidade do modelo em relação aos neurônios biológicos, maior a demanda por recursos computacionais para a sua implementação. Assim, muitos dos modelos utilizados para resolução de problemas computacionais são versões simplificadas dos neurônios biológicos, com menor plausibilidade biológica, porém, com grande capacidade computacional, especialmente quando organizados na estrutura topológica de redes neurais artificiais.

## 1.3 Modelos de soma e limiar

Considere um problema de classificação simples cujo objetivo seja identificar se uma determinada amostra $x_t$ pertence à classe das amostras vermelhas ou das azuis, conforme Figura 1.5. Parece óbvio, por inspeção da figura, que um classificador simples para resolver este problema poderia avaliar a posição de $x_t$ em relação a um valor de limiar $\theta$ sobre o eixo $x$, como por exemplo $\theta = 3$. Caso $x_t \geq 3$, então a amostra seria classificada como pertencente à classe das amostras azuis e, caso $x_t < 3$ então a amostra seria classificada como pertencente à classe das amostras vermelhas. A função do classificador também pode ser descrita através da avaliação do sinal da operação $x_t - 3$, ou seja, caso $x_t - 3 \geq 0$ então $x_t$ pertence à classe das amostras azuis e caso $x_t < 0$ então $x_t$ pertence à classe das amostras vermelhas. A operação $x_t - 3$ é, na verdade, a distância com sinal de $x_t$ em relação ao limiar $\theta$. Neste caso, a função de decisão $f(u)$ deve receber como argumento a distância $u = x_t - 3$ e sobre esta realizar a classificação com base em seu sinal. A Figura 1.5 também mostra o resultado da aplicação desta função de classificação para o intervalo $0 \leq x_t \leq 6$, o que resultou em uma resposta de classificação equivalente a uma função degrau. Neste caso, $f(u) = 0$ indica classe vermelha e $f(u) = 1$ indica classe azul. O comportamento do modelo artificial de McCulloch e Pitts [MP43] é semelhante ao apresentado no exemplo da Figura 1.5.

O exemplo descrito na Figura 1.5 mostra um classificador simples de uma única variável $x$ baseado em uma função de ativação $f(u)$ do tipo degrau, em que $u$ é uma medida de distância com sinal entre a amostra que se deseja classificar e um separador, caracterizado neste caso pelo limiar $\theta$. A resposta do modelo de McCulloch e Pitts [MP43], representado de maneira esquemática na Figura 1.6, baseia-se também na aplicação de uma função de ativação $f(u)$ sobre o argumento $u$, o qual é também uma medida de distância entre a amostra de entrada e um hiperplano, caracterizado pelo separador. A medida de distância, neste caso, é o produto interno $u = w_1x_1 + w_2x_2 + \cdots + w_nx_n$ entre o vetor de entrada $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$ e o vetor de pesos $\mathbf{w} = [w_1, w_2, \cdots, w_n]^T$, que caracteriza o hiperplano separador. Sobre o resultado deste produto é aplicada a função de ativação $f(u)$, a qual pode assumir várias formas, entre elas a função degrau representada na Equação 1.2, o que resulta em um modelo com comportamento análogo ao classificador da Figura 1.5.

$$f(u) = \begin{cases} 1 & u \geq \theta \\ 0 & u < \theta \end{cases} \tag{1.2}$$

em que $u = \sum_i w_i x_i$.

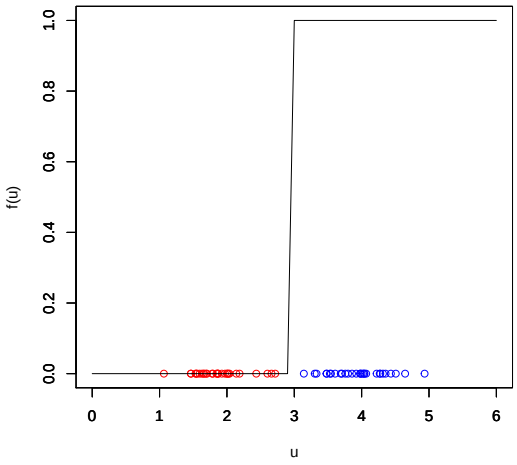



Figura 1.5: Problema de classificação simples cujo objetivo é classificar uma amostra arbitrária como pertencente à classe das amostras vermelhas ou azuis.

## 1.4 Estrutura de uma rede neural artificial

Uma RNA é caracterizada por uma estrutura de neurônios artificiais interconectados, os quais executam individualmente funções como aquela descrita na Equaçao 1.2. A forma da função de ativação $f(u)$ pode variar de acordo com o modelo adotado, porém, o seu argumento $u$ será tipicamente caracterizado pela soma dos elementos $x_i$ do vetor de entrada $\mathbf{x}$, multiplicados pelos pesos correspondentes $w_i$ do vetor $\mathbf{w}$. Esta soma ponderada, que pode ser representada pelo produto interno $u = \sum_i w_i x_i = \mathbf{w}^T\mathbf{x}$, é também uma medida de correlação, ou de proximidade, entre $\mathbf{w}$ e $\mathbf{x}$.

Uma RNA do tipo *feed-forward* (alimentada para frente) é representada de forma esquemática na Figura 1.7. Esta estrutura de duas camadas recebe como entradas os elementos do vetor $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$, o qual é propagado até a saída por meio dos neurônios $h_i(\mathbf{x}, \mathbf{z}_i)$ da camada intermediária, em que $\mathbf{z}_i$ representa o seu vetor de pesos. Assim, cada neurônio da camada intermediária executa a função $h_i(u_i)$, análoga àquela da Equação 1.2 em que $u_i = \sum_i z_i x_i$. Os $p$ vetores de pesos dos neurônios da camada intermediária compõem as colunas da matriz de pesos $\mathbf{Z}$ de dimensões $n \times p$. A propagação do vetor de entrada $\mathbf{x}$ até as saídas dos neurônios da camada intermediária resulta no vetor $\mathbf{h} = [h_1(\mathbf{x}, \mathbf{z}_1), h_2(\mathbf{x}, \mathbf{z}_2), \cdots, h_p(\mathbf{x}, \mathbf{z}_p)]^T$, o qual será aplicado às entradas do neurônio de saída para o cálculo da saída da rede neural. A função que representa o mapeamento entre o espaço de entrada e o espaço formado pelos neurônios da camada intermediária será representada aqui como $\Phi_h(\mathbf{x}, \mathbf{Z})$, em que a matriz $\mathbf{Z}$ contém, em cada linha, os parâmetros de cada um dos neurônios desta camada. O mapeamento entre o espaço da camada intermediária e a saída da rede será representado aqui pela função $\Phi_o(\mathbf{h}, \mathbf{w})$, em que $\mathbf{w}$ é o vetor que contém os seus parâmetros, considerando-se, sem perda de generalidade, um

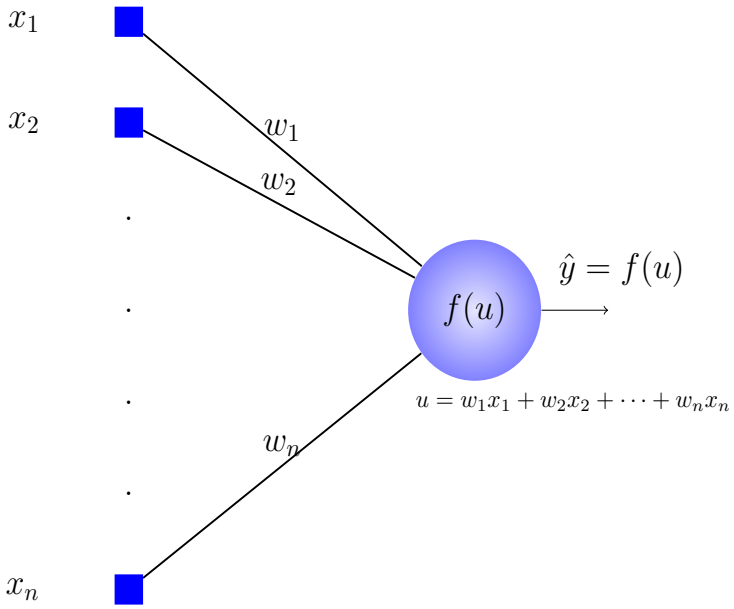



Figura 1.6: Representação esquemática do modelo neural de McCulloch e Pitts [MP43].

modelo de uma única saída, como aquele representado na Figura 1.7.

Assim, a saída deste modelo de RNA é obtida por meio de projeções sucessivas, inicialmente para a camada intermediária, $R^n \rightarrow R^p$, e posteriormente para a saída, $R^p \rightarrow R^1$. De maneira geral, $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$ representa a função executada pela RNA da Figura 1.7, cujo argumento é o vetor de entrada $\mathbf{x}$ e os parâmetros $\mathbf{Z}$ e $\mathbf{w}$ são obtidos durante o treinamento. Uma estrutura como esta é um aproximador universal de funções contínuas, conforme demonstrado formalmente no trabalho de Cybenko [Cyb89].

### 1.4.1 Representação matemática

A estrutura em camadas da Figura 1.7 pode ser descrita na forma de uma função composta, conforme representado na Equação 1.3.

$$f(\mathbf{x}, \mathbf{z}_1 \cdots \mathbf{z}_p, w_1 \cdots w_p) = \Phi_o \left( h_1(\mathbf{x}, \mathbf{z}_1)w_1 + \cdots + h_p(\mathbf{x}, \mathbf{z}_p)w_p + \beta \right) \tag{1.3}$$

$$f(\mathbf{x}, \mathbf{z}_1 \cdots \mathbf{z}_p, w_1 \cdots w_p) = \Phi_o \left( \sum_{i=1}^{p} h_i(\mathbf{x}, \mathbf{z}_i)w_i + \beta \right) \tag{1.4}$$

onde $p$ é o número de neurônios da camada escondida, $\Phi_o(\cdot)$ é a função de ativação do neurônio de saída, $h_i(\cdot)$ é função de ativação do neurônio $i$ da camada escondida, $w_i$ é o peso da conexão do neurônio $i$ ao neurônio de saída, $\beta$ é o termo de polarização do neurônio de saída, $\mathbf{w} = [w_1, \cdots, w_p]^T$ é o vetor de pesos do neurônio de saída e $\mathbf{Z} = [z_1, \cdots, z_p]^T$ a matriz de pesos da camada intermediária.

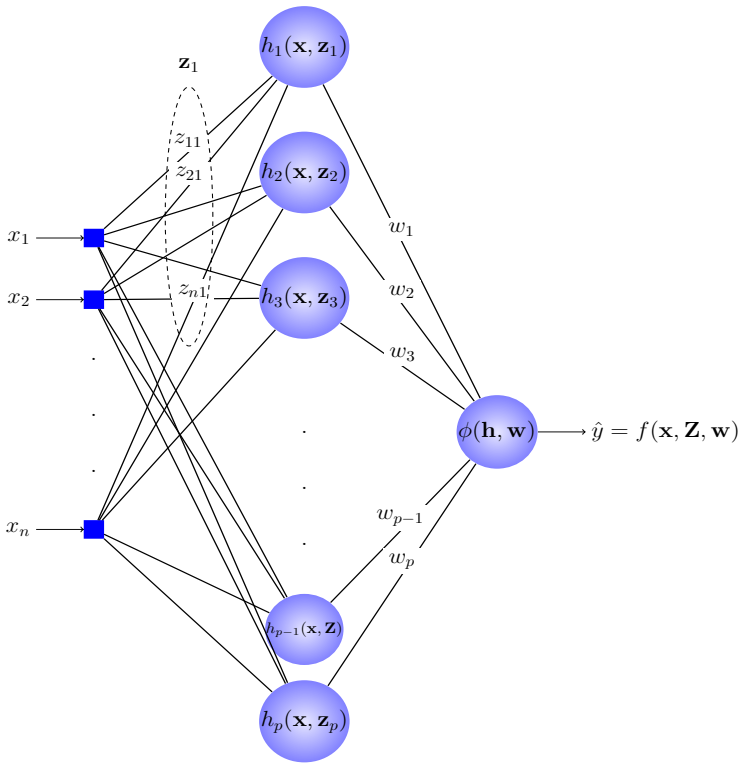



Figura 1.7: Representação geral de uma rede neural de uma saída e duas camadas.

Conforme representado nas Equações 1.3 e 1.4 uma RNA pode ser descrita na forma de composição de funções, usualmente não lineares, cujos argumentos são combinações lineares das suas entradas. Os coeficientes da combinação linear são os parâmetros dos neurônios, os quais são obtidos por meio de aprendizado, conforme será descrito conceitualmente na seção seguinte.

## 1.5 Aprendizado de Redes Neurais Artificiais

A expressão $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$ representa, de maneira geral, a RNA da Figura 1.7, em que os elementos da matriz $\mathbf{Z}$ e do vetor $\mathbf{w}$ são os únicos parâmetros que determinam o comportamento das funções $h_i(\cdot)$ e $\phi(\cdot)$. Assim, para que haja aprendizado, os parâmetros em $\mathbf{Z}$ e em $\mathbf{w}$ deverão ser modificados para que a RNA seja adaptada visando a representar uma determinada função a ser induzida a partir de um conjunto de dados amostrados. O aprendizado de redes neurais artificiais envolve, portanto, a adaptação de seus parâmetros por meio de um processo interativo com o meio externo, de forma que uma determinada função $f_g(\mathbf{x})$ seja representada na forma induzida $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$.

Visando a simplificar a sua representação, em vários dos capítulos seguintes adotaremos também a expressão $f(\mathbf{x}, \mathbf{w})$ para representar a função executada pela RNA, em que $\mathbf{w}$ é o vetor que contém a concatenação de todos os parâmetros do modelo. Assim, o objetivo do aprendizado é, portanto, encontrar o vetor

de parâmetros $\mathbf{w}$ que satisfaça a um determinado critério, ou função-objetivo. Obviamente, para que haja aprendizado é preciso que o desempenho do modelo, segundo um critério pré-determinado, melhore gradualmente através da adaptação de $\mathbf{w}$. O critério que determinará a adaptação dos parâmetros é usualmente representado por uma função-objetivo, que quantifica quão distante está a saída $\hat{y}_i = f(\mathbf{x}_i, \mathbf{w})$ do valor alvo, normalmente representado por valores de saída $y_i$ de um conjunto de amostras $\tau = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Assim, considerando-se que $\tau$ seja representativo, espera-se que o aprendizado resulte na aproximação de $y_i$ pela RNA, ou seja, que $f(\mathbf{x}_i, \mathbf{w}) \approx f_g(\mathbf{x}_i)\ \forall \mathbf{x}_i$. Como não há, usualmente, informação *a priori* sobre a forma de $f_g(\mathbf{x}_i)$, os parâmetros de $f(\mathbf{x}_i, \mathbf{w})$ são adaptados, ou aprendidos, com base na informação contida em $\tau$, de tal forma que $f(\mathbf{x}_i, \mathbf{w}) \approx f_g(\mathbf{x}_i)\ \forall \mathbf{x}_i \in \tau$. Assumindo-se representatividade de $\tau$ e $N$ suficientemente grande, espera-se que a aproximação em $\tau$ garanta $f(\mathbf{x}, \mathbf{w}) \approx f_g(\mathbf{x})$ no domínio de $\mathbf{x}$. Este é o princípio da minimização do risco empírico [Vap95], que será discutido também em outros capítulos deste livro. Com este objetivo, uma possível função de custo, frequentemente utilizada por sua simplicidade, é o erro médio quadrático, representado pela expressão $J = \frac{1}{N}\sum_i (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$.

O aprendizado de modelos neurais, como o representado na Figura 1.7, é normalmente realizado segundo o paradigma do Aprendizado Supervisionado, em que a resposta $f(\mathbf{x}_i, \mathbf{w})$ do modelo é calculada e comparada com a saída esperada $y_i$. Por meio da função de custo $J(y, f(\mathbf{x}, \mathbf{w}))$, o supervisor, externo ao modelo, determinará quão distante $f(\mathbf{x}_i, \mathbf{w})$ está de $y_i$, assim como a direção de ajuste de $\mathbf{w}$ que resultará na diminuição desta distância. Com base na informação de direção obtida por meio da função $J(y, f(\mathbf{x}, \mathbf{w}))$, o supervisor, que incorpora o algoritmo de aprendizado, fará o ajuste de $\mathbf{w}$ para todos os pares $(\mathbf{x}_i, y_i)$ do conjunto $\tau$. Não obstante, este critério não garante o objetivo maior do aprendizado que é aproximar a função geradora $f_g(\mathbf{x})$ dos dados e não somente entre aqueles valores $y_i$ presentes no conjunto de amostras, que são na verdade uma representação de $f_g(\mathbf{x})$. No caso extremo em que o conjunto $\tau$ é representativo e suficientemente grande ($N \to \infty$), teremos $f(\mathbf{x}_i, \mathbf{w}) \approx f_g(\mathbf{x}_i) \forall \mathbf{x}_i \in R^n$ ao final do aprendizado. Assim, o objetivo do treinamento é, na verdade, aproximar $f_g(\mathbf{x})$ em todo o domínio da variável $\mathbf{x}$ e não somente entre os elementos do conjunto de treinamento $\tau$. O grande desafio dos algoritmos de treinamento surge devido ao fato que, em situações reais o tamanho $N$ do conjunto de amostras é limitado e, mesmo com esta restrição, deve-se buscar a aproximação $f(\mathbf{x}, \mathbf{w}) \approx f_g(\mathbf{x})$.

As implicações relativas ao tamanho de $\tau$ e à sua representatividade na generalidade da função $f(\mathbf{x}, \mathbf{w})$ serão discutidas nos capítulos seguintes. Por ora, o leitor deve ter em mente que o aprendizado de redes neurais visa a induzir os parâmetros $\mathbf{w}$ através das amostras do conjunto $\tau$ e que o seu objetivo é obter um modelo que aproxime aquela função geradora dos dados, evitando-se, assim, que $f(\mathbf{x}, \mathbf{w})$ se especialize no conjunto $\tau$ somente, o qual pode ter algum viés que o afaste de $f_g(\mathbf{x})$.

## 1.6 Indução de Funções

O aprendizado de redes neurais artificiais pode ser representado pelo problema geral de indução de funções em que se deseja induzir o conjunto de parâmetros $\mathbf{w}$ da função genérica $f(\mathbf{x}, \mathbf{w})$ a partir de um conjunto de amostras $\tau$ e

de uma função-objetivo $J(y, f(\mathbf{x}, \mathbf{w}))$. Assumindo-se representatividade de $\tau$, este deverá conter informações sobre o comportamento de $f_g(\mathbf{x})$, os quais serão utilizadas para aproximá-la. Assim, espera-se que ao final do treinamento a função $f(\mathbf{x}, \mathbf{w})$ seja capaz de **imitar** o comportamento de $f_g(\mathbf{x})$ de tal forma que $f(\mathbf{x}, \mathbf{w}) \approx f_g(\mathbf{x})\ \forall \mathbf{x} \in R^n$. Espera-se que a interação entre $f(\mathbf{x}, \mathbf{w})$ e $f_g(\mathbf{x})$ por meio de representações do comportamento de $f_g(\mathbf{x})$ incorporadas ao conjunto de dados $\tau$, leve $f(\mathbf{x}, \mathbf{w})$ a se comportar como $f_g(\mathbf{x})$ em decorrência do ajuste de $\mathbf{w}$.

Uma outra forma de aproximar $f_g(\mathbf{x})$ é através da descoberta do operador que governa a função, no entanto, esta forma de modelagem é muitas vezes mais custosa, já que requer conhecimento sobre a estrutura dos processos que caracterizam $f_g(\mathbf{w})$. Quanto maior a complexidade da função geradora maior será o custo de encontrar um modelo que reproduza de maneira realística o seu operador. Assim, especialmente em problemas de maior complexidade, a reprodução do operador que governa $f_g(\mathbf{x})$ se torna inviável, restando como alternativa a indução da função aproximadora $f(\mathbf{x}, \mathbf{w})$ a partir do conjunto de amostras representativas $\tau$. Assim, a qualidade da aproximação resultante de $f(\mathbf{x}, \mathbf{w})$ dependerá essencialmente dos seguintes fatores:

- **universo de funções candidatas**: Para o caso particular de redes neurais artificiais, as funções candidatas são representadas pelo conjunto de modelos que podem ser obtidos a partir de uma estrutura de rede neural pré-definida, representada na forma geral como $f(\mathbf{x}, \mathbf{w})$. Uma vez definida a estrutura da rede neural, o universo de funções candidatas é determinado por todos os valores possíveis que podem assumir os elementos de $\mathbf{w}$.
- **tamanho e representatividade do conjunto de amostras**: Quanto maior o conjunto de amostras, melhor será a representividade de $f_g(\mathbf{x})$ que servirá como base para a indução da função aproximadora $f(\mathbf{x}, \mathbf{w})$.
- **princípio de indução adotado**: O princípio de indução, representado na forma de uma função-objetivo, determinará, então, o critério a ser adotado para selecionar uma das funções candidatas com base no conjunto de amostras que pode ser, por exemplo, a minimização do erro sobre o conjunto de amostras.
- **algoritmo que implementa o princípio de indução**: Finalmente, a seleção do modelo será realizada pelo algoritmo que implementará o princípio de indução ou, em outras palavras, que resolverá o problema de otimização caracterizado pela função-objetivo. Algoritmos comuns são, por exemplo, Gradiente Descendente, Levenberg-Marquadt, Algoritmos Evolucionários, entre outros [NW06a].

Sem perda de generalidade, considerando-se uma rede de duas camadas, o problema de indução de funções a partir de uma amostra finita de dados $\tau$, que caracteriza o aprendizado de RNAs, pode ser apresentado da seguinte forma:

> *Dado um conjunto de $N$ amostras $\tau = \{\mathbf{x}_i, y_i\}_{i=1}^N$, encontre os parâmetros $\mathbf{Z}$ e $\mathbf{w}$ que minimizem a função de custo $J(D, f(\mathbf{x}, \mathbf{Z}, \mathbf{w}))$.*

Como resultado do ajuste dos parâmetros contidos em $\mathbf{Z}$ e $\mathbf{w}$ espera-se que a função $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$ se aproxime da função geradora dos dados $f_g(\mathbf{x})$ ou, em outras

palavras, que esta seja induzida a partir da informação contida no conjunto de amostras $\tau = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$. O treinamento de RNAs é tipicamente caracterizado pelo ajuste simultâneo de $\mathbf{Z}$ e $\mathbf{w}$. Outros modelos de máquinas de aprendizado, como as redes de Funções de Base Radiais (RBFs) [BL88], Máquinas de Aprendizado Extremo (ELMs) [HZS04] e as Máquinas de Vetores de Suporte (SVM) [CV95] tratam os problemas de indução de $\mathbf{Z}$ e $\mathbf{w}$ separadamente. Nestes modelos, o treinamento é realizado obtendo-se inicialmente $\mathbf{Z}$ para então, a partir das projeções na camada intermediária em função de $\mathbf{Z}$, obter-se $\mathbf{w}$.

### 1.6.1 Função da Camada Intermediária

O Teorema de Cover [Cov65] teve uma grande importância no desenvolvimento das RNAs e de outros modelos de máquinas de aprendizado, como as redes RBF [BL88] e as SVMs [BGV92]. Visando à resolução de problemas usualmente não-lineares, estes modelos buscam a solução do problema de aprendizado por meio de mapeamentos sucessivos até que o mesmo se torne tratável pela camada de saída. Uma analogia com a forma de construção de um polinômio $p(x)$ pode ajudar a entender melhor o papel do mapeamento em camadas. Considere, por exemplo, um polinômio de grau quatro, descrito de forma genérica como $p(x) = w_4x^4 + w_3x^3 + w_2x^2 + w_1x^1 + w_0x^0$. A construção do polinômio pode ser vista como se o seu argumento $x$ fosse mapeado em um vetor de dimensão 5 por meio dos operadores $x^4$, $x^3$, $x^2$, $x^1$ e $x^0$. Assim, por exemplo, se $x_i = 2$ então o vetor obtido com o mapeamento é $\mathbf{h}_i = [16, 8, 4, 2, 1]^T$, o qual é combinado linearmente com o vetor de parâmetros $\mathbf{w} = [w_4, w_3, w_2, w_1, w_0]^T$ para obter a resposta $p(x)$ do polinômio. Na Figura 1.8 é apresentada uma representação esquemática da construção deste polinômio em uma estrutura de duas camadas. A camada de saída, ou seja, o somador, executa uma operação linear sobre a projeção não linear na camada intermediária, a qual é realizada por meio dos operadores $x^4$, $x^3$, $x^2$, $x^1$ e $x^0$. O resultado desta projeção não-linear é a linearização do problema, o que permite que a aproximação da função não-linear possa ser realizada com uma operação (linear) de soma pela camada de saída.

O problema de aproximação polinomial apresentado na Figura 1.8 é análogo ao problema geral de classificação não-linear de padrões e de aproximação de funções com RNAs, para os quais, da mesma forma, a solução do problema pode ser representada por uma estrutura em camadas. Assim, considere um modelo geral com uma camada intermediária representado pela função genérica $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$, conforme descrito nas seções anteriores. As duas funções $\Phi_h(\mathbf{x}, \mathbf{Z})$ e $\Phi_o(\mathbf{x}, \mathbf{w})$ que compõem $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$ são responsáveis pelos dois mapeamentos sucessivos que permitem que $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$ seja um mapeador universal de funções. A função $f(\mathbf{x}, \mathbf{Z}, \mathbf{w})$ é na verdade uma função composta, caracterizada pelas duas funções $\Phi_h(\mathbf{x}, \mathbf{Z})$ e $\Phi_o(\mathbf{h}, \mathbf{w})$, conforme mostrado de forma esquemática na Figura 1.9.

## 1.7 Histórico

O desenvolvimento das áreas de RNAs e RP sofreu influências de várias outras áreas, como Estatística, Neurofisiologia, Psicologia, etc, ao longo das últimas décadas. Assim, ao traçar uma linha do tempo da área de RNAs, é importante considerar avanços em outras áreas e a influência que elas tiveram na grande área

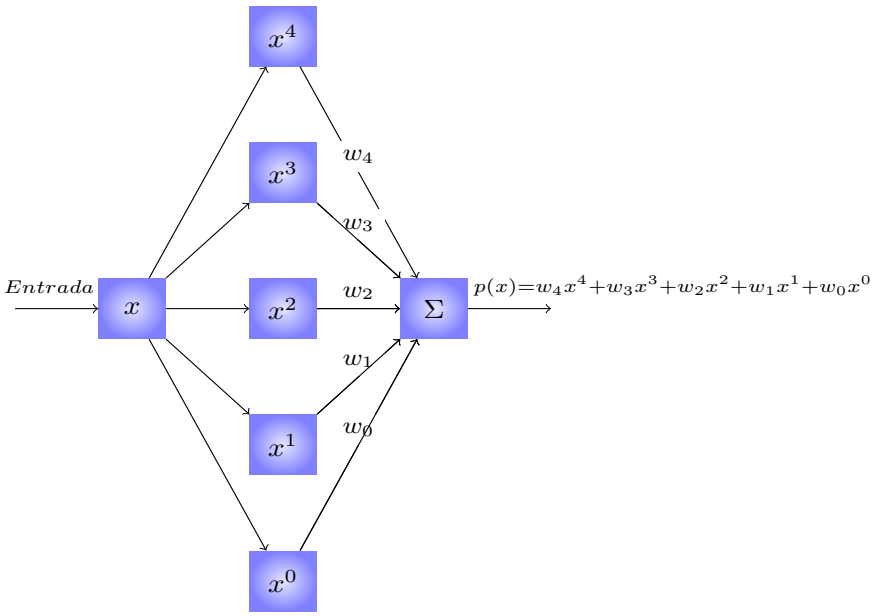



Figura 1.8: Representação esquemática da construção do polinômio $p(x) = w_4x^4 + w_3x^3 + w_2x^2 + w_1x^1 + w_0x^0$ na forma de uma estrutura em camadas. A camada intermediária tem por função a linearização do problema.

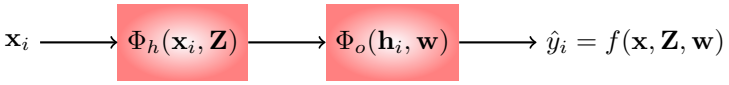



Figura 1.9: Representação esquemática do mapeamento em camadas em RNAs.

de RNAs e Aprendizado de Máquinas. Apesar de as décadas de 1940 e 1950 serem consideradas marcos iniciais da área, devido às descrições dos modelo de McCulloch e Pitts [MP43] e do Perceptron Simples [Ros58], não é possível desvincular totalmente as descrições destes modelos de trabalhos anteriores em áreas como Psicologia Cognitiva e Neurofisiologia. Assim, o aprendizado de associações entre atividades de grupos neuronais distintos, base para os modelos de redes neurais artificiais, foi abordado pela primeira vez no livro de William James em 1892 [Jam85]. James descreve em seu trabalho hipóteses para regras de adaptação sináptica que formaram a base para a descrição da Regra de Hebb em 1949 [Heb49]:

> *Quando dois processos cerebrais elementares estiveram ativos conjuntamente, ou em sucessão imediata, um deles, de forma recorrente, tende a propagar a sua ativação ao outro. (Traduzido de [And95].)*

A Regra de Hebb [Heb49] é mais específica ao referenciar células nervosas e axônios, no entanto, se baseia nos mesmos princípios apresentados por James em 1892 e deu origem a um dos primeiros métodos de aprendizado de RNAs. Segundo a mesma, uma sinapse $w_{ij}$ entre os neurônios $x_j$ e $y_i$ é adaptada por meio do produto entre as ativações pré e pós-sinápticas, $x_j$ e $y_i$, ou seja, $w_{ij} \propto$

$y_i x_j$. Os modelos neurais de memórias associativas lineares e recorrentes, entre eles o modelo de Hopfield [Hop82], que surgiram entre as décadas de 1960 e 1980, são baseados em Aprendizado Hebbiano [And68, And70, Koh74, Kos88].

Um diagrama esquemático da representação de memória associativa descrita por William James é apresentada na Figura 1.10, em que as conexões representam as relações de influência entre neurônios, a qual seria reforçada, segundo James e Hebb, de acordo com a co-ocorrência de ativações entre pares de neurônios. Segundo a representação de James, que cada neurônio se conecta a todos os outros do grupo vizinho e, consequentemente, sua ativação depende da força destas conexões. Em outro trecho do seu livro, James escreve também:

> *A magnitude da atividade em um ponto qualquer do cortex cerebral é a soma das tendências de todos os outros pontos que descarregam nele... (Traduzido de [And95].)*

A leitura deste segundo trecho do trabalho de William James sugere também que a ativação de um determinado neurônio depende da soma das ativações recebidas por este neurônio. Este é exatamente o princípio do nosso entendimento atual do comportamento funcional de um neurônio biológico, em que a sua ativação é função do somatório das ativações anteriores, ou pré-sinápticas. Assim, William James, pesquisador pioneiro da área de Psicologia, estabeleceu as bases para o desenvolvimento de modelos matemáticos de neurônios biológicos, que surgiram a partir de meados do século XX.

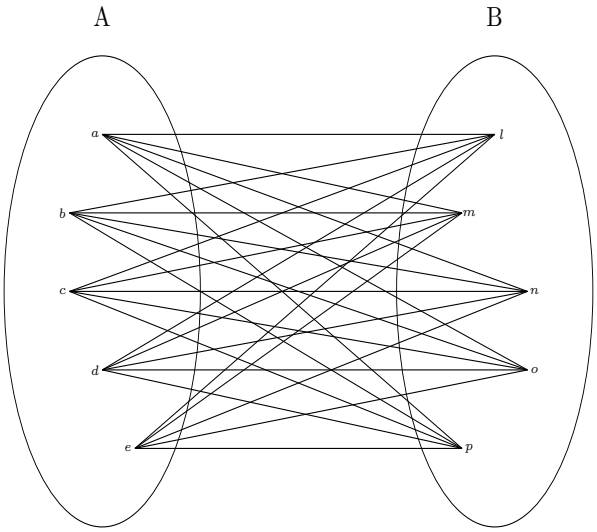



Figura 1.10: Representação esquemática do modelo de memória associativa de William James [Jam85].

O trabalho de McCulloch e Pitts [MP43], publicado em 1943, durante a primeira guerra mundial, descreve o modelo de soma-e-limiar na forma como ele é atualmente adotado nos principais modelos de RNAs. Uma representação esquemática do modelo MCP foi apresentada na Figura 1.6 e sua descrição matemática na Equação 1.2. A sua ativação é, assim, função da soma das ativações dos neurônios anteriores (pré-sinápticos), ponderados pelos pesos das conexões

(sinapses) entre eles, princípio que foi explorado anteriormente por Willian James [Jam85]. McCulloch era neurofisiologista e Pitts pesquisador em Lógica Matemática, assim, o trabalho desenvolvido por eles teve influência destas duas áreas e também do período em que foi apresentado, em meio à segunda guerra mundial, quando os primeiros computadores digitais foram construídos. O trabalho apresenta várias configurações de redes de neurônios MCP visando à implementação de funções lógicas (Booleanas), o que deixa claro o contexto em que os autores desenvolveram o trabalho. Este enfoque sugere que os computadores digitais que surgiam naquela época poderiam ser implementados utilizando estruturas de neurônios artificiais.

É interessante observar que a descrição do neurônio MCP deu origem também a uma sub-área de Sistemas Lógicos, a Lógica de Limiar (*Threshold Logic*), que envolvia, basicamente a implementação de funções Booleanas utilizando neurônios do tipo MCP. Durante algumas décadas, o assunto foi abordado dentro das disciplinas Sistemas Lógicos e Eletrônica Digital e é possível encontrar em vários livros da época capítulos dedicados exclusivamente ao assunto [Koh78]. A partir do final dos anos 1970, o assunto deixou de ser abordado dentro da área de Sistemas Digitais e passou a ser adotado como um assunto à parte, com o surgimento da grande área de RNAs.

McCulloch e Pitts descreveram o neurônio MCP como um modelo capaz de implementar funções lógicas, porém, as estruturas apresentadas eram estáticas, não tendo sido apresentado pelos autores nenhum método de aprendizado para adaptá-las. A primeira teoria concreta para a adaptação sináptica surgiu com o trabalho de Hebb [Heb49], cujos fundamentos haviam sido apresentados anteriormente por Willian James [Jam85].

### 1.7.1 Aprendizado Hebbiano

A interpretação da Regra de Hebb para a adaptação de um modelo artificial envolve, basicamente, a atualização dos pesos por meio do produto cruzado entre as ativações anteriores e posteriores daquele peso. Assim, o peso $w_{ij}$ entre os neurônios $x_j$ e $y_i$ deve ser adaptado de acordo com a regra $w_{ij}(t+1) = w_{ij}(t) + \eta x_j y_i$, em que $\eta$ é uma constante de proporcionalidade e $t$ o instante de tempo atual. Considere, por exemplo, que $x_j, y_i \in \{0,1\}$ e que os valores $\{0,1\}$ representem ativação ou não das variáveis $x_j$ e $y_i$. Assim, o peso final, após $t$ iterações, dependerá da correspondência das ativações de $x_j$ e $y_i$, já que $w_{ij}(t+1) = w_{ij}(0) + \eta \sum_{k=1}^{t} x_j(k) y_i(k)$.

# Capítulo 2

# Nomenclatura e Estrutura de Dados

## 2.1 Estrutura de Dados

Neste capítulo serão apresentadas a estrutura de dados básica e a nomenclatura utilizadas ao longo deste livro. De uma maneira geral, as informações deste capítulo se referem a como os dados são armazenados na forma matricial, como os modelos são estruturados e qual a nomenclatura utilizada na sua representação.

### 2.1.1 Dados amostrados

Os dados de entrada são amostrados no espaço $\mathcal{X}$ e podem ter a eles um valor correspondente amostrado no espaço de saída $\mathcal{Y}$. Neste caso, o conjunto de dados $\tau = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ é composto por $N$ pares de vetores $(\mathbf{x}_i, \mathbf{y}_i)$ armazenados por linha nas matrizes $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2 \cdots \mathbf{x}_N]^T$ e $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2 \cdots \mathbf{y}_N]^T$, conforme representado nas Equações 2.1 e 2.2. As linhas de $\mathbf{X}$ contêm os $N$ vetores $\mathbf{x}_i = [x_{i1}, x_{i2} \cdots x_{in}]$ de entrada de dimensão $n$ e as linhas de $\mathbf{Y}$ contêm as $N$ amostras $\mathbf{y}_i = [y_{i1}, y_{i2} \cdots y_{im}]$ de saída de dimensão $m$, assim, para cada linha de $\mathbf{X}$, haverá uma linha correspondente em $\mathbf{Y}$. A Figura 2.1 mostra um exemplo de como as linhas de $\mathbf{X}$ e $\mathbf{Y}$ estão relacionadas.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Nn} \end{bmatrix} \tag{2.1}$$

onde $n$ é a dimensão do espaço de entrada e $N$ o número de amostras.

$$\mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ y_{N1} & y_{N2} & \cdots & y_{Nm} \end{bmatrix} \tag{2.2}$$

onde $m$ é a dimensão do espaço de saída.

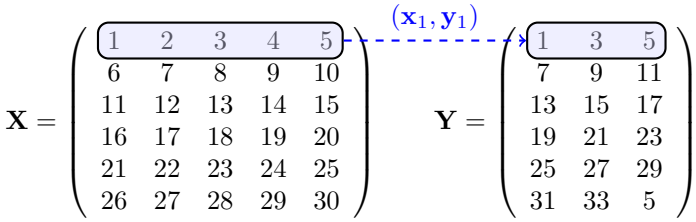



Figura 2.1: Correspondência entre as linhas de $\mathbf{X}$ e $\mathbf{Y}$, conforme Equações 2.1 e 2.2 para $N = 6$, $n = 5$ e $m = 3$.

### 2.1.2 Dados de treinamento

Nesta seção será feita uma distinção entre as formas de armazenamento dos dados amostrados e os de treinamento, particularmente de RNAs, visando à realização de operações matriciais. Modelos de neurônios, como o o modelo MCP de McCulloch e Pitts [MP43], são baseados em operações de soma e limiar, que resultam em discriminadores lineares, caracterizados por hiperplanos separadores na forma $\sum_{i=0}^{n} w_i x_i = 0$. Seja, por exemplo, o caso de um neurônio de duas entradas na forma $w_2 x_2 + w_1 x_1 + w_0 x_0 = 0$, definido no plano das variáveis $x_1 \times x_2$. Pode ser observado que o problema é definido por 3 parâmetros, $w_2$, $w_1$ e $w_0$, enquanto que os dados amostrados possuem somente duas dimensões, caracterizadas pelas variáveis $x_1$ e $x_2$. Em outras palavras, devido à forma como o problema é representado, a entrada $x_0$ não faz parte dos dados amostrados, sendo a mesma fixa e igual a 1. Assim, para que a operação de soma ponderada das entradas possa ser realizada na forma matricial $\mathbf{w} \cdot \mathbf{x}$, é necessário acrescentar uma coluna fixa em 1 à Equação 2.1. Em outras palavras, o número de parâmetros de um neurônio tipo MCP será sempre $n + 1$, onde $n$ é o número de variáveis, ou dimensão, do espaço de entrada. Portanto, ao longo dos capítulos que se seguem, toda vez em que se tratar de uma RNA, a matrix $\mathbf{X}$ dos dados de entrada, será representada na forma da Equação 2.3 com uma coluna adicional fixa em $+1$ e $n + 1$ será o número de parâmetros do modelo. Para simplificar, sempre que houver treinamento de uma RNA, os dados de entrada serão acrescidos da coluna adicional com $+1$. Esta regra valerá também para o caso de projeções em camadas, discutida na seções seguintes.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} & +1 \\ x_{21} & x_{22} & \cdots & x_{2n} & +1 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Nn} & +1 \end{bmatrix} \tag{2.3}$$

### 2.1.3 Parâmetros dos modelos

A forma geral para representar os modelos abordados neste livro é $f(\mathbf{x}, \mathbf{w})$, em que $\mathbf{x}$ é o vetor de entrada, argumento da função, e $\mathbf{w}$ o vetor que contém todos os seus parâmetros, independentemente de qual tipo de modelo adotado. Assim, caso o modelo seja um polinômio, $\mathbf{w} = [w_1, w_2, \cdots, w_n]^T$ conterá os coeficientes do polinômios ou, caso seja uma RNA, conterá os seus pesos. No entanto, uma representação em camadas, como das RNAs, requer nomenclatura diferenciada

para os parâmetros de cada uma das suas camadas, conforme representado na Figura 2.2. Assim, para uma estrutura de duas camadas, como da figura, a matriz de pesos da camada de entrada será representada por $\mathbf{Z}$, enquanto que a matriz de pesos da camada de saída será representada por $\mathbf{W}$.

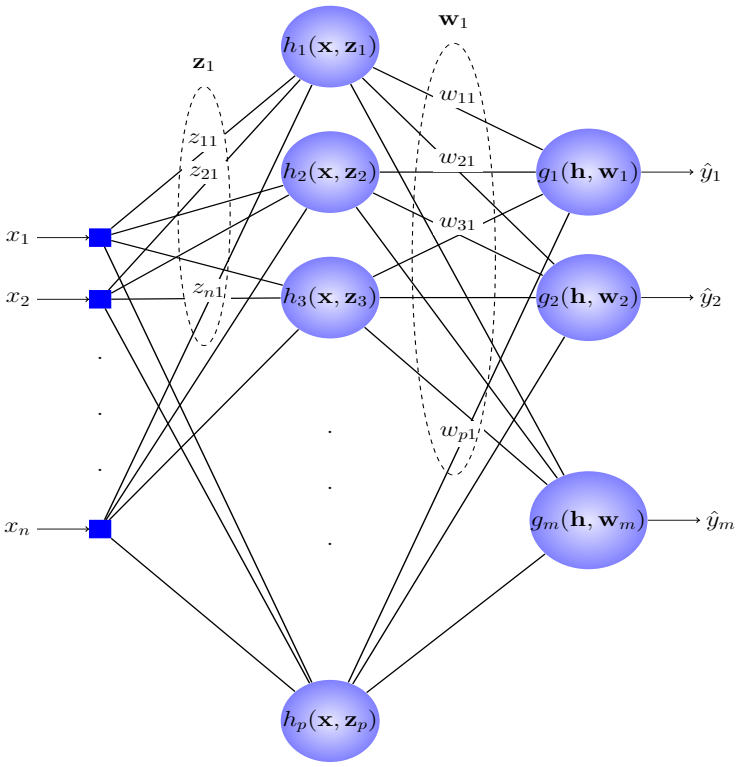



Figura 2.2: Representação geral de uma rede neural de uma saída e duas camadas com $n$ entradas e $m$ saídas.

A matriz de pesos $\mathbf{Z}$ da camada escondida com dimensões $n \times p$ é definida na Equação 2.4 e a matriz de pesos $\mathbf{W}$ da camada de saída com dimensões $p \times m$, em que $m$ é o número de saídas, é definida na Equação 2.5. Assim, as colunas de $\mathbf{Z}$ contêm os vetores de pesos de cada um dos $p$ neurônios da camada escondida. De forma análoga, as colunas de $\mathbf{W}$ contêm os vetores de pesos de cada um dos $m$ neurônios de saída.

$$\mathbf{Z} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1p} \\ z_{21} & z_{22} & \cdots & z_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{np} \end{bmatrix} \tag{2.4}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ w_{p1} & w_{h2} & \cdots & w_{pm} \end{bmatrix} \tag{2.5}$$

### 2.1.4 Projeção na camada intermediária

As dimensões das matrizes **X**, **Z** **W** foram definidas de forma tal a possibilitar e simplificar as operações matriciais para obtenção das respostas dos modelos. Assim, a projeção dos dados de entrada **X** no espaço da camada intermediária, caracterizado pelos $p$ neurônios representados na Figura 2.2, pode ser realizado com base no produto de **X** por **Z**, que resulta na matriz **U** de dimensões $N \times p$ representada na Equação 2.6.

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1p} \\ u_{21} & u_{22} & \cdots & u_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ u_{N1} & u_{N2} & \cdots & u_{Np} \end{bmatrix} \tag{2.6}$$

A matriz **U** contém as projeções lineares de **X** no espaço da camada intermediária, já que a operação $\mathbf{U} = \mathbf{X} \cdot \mathbf{Z}$ antecede a aplicação das funções de ativação não lineares $h_i(u_{ji})$, que caracterizam a projeção na camada intermediária. Assim, sobre a matriz **U** é aplicada a função de ativação comum a todos os neurônios da camada escondida, obtendo-se desta forma a matriz **H** que contém os mapeamentos não-lineares de todas as amostras de entrada no espaço da camada escondida. A matriz **H**, que tem mesma dimensão de **U**, ou seja, é uma matriz $N \times p$, pode ser obtida por meio da aplicação das funções de ativação $h_i(u_{ji})$ a todos os elementos de **U**, conforme Equação 2.7.

$$\mathbf{H} = \begin{bmatrix} h_1(u_{11}) & h_2(u_{12}) & \cdots & h_p(u_{1p}) \\ h_1(u_{21}) & h_2(u_{22}) & \cdots & h_p(u_{2p}) \\ \vdots & \vdots & \cdots & \vdots \\ h_1(u_{N1}) & h_2(u_{N2}) & \cdots & h_p(u_{Np}) \end{bmatrix} \tag{2.7}$$

em que $p$ corresponde à dimensão (número de funções) da camada intermediária.

Assim, a resposta do neurônio $i$ ao vetor de entrada $j$ pode ser representada de forma geral como $h_i(\mathbf{x}_j, \mathbf{z}_i)$, em que $\mathbf{z}_i$ é o vetor que contém os parâmetros do neurônio $i$. Toda projeção na camada intermediária depende, portanto, do tipo de função de ativação $h_i(\cdot)$ e dos parâmetros que a caracterizam. Uma vez obtida a matriz **H** tem-se um novo problema caracterizado pelas projeções no novo espaço. Para o diagrama da Figura 2.2, uma nova projeção deve ser realizada para a obtenção da resposta do modelo, conforme descrito a seguir.

### 2.1.5 Resposta do modelo

Uma nova operação matricial deve ser realizada para a obtenção da projeção de **H** no espaço de saída **Y**. De maneira análoga à projeção de **X** em **H**, a projeção de **H** em **Y** é realizada multiplicando-se inicialmente **H** por **W** e sobre o resultado deste produto aplicando-se as funções de ativação $g(\cdot)$ dos neurônios de saída.

A saída $j$ do modelo é representada por $\hat{y}_{ij} = g_j(\mathbf{h}_i, \mathbf{w}_j)$ para cada projeção $\mathbf{h}_i$ de $\mathbf{x}_i$ em **H**, resultando na matriz de aproximação $\hat{Y}$ apresentada na Equação 2.8. O argumento $\mathbf{h}_i$ de $g_j(\mathbf{h}_i, \mathbf{w}_j)$ corresponde à linha $i$ da matriz de mapeamento **H**. O elemento $\hat{y}_{ij}$ corresponde à saída $j$ do modelo para o padrão de entrada $\mathbf{x}_i$, conforme Equação 2.8. A obtenção de **Y** é análoga à obtenção de **H** descrita na Seção 2.1.4, ou seja, $\hat{\mathbf{Y}} = G(\mathbf{H} \cdot \mathbf{W})$, lembrando-se que, de

maneira análoga a $\mathbf{X}$, uma coluna adicional fixa em +1 deve ser adicionada a $\mathbf{H}$ para que a operação matricial seja realizada.

$$\hat{\mathbf{Y}} = \begin{bmatrix} \hat{y}_{11} & \hat{y}_{12} & \cdots & \hat{y}_{1m} \\ \hat{y}_{21} & \hat{y}_{22} & \cdots & \hat{y}_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ \hat{y}_{N1} & \hat{y}_{N2} & \cdots & \hat{y}_{Nm} \end{bmatrix} \tag{2.8}$$

### 2.1.6 Estimativa do erro de saída

Uma vez obtida a matriz $\hat{\mathbf{Y}}$ é possível calcular o erro de estimativa do modelo por amostra ou para todo o conjunto de dados $\tau = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$. O erro estimado será utilizado para avaliar a qualidade da resposta do modelo e para o ajuste dos seus parâmetros durante o treinamento. A matriz de erros de saída $\mathbf{E} = [e_{ij}]$ é obtida por meio do cálculo do desvio $\mathbf{E} = (\mathbf{Y} - \hat{\mathbf{Y}})$ entre a matriz de saída $\mathbf{Y}$ e a matriz correspondente $\hat{\mathbf{Y}}$ gerada pelo modelo, conforme apresentado na Equação 2.9.

$$\mathbf{E} = \begin{bmatrix} (y_{11} - \hat{y}_{11}) & (y_{12} - \hat{y}_{12}) & \cdots & (y_{1m} - \hat{y}_{1m}) \\ (y_{21} - \hat{y}_{21}) & (y_{22} - \hat{y}_{22}) & \cdots & (y_{2m} - \hat{y}_{2m}) \\ \vdots & \vdots & \cdots & \vdots \\ (y_{N1} - \hat{y}_{N1}) & (y_{N2} - \hat{y}_{N2}) & \cdots & (y_{Nm} - \hat{y}_{Nm}) \end{bmatrix} \tag{2.9}$$

## 2.2 Exemplo: Aproximação Polinomial em Camadas

Exemplos com polinômios são utilizados em vários momentos ao longo deste texto como exemplo de várias características comuns com RNAs e outros modelos de aprendizado. Nesta seção, um polinômio de grau 2 será utilizado como exemplo do mapeamento em camadas.

Considere um polinômio de grau $p$ conforme representado na sua forma geral na Equação 2.10.

$$p(x) = w_p x^p + w_{p-1} x^{p-1} + \cdots + w_1 x + w_0 \tag{2.10}$$

em que $x$ é o argumento e $w_i$ é o coeficiente do termo de grau $i$.

Dadas as observações $(x_i, y_i)$ representadas na forma do conjunto de dados $\tau = \{x_i, y_i\}_{i=1}^{N}$, deseja-se encontrar o polinômio de grau $p$ que melhor aproxime a função geradora $f_g(x)$ do conjunto $\tau$. O objetivo é, a partir das amostras de dados, encontrar o grau $p$ e os coeficientes $w_i$ de forma tal que $p(x) \approx f_g(x)\ \forall x$. A aproximação de $f_g(x)$ é usualmente feita com base na minimização do erro dos termos quadráticos $(y_i - p(x_i))^2$ $(i = 1 \cdots N)$. Espera-se que o conjunto $\tau$ contenha informação suficiente para que seja possível aproximar $f_g(x)$ por $p(x)$ com base somente nas suas $N$ amostras. Os parâmetros de $p(x)$ são ajustados de forma tal que $y_i = w_p x_i^p + w_{p-1} x_i^{p-1} + \cdots + w_1 x_i + w_0\ \forall x_i \in D$, conforme representado no sistema de equações 2.11.

$$\begin{array}{rrrrrr} y_1 = w_p x_1^p & +w_{p-1}x_1^{p-1} & + & \cdots & +w_1 x_1 & +w_0 \\ y_2 = w_p x_2^p & +w_{p-1}x_2^{p-1} & + & \cdots & +w_1 x_2 & +w_0 \\ \vdots & \vdots & \cdots & \vdots & & \\ y_N = w_p x_N^p & +w_{p-1}x_N^{p-1} & + & \cdots & +w_1 x_N & +w_0 \end{array} \tag{2.11}$$

O sistema representado em 2.11 possui $N$ equações e $p$ incógnitas, podendo também ser representado na forma matricial 2.12.

$$\mathbf{Hw} = \mathbf{y} \tag{2.12}$$

em que $\mathbf{H}$, $\mathbf{w}$ e $\mathbf{y}$ são representados em 2.13, 2.14 e 2.15.

$$\mathbf{H} = \begin{bmatrix} x_1^p & x_1^{p-1} & \cdots & x_1 & 1 \\ x_2^p & x_2^{p-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & \cdots & \vdots & \\ x_N^p & x_N^{p-1} & \cdots & x_N & 1 \end{bmatrix} \tag{2.13}$$

$$\mathbf{w} = \begin{bmatrix} w_p \\ w_{p-1} \\ \vdots \\ w_1 \\ w_0 \end{bmatrix} \tag{2.14}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{2.15}$$

A matriz $\mathbf{H}$ possui um papel importante na resolução do problema de aproximação, pois ela contém os termos não-lineares que compõem o polinômio $p(x)$, os quais serão responsáveis pela projeção dos elementos $x_i$ no espaço composto pelo sistema de coordenadas caracterizado pelas colunas de $\mathbf{H}$. Como $\mathbf{H}$ e $\mathbf{y}$ são dados pelo problema, a solução da Equação 2.12 pode ser obtida por meio da pseudoinversa, conforme Equação 2.16.

$$\mathbf{w} = \mathbf{H}^{+}\mathbf{y} \tag{2.16}$$

em que $\mathbf{H}^{+}$ é a pseudoinversa de $\mathbf{H}$.

### 2.2.1 Exemplo numérico

O exemplo a seguir mostra os passos para construção de uma aproximação polinomial utilizando a estrutura de dados apresentada nas seções anteriores. Os dados para o exemplo serão amostrados da função geradora $f_g(x) = \frac{1}{2}x^2 + 3x + 10$, aos quais serão adicionados um ruído gaussiano em torno de $f_g(x)$. Em outras palavras, para cada valor $x_i$ de entrada será associado um valor de saída $y_i = f_g(x_i) + \epsilon_i$ em que $\epsilon_i$ é amostrado de uma distribuição Normal com média 0 e desvio padrão unitário. O trecho de código a seguir mostra o passo-a-passo para a aproximação. Nas linhas iniciais é feita a inicialização das

variáveis, carregamento do pacote **corpcor** que contém a função para o cálculo da pseudoinversa conforme Equação 2.16, assim como a definição da função **fgx** que retorna o valor de $f_g(x)$. A seguir, os valores de $x$ são amostrados por meio do comando **x<-runif(n = N,min=-15,max=10)** que resulta em 20 amostras uniformemente distribuídas entre $-15$ e $+10$.

```
> rm(list=ls())
> library('corpcor') #  Pacote para o cálculo de pseudoinversa
> fgx<-function(xin) 0.5*xin^2+3*xin+10 # Função fg(x)
> X<-runif(n = 20,min=-15,max=10) # Amostra x
> Y<-fgx(X) + 10*rnorm(length(X))
```

Uma vez amostrados os valores de $x$ e $y$, conforme código anterior, é necessário escolher o grau do polinômio que será utilizado na aproximação. Para este exemplo será utilizado um polinômio de grau 2, mesmo grau da função geradora. A influência do grau do polinômio na qualidade da aproximação e como ele se relaciona com a complexidade das máquinas de aprendizado não será discutida neste momento, já que o nosso objetivo agora é apenas dar um exemplo de manipulação da estrutura de dados. Assim, para um polinômio de grau 2 a estrutura em camadas correspondente é apresentada na Figura 2.3.

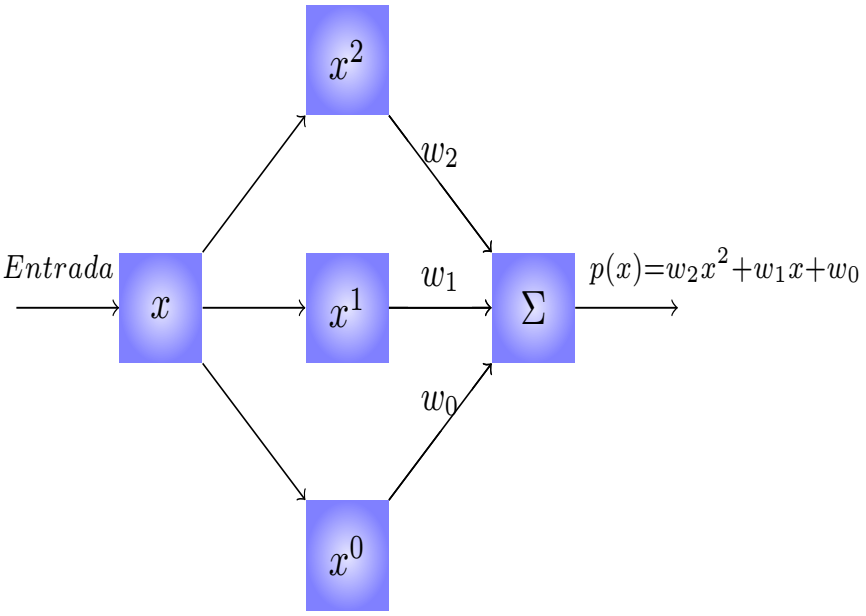



Figura 2.3: Polinômio $p(x) = w_2x^2 + w_1x^1 + w_0x^0$ para o exemplo de projeções em camadas.

Conforme código anterior, a matriz $\mathbf{X}$ deste exemplo, correspondente à Equação 2.1 tem dimensões $20 \times 1$, ou seja, é na verdade um vetor contendo todas as amostras de entrada. De maneira análoga, a matriz de saída $\mathbf{Y}$, correspondente à Equação 2.2, possui as mesmas dimensões $20 \times 1$. A matriz $\mathbf{H}$, Equação 2.7, que corresponde à projeção de $\mathbf{X}$ na camada intermediária, conforme Figura 2.3, é obtida por meio da aplicação dos termos do polinômio à matriz $\mathbf{X}$. No código que segue esta projeção é realizada por meio da linha

de código `H<-cbind(X^2,X,1)` em que o termo 1 corresponde à entrada $x^0$. O resultado é uma matriz de dimensões $20 \times 3$ em que cada linha corresponde à projeção da linha correspondente de $\mathbf{X}$ em $\mathbf{H}$. Uma vez obtida a projeção $\mathbf{H}$, o vetor de pesos $\mathbf{w}$ pode ser obtido por meio da operação $\mathbf{w} = \mathbf{H}^{+}\mathbf{y}$ apresentada na Equação 2.16 e implementada no trecho de código a seguir por meio da expressão `w<-pseudoinverse(H)`%*% Y . Os elementos de $\mathbf{w}$ correspondem à solução final dos coeficientes do polinômio aproximador.

```
> # Aproximação de grau dois
> H<-cbind(X^2,X,1)
> w<-pseudoinverse(H) %*% Y
```

A qualidade da aproximação dos coeficientes do polinômio em relação aos valores $[0.5, 3, 10]$ da função geradora pode ser verificada comparando-os individualmente, como mostrado a seguir. A primeira coluna da impressão que se segue contém os valores estimados e a segunda coluna os valores dos coeficientes da função geradora. Pode ser observado que os valores estimados são bem próximos daqueles da função original. O desvio que ocorre entre eles é devido a vários fatores, entre eles o tamanho da amostra e o ruído gaussiano adicionado a cada amostra.

```
           [,1] [,2]
[1,]  0.5170383  0.5
[2,]  3.5358834  3.0
[3,] 12.2495178 10.0
```

Uma vez obtidos os coeficientes da aproximação, o modelo induzido, representado de forma genérica como $f(\mathbf{x}, \mathbf{w})$ pode ser então utilizado para estimar a função geradora $f_g(\mathbf{x})$ em outros pontos, diferentes daqueles utilizados para induzí-la. Com o objetivo de comparar o resultado da aproximação, já que este se trata de um problema univariado, cuja resposta pode ser visualizada, as respostas de $f_g(\mathbf{x})$ e de $f(\mathbf{x}, \mathbf{w})$ no intervalo $[-15, 10]$ serão apresentadas na forma gráfica a seguir. Para que as funções sejam avaliadas em pontos não utilizados anteriormente, será utilizado um intervalo de 0.1 para a variável *xgrid* a seguir. A variável *ygrid*, que é obtida basicamente pela aplicação de *xgrid* à função geradora, corresponde aos valores aproximados de $f_g(\mathbf{x})$ no intervalo.

```
> xgrid<-seq(-15,10,0.1)
> ygrid<-(0.5*xgrid^2+3*xgrid+10)
```

Para que $f(\mathbf{x}, \mathbf{w})$ seja estimada nos valores contidos em *xgrid*, é preciso inicialmente projetar *xgrid* na camada intermediária por meio do comando `Hgrid<-cbind(xgrid^2,xgrid,1)` e então *yhatgrid* é obtida utilizando-se para isto os coeficientes $\mathbf{w}$ estimados anteriormente. O resultado da aproximação é apresentado na Figura 2.4. Como pode ser observado, apesar do pequeno desvio nos valores dos coeficientes, a aproximação resultante é bastante próxima da função geradora dos dados.

```
> Hgrid<-cbind(xgrid^2,xgrid,1)
> yhatgrid<-Hgrid %*% w
```

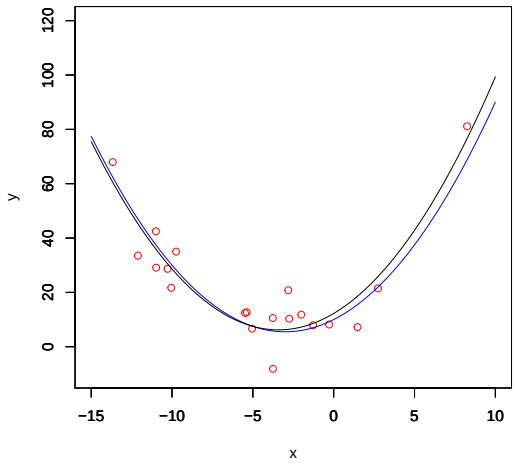



Figura 2.4: Aproximação de uma função por meio de um polinômio de grau 2. Os dados foram gerados à partir da função $f_g(x) = \frac{1}{2}x^2 + 3x + 10$, os quais são mostrados na cor vermelha na figura. A linha contínua em preto mostra a função aproximada e em azul a função geradora.

# Capítulo 3

# Redes Neurais Lineares

## 3.1 Introdução

Nos capítulos anteriores foi feita uma analogia entre as aproximações polinomiais e outros modelos não lineares baseados no princípio das projeções em camadas, como as RNAs. De uma maneira geral, um polinômio pode ser descrito por meio da equação $\mathbf{Hw} = \hat{\mathbf{Y}}$ em que $\mathbf{H}$ corresponde à matriz de entrada $\mathbf{X}$ transformada não-linearmente pelos termos do polinômio, $\hat{\mathbf{Y}}$ é a matriz que contém os valores previstos de saída e $\mathbf{w}$ o vetor que contém os coeficientes do polinômio. Como o grau do polinômio é estabelecido previamente e determina diretamente a projeção de $\mathbf{X}$ em $\mathbf{H}$, o problema de aproximação polinomial envolve a resolução do sistema de equações lineares $\mathbf{Hw} = \mathbf{Y}$, em que $\mathbf{Y}$ corresponde aos valores amostrados de saída. A solução deste sistema linear pode ser obtida de várias formas, entre elas utilizando-se o método dos mínimos quadrados, conforme discutido no capítulo anterior.

Serão apresentados neste capítulo modelos que se assemelham à aproximação polinomial, porém, sem a transformação intermediária não-linear de $\mathbf{X}$ em $\mathbf{H}$. Em outras palavras, os modelos abordados neste capítulo serão caracterizados pela equação $\mathbf{Xw} = \mathbf{Y}$ ou, de maneira mais geral $f(\mathbf{Xw}) = \mathbf{Y}$ em que $f(\cdot)$ é a função de ativação do modelo. A função de ativação pode assumir várias formas, entre elas a função identidade $f(u) = u$, a função degrau apresentada nos capítulos anteriores e a função logística, que é utilizada como aproximação contínua da função degrau. Para o caso particular em que $f(u) = u$ o modelo resultante é linear, sendo o mesmo caracterizado pelo sistema linear $\mathbf{Xw} = \mathbf{Y}$. Em capítulos posteriores serão estudados outros modelos de aproximação cujas funções de ativação são não-lineares.

Independemente do tipo de função de ativação, estes modelos são de uma única camada, já que as amostras de entrada, as linhas de $\mathbf{X}$, são multiplicadas diretamente pelo vetor $\mathbf{w}$, resultando nos elementos de $\mathbf{Y}$. As Figuras 3.1 e 3.2 mostram um exemplo de como um modelo de camada simples pode ser representado de forma geral pela equação $f(\mathbf{Xw}) = \mathbf{Y}$. Na Figura 3.1, a primeira linha da matriz $\mathbf{X}$ é multiplicada pelo vetor $\mathbf{w}$, resultando no primeiro elemento do vetor de saída $\mathbf{y}$. Neste exemplo, a função de ativação é linear, ou seja, $f(u) = u$. Esta operação é representada na estrutura de um modelo de uma única camada apresentado na Figura 3.2. Caso o modelo fosse não linear com

$$\underbrace{\begin{pmatrix} 0.3 & 0.7 & 0.5 \\ -1.2 & 0.5 & 3 \\ 0.4 & -1.7 & -2.1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} 3.0 \\ 2.5 \\ -0.7 \end{pmatrix}}_{\mathbf{w}} = \underbrace{\begin{pmatrix} 2.3 \\ -4.45 \\ -1.58 \end{pmatrix}}_{\mathbf{y}}$$

Figura 3.1: Exemplo de modelo de uma única camada representado por meio da operação matricial $f(\mathbf{Xw}) = \mathbf{u})$ em que $y = f(u) = u$, ou seja, $\mathbf{Xw} = \mathbf{Y}$.

a função de ativação degrau com limiar em zero, por exemplo, o vetor de saída seria igual a $[1, 0, 0]^T$.

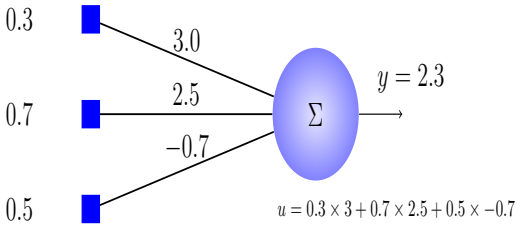



Figura 3.2: Representação da operação matricial apresentada na Figura 3.1 na forma de um modelo linear de uma única camada. Para este exemplo, $y = u$.

Assim, à partir da matriz $\mathbf{X}$ e do vetor $\mathbf{y}$, representadas na forma do conjundo de dados $\tau = \{\mathbf{x}_i, y_i\}_{i=1}^N$, o objetivo do aprendizado de modelos de camada única é encontrar o vetor de parâmetros $\mathbf{w}$, ou a matriz $\mathbf{W}$ no caso de múltiplas saídas, capaz de aproximar $f(\mathbf{x}, \mathbf{w}) \approx f_g(\mathbf{x}_t)$ também para vetores quaisquer $\mathbf{x}_t \notin \tau$. Em outras palavras, espera-se que a função $f(\mathbf{x}_t, \mathbf{w})$ seja uma boa aproximação de $f_g(\mathbf{x}_t)$, não restrita ao conjunto $\tau$. Nas seções seguintes serão apresentadas diferentes abordagens para estimar o vetor $\mathbf{w}$.

## 3.2 Aprendizado Hebbiano

De acordo com a Teoria de Hebb [Heb49], a eficiência de uma determinada sinapse que conecta os neurônios pré-sináptico A e pós-sináptico B deve ser reforçada quando o neurônio A toma parte de maneira persistente e frequente na ativação do neurônio B. Esta teoria para explicar a plasticidade sináptica é traduzida em linguagem matemática para o aprendizado de redes neurais através da *Regra de Hebb*, conforme Equação 3.1.

$$w_{ij} \propto x_{ki} y_{kj} \tag{3.1}$$

onde $w_{ij}$ é o peso (sináptico) que conecta o elemento $i$ do vetor de entrada $\mathbf{x}_k$ ao elemento $j$ do vetor de saída $\mathbf{y}_k$. Os vetores $\mathbf{x}_k$ e $\mathbf{y}_k$ correspondem às linhas $k$ das matrizes $\mathbf{X}$ e $\mathbf{Y}$, respectivamente.

Uma representação esquemática da estrutura de rede linear de uma única camada que executa a operação $\mathbf{XW} = \mathbf{Y}$ é apresentada na Figura 3.3. A estrutura de rede genérica da figura possui $n$ entradas e $m$ saídas e é um caso geral da estrutura de uma única saída da Figura 3.2. O princípio do aprendizado

Hebbiano visa, então, a reforçar as conexões relevantes para a ativação de uma determinada saída da rede.

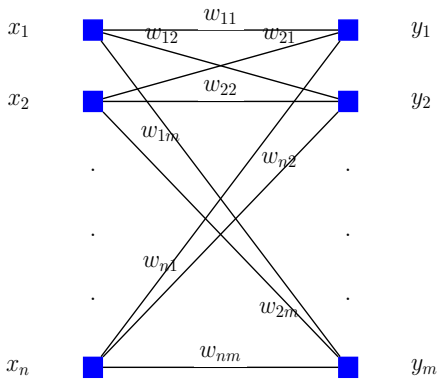



Figura 3.3: Estrutura de rede uma única camada, representando a operação $\mathbf{XW} = \mathbf{Y}$.

Do ponto de vista do aprendizado de RNAs, a Regra de Hebb pode ser interpretada como: *o ajuste de um determinado parâmetro $w$, correspondente ao peso de uma RNA, deve ser proporcional ao produto dos elementos correspondentes dos vetores anteriores e posteriores conectados a este peso.* Assim, uma possível solução para o vetor de pesos da equação $\mathbf{XW} = \mathbf{Y}$ é a aplicação direta da Regra de Hebb como critério de ajuste dos pesos, conforme Equação 3.2.

$$\hat{\mathbf{W}} = \mathbf{X}^T\mathbf{Y} \tag{3.2}$$

Assim, a aplicação da Equação 3.2 diretamente à equação $\mathbf{XW} = \hat{\mathbf{Y}}$, em que $\hat{\mathbf{Y}}$ é o valor estimado de $\mathbf{Y}$, resulta na Equação 3.3.

$$\mathbf{X}\overbrace{\mathbf{X}^T\mathbf{Y}}^{\hat{\mathbf{W}}} = \hat{\mathbf{Y}} \tag{3.3}$$

Para que o erro de aproximação seja nulo, ou seja, $\mathbf{XX}^T\mathbf{Y} - \hat{\mathbf{Y}} = 0$, é preciso que $\mathbf{XX}^T = I$, ou seja, que os vetores de entrada sejam ortogonais entre si e que suas normas sejam unitárias. Assim, para que esta condição seja satisfeia, devemos ter:

$$\begin{array}{lll} \text{Condição 1:} & \mathbf{x}_i \perp \mathbf{x}_j & \forall i \neq j \\ \text{Condição 2:} & \mathbf{x}_i\mathbf{x}_i^T = 1 & \forall i \end{array} \tag{3.4}$$

A Condição 1 depende do conjunto de dados amostrados, ou seja, é considerada inerente à amostragem, enquanto que a Condição 2 pode ser satisfeita através da normalização dos dados de entrada. Assim, o desempenho dos modelos baseados na Regra de Hebb sempre dependerá do termo residual proporcional à matriz $\mathbf{K} = \mathbf{XX}^T$, em que $\mathbf{K}$ é a matriz de autocorrelação [OS75] de dimensões $N \times N$. Quanto mais próxima da identidade esta matriz for, melhor será a aproximação.

Ao longo deste texto em vários momentos a matriz $\mathbf{K}$ assumirá um papel importante por ser análoga à Matriz de Kernel Linear [CST00] e por representar

também as afinidades entre os padrões de entrada. As afinidades, representadas aqui através de medidas de correlação, têm um papel importante na construção de modelos baseados em kernels, especialmente aqueles envolvendo tarefas de classificação ou agrupamento de dados. Em modelos como as Máquinas de Vetores de Suporte (SVM - *Support Vector Machines*) [BGV92, Vap95, CV95] a transformação para o novo espaço, chamado de Espaço de Características, é realizada através de matrizes de kernel análogas à matriz **K** descrita acima.

### 3.2.1 Exemplo de Aproximação Linear Hebbiana

Um exemplo simples de aproximação linear baseado em aprendizado Hebbiano é apresentado a seguir. Considere que o sistema a ser modelado possa ser descrito através da expressão $y = -2x$, ou seja, todo sinal de entrada $x$ será invertido e multiplicado por 2. Obviamente, esta função que representa o comportamento do sistema não é conhecida previamente. Com o objetivo de obter um modelo para este sistema, um sinal de entrada aplicado à sua entrada é amostrado em intervalos regulares de $\frac{\pi}{2}$. Para se obter as saídas correspondentes para cada um dos valores de entrada, a saída do sistema é também amostrada nos mesmos intervalos de tempo. Os sinais de entrada e saída obtidos correspondem aos vetores **x** e **y** do código que se segue.

```
> rm(list=ls())      # Inicializa todas as variáveis
> t<-seq(0,2*pi,0.5*pi)
> x<-as.matrix(sin(t))
> y<-as.matrix(-2*x)
```

Como deseja-se obter um modelo linear para o sistema, a entrada $x$ deve ser aumentada em uma dimensão para que, ao ser multiplicada por **w**, resulte na expressão geral de uma reta na forma $y = ax + b$. Esta composição do vetor de entrada é feita por meio do comando $x <- cbind(1, x)$ no trecho de código que se segue. Em seguida, faz-se a normalização das entradas e obtém-se o vetor de pesos de acordo com a Equação 3.2.

```
> x<-cbind(x,1)
> dK<-diag(x %*% t(x))
> x[,1]<-x[,1] / dK
> x[,2]<-x[,2] / dK
> w<-t(x) %*% y
```

Uma vez obtido o vetor de pesos que caracteriza o modelo, deseja-se, então, verificar a qualidade do modelo obtido. No nosso caso, como geramos os sinais de entrada e saída sinteticamente e como conhecemos a expressão do sistema ($y = -2x$), podemos avaliar inicialmente os valores obtidos para o vetor de pesos como se segue.

```
> w

              [,1]
[1,] -2.000000e+00
[2,]  2.678141e-16
```

Os valores obtidos para os pesos do modelo foram $w(1) = -2$ e $w(2) = 2.678141e{-}16$, o que resulta em um modelo com função $\hat{y} = -2x + 2.678141e{-}16$ praticamente equivalente à função $y = -2x$ executada pelo sistema. Neste caso, por ser um problema linear simples, foi possível comparar diretamente os parâmetros obtidos com os parâmetros do sistema, no entanto, isto normalmente não é possível, já que a função a ser modelada não é conhecida de antemão.

### 3.2.2 Interferência Cruzada

Conforme Equação 3.3, a qualidade da aproximação utilizando-se a Regra de Hebb [Heb49] dependerá da forma como os vetores de entrada $D_u = \{\mathbf{x}_i\}_{i=1}^{N}$ se relacionam espacialmente, resultando em recuperação ótima quando estes são ortogonais entre si. O comportamento do modelo Hebbiano pode ser melhor avaliado analisando-se a resposta a um vetor de entrada $\mathbf{x}_i$ arbitrário. De forma geral, a resposta $\hat{y}_i$ ao vetor de entrada $\mathbf{x}_i$ pode ser expressa conforme a Equação 3.5.

$$\sum_{j=1}^{N} \mathbf{x}_i \mathbf{x}_j^T y_j = \hat{y}_i \tag{3.5}$$

Retirando-se o termo $\mathbf{x}_i \mathbf{x}_i^T y_i$ do somatório chega-se à Equação 3.6.

$$\sum_{j=1, j\neq i}^{N} \mathbf{x}_i \mathbf{x}_j^T y_j + \overbrace{\mathbf{x}_i \mathbf{x}_i^T}^{||\mathbf{x}_i||=1} y_i = \hat{y}_i \tag{3.6}$$

Considerando-se que os vetores de entrada sejam normalizados, o que pode ser garantido antes do treinamento, chega-se então à Equação 3.7, onde pode ser visto que o termo de interferência cruzada é adicionado ao valor $y_i$ que se deseja recuperar. Pode-se dizer que $y_i$ é distorcido devido à interferência cruzada, ou não-neutralidade, entre os vetores de entrada.

$$\underbrace{\sum_{j=1, j\neq i}^{N} \mathbf{x}_i \mathbf{x}_j^T y_j}_{\text{Interferência Cruzada}} + y_i = \hat{y}_i \tag{3.7}$$

Uma forma de reduzir a interferência é buscar uma representação ortogonal para as amostras de entrada ou fazer uma projeção em um espaço intermediário de tal forma que a ortogonalidade neste novo espaço seja garantida. Nesta abordagem, a matriz $\mathbf{X}$ é transformada em uma nova matrix $\mathbf{H}(\mathbf{X})$ em que a minimização da interferência cruzada seja garantida. Assim, neste caso, considera-se a projeção $\mathbf{H}(\mathbf{X})$, usualmente não-linear, em um novo espaço e não mais a matriz de entrada $\mathbf{X}$. Não obstante, não trataremos neste ponto de métodos de ortogonalização ou de mapeamentos não-lineares sucessivos, mas é importante enfatizar aqui que a solução para o problema de minimização da interferência envolve a busca de uma nova representação espacial para os dados de entrada e que a função $\mathbf{H}(\mathbf{X})$ representaria a projeção em um novo espaço.

Pode-se observar que o tamanho $N$ do conjunto de dados também influencia na interferência cruzada, já que quanto maior o número de termos do somatório maior a probabilidade de a sua magnitude aumentar. É claro que este aumento

dependerá também dos produtos individuais entre os vetores, mas quanto maior o valor de $N$ mais termos existirão no somatório, podendo resultar no aumento da interferência. Segundo este princípio, mesmo havendo interferência, é possível armazenar um número limitado de padrões de tal forma que o somatório não cause um deslocamento relevante no valor de $y_i$. Por esta razão, as memórias associativas que se baseiam na Regra de Hebb têm a sua capacidade de armazenamento de pares de entrada e saída limitada. Os estudos sobre a capacidade de armazenamento deste tipo de memória, especialmente as Redes de Hopfield [Hop82], envolvem alguma estimativa da distribuição dos dados de entrada. No caso mais simples, considera-se que a distribuição dos valores de entrada seja uniforme o que resulta em uma capacidade de armazenamento limitada e dependente da dimensão da rede [MPRV87, Bra94].

## 3.3 Mínimos Quadrados

A solução da equação geral $\mathbf{XW} = \mathbf{Y}$ pode ser obtida diretamente por meio da inversa de $\mathbf{X}$ ou, na forma mais geral, através da sua pseudoinversa $\mathbf{X}^+$, conforme representado na Equação 3.8. É interessante observar também que a solução para $\mathbf{W}$ representada pela Equação 3.8 é análoga à solução Hebbiana da Equação 3.2. Na verdade, as duas soluções são equivalentes na situação em que $\mathbf{X}$ forma uma base ortonormal, já que neste caso as matrizes inversa e transposta são iguais [Leo94].

$$\mathbf{W} = \mathbf{X}^+\mathbf{Y} \tag{3.8}$$

onde $\mathbf{X}^+$ é a pseudoinversa de $\mathbf{X}$ [Leo94].

A solução pela pseudoinversa não sofre de interferência cruzada como a solução Hebbiana, já que a mesma resulta na solução por mínimos quadrados [NW06b], que minimiza o erro $||\mathbf{Y} - \mathbf{XW}||^2$. A solução pela pseudoinversa é tal que equivale à solução Hebbiana com os vetores de entrada normalizados e ortogonais entre si.

## 3.4 Adaline

Conforme descrito nas seções anteriores, um neurônio MCP [MP43] é caracterizado pela aplicação de uma função de limiar $f(\cdot)$ à soma ponderada das entradas $u = \sum w_i x_i$. De uma maneira geral, a saída $\hat{y}$ de um neurônio MCP pode ser descrita por meio da expressão $\hat{y} = f(\mathbf{x}, \mathbf{w})$, em que $\mathbf{x}$ é o vetor de entrada, $\mathbf{w}$ o vetor de parâmetros[1] e $f(\cdot)$ é a função de ativação do neurônio. A função de ativação de limiar, ou degrau, caracteriza o neurônio MCP, no entanto, dependendo da aplicação, outras funções de ativação podem ser utilizadas, como funções sigmoidais ou lineares. O modelo Adaline [WH60] é caracterizado pela utilização da função identidade $f(u) = u$ como função de ativação, assim, a sua saída corresponde exatamente à soma ponderada das entradas, ou seja, $\hat{y} = \sum w_i x_i$.

Nas seções anteriores foram apresentadas duas abordagens para a solução do problema de treinamento de modelos lineares de uma única camada, como

[1] Como o Perceptron e o Adaline possuem somente uma camada, temos $\theta = \mathbf{w}$ para este caso particular de redes de camada única.

o Adaline. A primeira delas foi a Regra de Hebb [Heb49] em que o vetor de pesos pode se encontrado diretamente por meio do produto cruzado entre as matrizes $\mathbf{X}$ e $\mathbf{Y}$, ou seja, $\mathbf{w} = \mathbf{X}^T\mathbf{Y}$. Conforme discutido anteriormente, a qualidade da solução pela Regra de Hebb dependerá da interferência cruzada entre os vetores $\mathbf{x}_i$ que compõem as linhas de $\mathbf{X}$. A outra forma de solução do problema é por meio da inversa de $\mathbf{X}$, ou pseudoinversa no caso geral, que resulta em $\mathbf{w} = \mathbf{X}^+\mathbf{Y}$, em que $\mathbf{X}^+$ é a pseudoinversa de $\mathbf{X}$. Esta abordagem equivale à solução direta do problema de mínimos quadrados que minimiza a função de custo $J = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ [NW06b]. As soluções por meio destas duas abordagens são obtidas diretamente, em um único passo. O treinamento do Adaline, no entanto, é feito por meio de uma abordagem iterativa, utilizando-se para isto o gradiente descendente, descrito a seguir.

### 3.4.1 Função de Custo Quadrática

A utilização de uma função de custo quadrática, como a apresentada na Equação 3.9, é bastante conveniente, já que a mesma é quadrática nos parâmetros $w_j$, o que facilita a busca por seu mínimo global.

$$J = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.9}$$

em que $N$ é o número de amostras, $y_i$ é a saída desejada para o vetor de entrada $\mathbf{x}_i$ e $\hat{y}_i$ é a saída estimada pelo modelo para a mesma entrada $\mathbf{x}_i$.

Como $\hat{y}_i = \sum_{j=1}^{n} w_j x_{ij}$, em que $n$ é a dimensão de entrada e $x_{ij}$ é o elemento $j$ do vetor $\mathbf{x}_i$, a minimização de $J$ é um problema quadrático nos pesos $w_j$, já que a função de custo pode ser reescrita como $J = \sum_{i=1}^{N}(y_i - \sum_{j=1}^{n} w_j x_{ij})^2$. Considere, por exemplo, um problema simples de uma única variável, em que o modelo Adaline seja caracterizado por $\hat{y}_i = w_1 x_i + w_0$. Para este modelo, a função de custo $J$ da Equação 3.9 pode ser escrita como $J = \sum_{i=1}^{N}(y_i - (w_1 x_i + w_0))^2$, que é quadrática em $w_1$ e $w_0$, que são os parâmetros que definem a função. A Figura 3.4 mostra a função de custo da Equação 3.9 para um problema de duas variáveis em que $w_1 = 2$ e $w_0 = 2$, ou seja, $\hat{y}_i = 2x_i + 2$. A função $J$ foi calculada na figura para $\mathbf{x} = [0.1, 2, -1, 0.2]^T$ e $\mathbf{y} = [2.2, 6, 0, 2.4]^T$, considerando-se valores dos parâmetros $w_1 = 2$ e $w_0 = 2$. Como pode ser observado, a função $J$ possui a forma quadrática e um mínimo global no ponto $(2, 2)$. O ajuste dos pesos para atingir esta solução é feito no Adaline de forma iterativa por meio do gradiente descendente, como será descrito a seguir.

### 3.4.2 Regra Delta

Considere a função de custo da Equação 3.9 em que $\hat{y}_i = \sum_{j=1}^{n} w_j x_{ij}$, ou seja, $f(u) = u$. Deseja-se obter a direção de ajuste $\Delta\mathbf{w}$ do vetor de pesos $\mathbf{w}$ por meio do gradiente descendente, ou seja, $\Delta\mathbf{w}$ deve ser aplicado em direção contrária ao gradiente em que cada ponto da superfície $J$. Em outras palavras, para cada par $(w_1, w_0)$, um valor de $J$ é calculado e para cada valor de $J$ os valores de $w_1$ e $w_0$ são ajustados em sentido contrário ao gradiente no ponto $(w_1, w_0)$. Assim, o ajuste $\Delta\mathbf{w}_j$ visando à minimização da função de custo da Equação 3.9 pelo método do gradiente descendente deve ser tal que $\Delta\mathbf{w}_j \propto -\frac{\partial J}{\partial w_j}$. Considerando-se que o ajuste será realizado para cada padrão $i$ individualmente, somente o

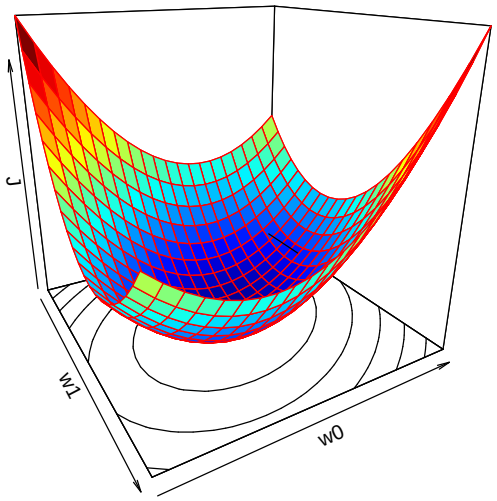



Figura 3.4: Superfície de erro quadrática correspondente à função de custo da Equação 3.9 para $\hat{y}_i = 2x_i + 2$.

termo $i$ da Equação 3.9 importa para o ajuste, conforme Equação 3.10, o que leva à Equação 3.11.

$$\frac{\partial J}{\partial w_j} = \frac{1}{2}\frac{\partial J}{\partial w_j}\left(y_i - \sum_{k=1}^{n} w_k x_{ik}\right)^2 \tag{3.10}$$

$$\frac{\partial J}{\partial w_j} = \frac{1}{2}\cdot 2\cdot\left(y_i - \sum_{k=1}^{n} w_k x_{ik}\right)(-1)\cdot\frac{\partial}{\partial w_j}\left(\sum_{k=1}^{n} w_k x_{ik}\right) \tag{3.11}$$

Como todos os termos de $\frac{\partial}{\partial w_j}(\sum_{k=1}^{n} w_k x_{ik})$ são nulos, exceto para $k = j$, então, a Equação 3.10 pode ser escrita na forma da Equação 3.12.

$$\frac{\partial J}{\partial w_j} = -\overbrace{\left(y_i - \sum_{k=1}^{n} w_k x_{ik}\right)}^{erro} x_{ij} \tag{3.12}$$

Assim, a expressão final para o ajuste do peso $w_j$ para um vetor de entrada $\mathbf{x}_i$ pelo método do gradiente descendente deve ser $\Delta\mathbf{w}_j = \eta e_i x_{ij}$, em que a constante $\eta$ foi adicionada para substituir a proporcionalidade de $\Delta\mathbf{w}_j \propto -\frac{\partial J}{\partial w_j}$. Aplicando-se esta expressão de ajuste a todos os elementos do vetor $\mathbf{w}$, a expres-

são final para a atualização de $\mathbf{w}$ pelo gradiente descendente para um modelo do tipo Adaline é apresentada na Equação 3.13.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta e_i \mathbf{x}_i \tag{3.13}$$

em que $\mathbf{w}_i(t)$ é o vetor de pesos atual, $\mathbf{w}_i(t+1)$ é o vetor de pesos no instante seguinte após atualização, $\eta$ é o passo do ajuste, $\mathbf{x}_i$ é o vetor de entrada e $e_i$ é o erro para $\mathbf{x}_i$, ou seja, $e_i = (y_i - \hat{y}_i)$.

### 3.4.3 Algoritmo de treinamendo do Adaline

Antes de apresentar o pseudo-código do algoritmo de treinamento do Adaline, considere como exemplo o ajuste dos pesos de um neurônio linear simples com três entradas $x_1$, $x_2$ e $x_3$. Os pares de entrada e saída de treinamento são $(\mathbf{x}_1, y_1) = ([0.4, 1.2, 1]^T, 4.64)$ e $(\mathbf{x}_2, y_2) = ([1.35, 0.7, 1]^T, 2.26)$ e o vetor de pesos atual $\mathbf{w} = [-0.21, 3.35, 0.5]^T$. Considerando-se $\eta = 0.1$, a atualização do vetor de pesos para o par $(\mathbf{x}_1, y_1)$ segundo a Equação 3.13 é apresentada na Equação 3.14. Como pode ser observado, o ajuste resultou em uma pequena alteração na magnitude dos pesos, tendo havido troca de sinal nos pesos $w_2$ e $w_3$. O treinamento deste modelo ocorre, então, de maneira iterativa, sendo o vetor de pesos ajustado alternadamente para os pares $(\mathbf{x}_1, y_1)$ e $(\mathbf{x}_2, y_2)$ até que a soma dos erros quadráticos para estes dois pares atinja o limiar de tolerância pré-definido.

$$\begin{pmatrix} -0.20184 \\ 3.37448 \\ 0.52040 \end{pmatrix} = \begin{pmatrix} -0.21 \\ 3.35 \\ 0.5 \end{pmatrix} + 0.1(4.640 - 4.436) \begin{pmatrix} 0.4 \\ 1.2 \\ 1 \end{pmatrix} \tag{3.14}$$

Assim, o treinamento do modelo Adaline pelo método do gradiente descendente envolve basicamente a aplicação da Equação 3.13 a todos os dados do conjunto de treinamento até que o erro atinja um limiar de tolerância previamente definido. O pseudo-código apresentado no Algoritmo 1 apresenta os passos para a implementação do algoritmo de treinamento do Adaline.

### 3.4.4 Implementação do treinamento do Adaline

Uma possível implementação do algoritmo de treinamento do Adaline, conforme pseudo-código do Algoritmo 1 é apresentado na Listagem 3.1. A função de treinamento **trainadaline()** recebe como entrada os dados e os parâmetros de treinamento e como saída fornece os pesos obtidos e o erro à cada época. O parâmetro de entrada **par**, quando em 1, indica que a entrada adicional +1 não foi acrescida aos dados e que ela deve ser adicionada dentro da função. A inicialização dos pesos então ocorre por meio de amostragem uniforme na faixa de valores $-0.5$ a $+0.5$ visando a dar chances iguais a pesos positivos e negativos durante o treinamento. Outra parte importante do código é a realização de sequência aleatória de treinamento por meio dos comandos `xseq<-sample(N)` e `irand<-xseq[i]` que visam a evitar sequências determinísticas de ajuste dos pesos. O restante do código envolve basicamente a aplicação da equação de ajuste dos pesos e o acúmulo do erro a cada iteração.

**Algoritmo 1** Algoritmo de treinamento do Adaline.

**função** TRAINADALINE
  **Entrada:** X,y,$\eta$,tol,maxepocas
  **Saída:** **w**,erro por época
  $N \leftarrow$ Número de linhas de **X**
  Inicializa vetor de pesos **w**
  **enquanto** (erroepoca > tol) **e** (nepocas < maxepocas) **faça**
    vetor xseq = permutação de 1 a N
    **para** i $\leftarrow$ 1 **até** N **faça**
      iseq = xseq[i]
      $erro$ = (y[iseq] - $\mathbf{w}^T\mathbf{x}[iseq,]$)
      **w** = **w** + $\eta$ erro **x[iseq,]**
      ei2 = ei2 + $erro^2$
    **fim para**
    erroepoca[nepocas] = ei2
    nepocas = nepocas + 1
  **fim enquanto**
**fim função**

```
trainadaline <- function(xin,yd,eta,tol,maxepocas,par)
{
  dimxin<-dim(xin)
  N<-dimxin[1]
  n<-dimxin[2]

  if (par==1){
    wt<-as.matrix(runif(n+1)-0.5)
    xin<-cbind(1,xin)
  }
  else wt<-as.matrix(runif(n)-0.5)

  nepocas<-0
  eepoca<-tol+1

  evec<-matrix(nrow=1,ncol=maxepocas)
  while ((nepocas < maxepocas) && (eepoca>tol))
  {
    ei2<-0
    xseq<-sample(N)
    for (i in 1:N)
    {
      irand<-xseq[i]
      yhati<-1.0*((xin[irand,] %*% wt))
      ei<-yd[irand]-yhati
      dw<-eta*ei*xin[irand,]
      wt<-wt+dw
      ei2<-ei2+ei*ei
    }
    nepocas<-nepocas+1
    evec[nepocas]<-ei2/N

    eepoca<-evec[nepocas]
  }
  retlist<-list(wt,evec[1:nepocas])
  return(retlist)
```

```
37 }
```

Listagem 3.1: Treinamento do Adaline.

### 3.4.5 Exemplos de treinamento do Adaline

**Problema univariado**

Considere para este exemplo um modelo Adaline de apenas uma entrada $x$, de tal forma que a sua saída seja obtida como $\hat{y} = w_1 x + w_0$, ou seja, o modelo tem dois parâmetros $w_0$ e $w_1$. O problema envolverá a estimativa dos parâmetros da função aproximadora a partir de dados gerados sinteticamente por meio da função $f(x) = 4x + 2$. Suponha que a função $f(x)$ represente um processo industrial que aumenta em 4 vezes a magnitude de qualquer sinal em sua entrada e adiciona duas unidades ao valor obtido. É claro que a função não é conhecida de antemão, portanto, para estimá-lo iremos considerar que os sinais na entrada e saída da planta foram amostrados e armazenados para a estimativa que ela executa. Suponha que, com este objetivo, o sinal $seno(x)$ foi apresentado à entrada da planta e, ao mesmo tempo, a sua resposta $y$ foi também amostrada, formando, assim, os pares $(x(t), y(t))$ do conjunto de treinamento, conforme trecho de código a seguir.

```
> t<-matrix(seq(0,2*pi,0.1*pi),ncol=1)
> x<-sin(t)
> y<-matrix(4*x+2,ncol=1)
```

Os vetores **x** e **y** gerados no código anterior contêm, então, os dados de treinamento, que serão utilizados na etapa seguinte. A linha de comando `retlist<-trainadaline(x,y,0.01,0.01,1000,1)` resulta no treinamento do modelo, tendo como entrada os dados amostrados e os seguintes parâmetros: $\eta = 0.01$, $tol = 0.01$, $maxtol = 50$ e $par = 1$.

```
> retlist<-trainadaline(x,y,0.01,0.01,50,1)
> w<-retlist[[1]]
> erro<-retlist[[2]]
```

Os parâmetros utilizados para o treinamento do modelo acima foram obtidos de forma experimental para efeitos da apresentação deste exemplo. Mais adiante serão discutidas formas mais sistemáticas para a obtenção destes parâmetros. No momento, o mais importante é focarmos a atenção no funcionamento do algoritmo de treinamento e nos seus resultados. Os pesos finais da aproximação são lidos da lista de retorno da função de treinamento por meio do comando `w<-retlist[[1]]`. Conforme pode ser observado nos seus valores impressos a seguir por meio do comando `print(w)`, o vetor de pesos obtido se aproxima bastante do vetor $[2, 4]^T$ que contémm os parâmetros da função geradora $f(x) = 4x + 2$. O erro ao longo do treinamento, em função do número de épocas, é apresentado na Figura 3.5.

```
> print(w)
```

```
          [,1]
[1,] 1.997360
[2,] 3.870637
```

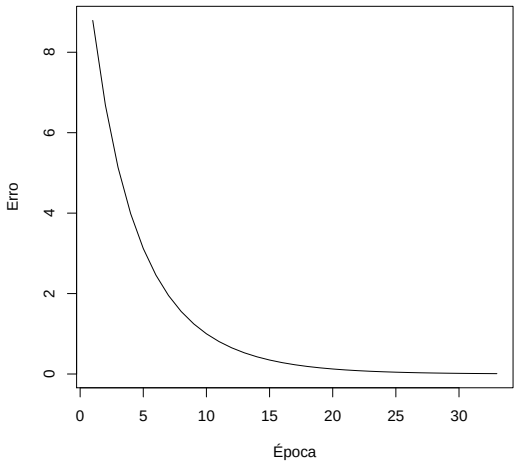



Figura 3.5: Erro durante o treinamento do modelo Adaline ao aproximar a função geradora de um sistema linear hipotético por meio de dados amostrados em suas entradas e saídas.

### Problema multivariado

Para o problema a seguir, considere que o sistema a ser modelado possui agora 4 entradas, $x_1$, $x_2$, $x_3$ e $x_4$ e que a função executada é uma mistura das 3 entradas. Assim, a função executada pelo sistema pode ser descrita de forma geral como $f(x_1, x_2, x_3) = a_1x_1 + a_2x_2 + a_3x_3 + a_0$, cujos coeficientes $\mathbf{a} = [a_0, a_1, a_2, a_3, a_4]^T$ foram selecionados empiricamente. Assim, novamente, o sistema foi estimulado com entradas arbitrárias ao longo do tempo, as quais foram amostradas, assim como os valores correspondentes da saída. Os dados de entrada e saída foram então utilizados para treinar o Adaline e aproximar a função executada pela planta. Como pode ser observado no trecho de código a seguir, o vetor de coeficientes selecionado para o exemplo foi $\mathbf{a} = [\frac{\pi}{2}, 1, 2, 0.8, 3.2]^T$, o que resultará em uma forma não-linear para a resposta do sistema, quando estimulado com $x_1 = sin(t) + cos(t)$, $x_2 = tanh(t)$, $x_3 = sin(4t)$ e $x_1 = abs(sin(t))$, em que $abs(sin(t))$ retorna o valor absoluto de $sin(t)$. As funções aplicadas às entradas foram selecionadas para que a resposta do sistema tivesse uma característica não-linear, devido à combinação destas funções, como será mostrado a seguir.

```
> t<-seq(0,2*pi,0.1*pi)
> x1<-matrix(sin(t)+cos(t),ncol = 1)
> x2<-matrix(tanh(t),ncol = 1)
> x3<-matrix(sin(4*t),ncol = 1)
> x4<-matrix(abs(sin(t)),ncol = 1)
> y<-x1+2*x2+0.8*x3+3.2*x4+pi/2
```

O trecho de código anterior mostrou a atribuição, no tempo, das variáveis $x_1$, $x_2$, $x_3$ e $x_4$ e a amostragem da saída $y$, obtida de maneira sintética por

meio da expressão $y = x + 1 + 2x_2 + 0.8x_3 + 3.2x_4 + \frac{pi}{2}$. Para que o modelo possa ser treinado pela função de treinamento `trainadaline()`, é preciso que os vetores de dados amostrados $\mathbf{x}_1(t)$, $\mathbf{x}_2(t)$, $\mathbf{x}_3(t)$ e $\mathbf{x}_4(t)$ sejam combinados em uma única matriz com 4 colunas por meio do comando `cbind(x1,x2,x3,x4)` apresentado no trecho de código a seguir. O modelo é então treinado com os mesmos parâmetros do exemplo anterior.

```
> x<-cbind(x1,x2,x3,x4)
> retlist<-trainadaline(x,y,0.01,0.01,50,1)
```

A qualidade da aproximação obtida pode ser observada no gráfico da Figura 3.6, em que as amostras de treinamento são mostradas como círculos e a resposta real do sistema, assim como a função aproximada, por meio de linhas contínuas em cores distintas. O trecho de código a seguir mostra a sequência de comandos para a geração da figura. Inicialmente a variável *ttest* é gerada com um intervalo de amostragem menor do que a variável $t$ utilizada para gerar os dados de treinamento. O objetivo é avaliar a função entre as amostras, diferentes daqueles utilizados no treinamento. Em seguida as 4 entradas são combinadas em uma matriz, desta vez de 5 colunas, já que é necessário também fixar a entrada correspondente ao termo de polarização $\frac{\pi}{2}$ em 1. No treinamento, a adição da entrada fixa em 1 foi realizada dentro da rotina de treinamento `trainadaline()`. A matriz com os dados de testes possui então 5 colunas e 201 linhas, correspondentes às amostras de testes.

```
> w<-retlist[[1]]
> ttest<-seq(0,2*pi,0.01*pi)
> x1t<-matrix(sin(ttest)+cos(ttest),ncol = 1)
> x2t<-matrix(tanh(ttest),ncol = 1)
> x3t<-matrix(sin(4*ttest),ncol = 1)
> x4t<-matrix(abs(sin(ttest)),ncol = 1)
> xt<-cbind(1,x1t,x2t,x3t,x4t)
```

Uma vez obtida a matriz de dados de teste de entrada, a resposta do modelo e a resposta real do sistema são obtidas por meio dos comandos `yt<-`xt %*% w e `yreal<-x1t+2*x2t+0.8*x3t+3.2*x4t+pi/2` no trecho de código que se segue. Finalmente, as saídas $y$, $yt$ e $yreal$ são plotadas em função de $t$ e de $ttest$ no gráfico da Figura 3.6.

```
> yt<- xt %*% w
> yreal<-x1t+2*x2t+0.8*x3t+3.2*x4t+pi/2
```

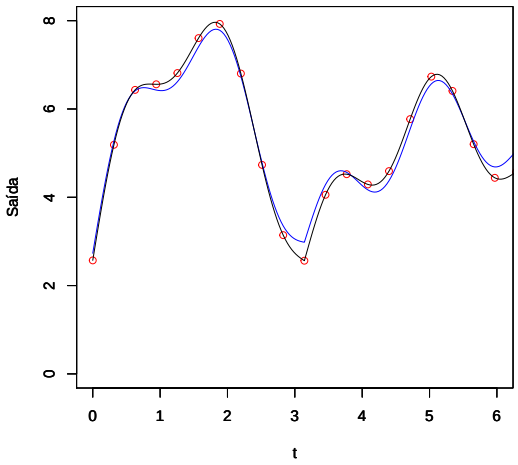



Figura 3.6: Resposta do modelo obtido e da função real para o exemplo de aproximação multivariada. Dados amostrados são apresentados como círculos vermelhos, função real como linha contínua preta e função aproximada como linha contínua azul.

# Capítulo 4

# Classificadores Lineares

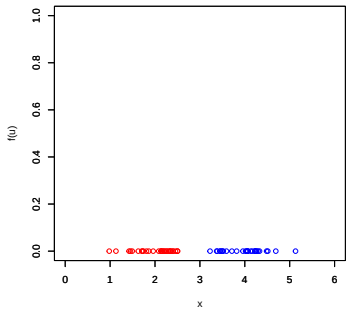
## 4.1 Introdução

Classificadores lineares são caracterizados pela aplicação de uma função separadora linear sobre o resultado de uma operação (usualmente) linear aplicada ao vetor de entrada $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$. A linearidade do classificador é caracterizada pela regra de decisão, que é aplicada diretamente sobre os dados no espaço de entrada, como uma função degrau, por exemplo. Modelos de soma-e-limiar, como o Perceptron [Ros58], que serão estudados neste capítulo, são modelos representativos de classificadores lineares. Considere, por exemplo, o problema simples de separação das amostras azuis (classe 1) e das amostras vermelhas (classe 2), cujos dados amostrados são apresentados na Figura 4.1a. Um classificador simples para discriminar entre as duas classes com base nas amostras da Figura 4.1a poderia ser implementado diretamente com a função de limiar, ou função degrau, $x < 3 \rightarrow \textit{classe}\ 1$ e $x \geq 3 \rightarrow \textit{classe}\ 2$, resultando na resposta do classificador apresentada na Figura 4.1b. Neste caso, o separador é caracterizado pelo limiar $x = 3$ sobre o eixo $x$, que caracteriza a fronteira de separação entre as duas classes.

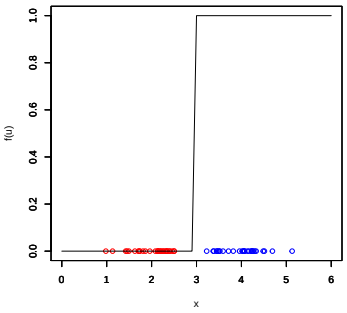
De acordo com a dimensão do problema, os classificadores lineares representarão retas, planos ou hiperplanos no espaço de entrada. Modelos como o Percetron são caracterizados pela função $f(\mathbf{x} \cdot \mathbf{w}) = \hat{y}$ em que $\mathbf{x}$ é o vetor de entrada, $\mathbf{w}$ é o vetor de parâmetros e $\hat{y}$ a saída do modelo. O produto escalar $\mathbf{x} \cdot \mathbf{w}$ caracteriza a combinação linear dos elementos de $\mathbf{x}$ pelos elementos do vetor $\mathbf{w}$, ou seja, $\mathbf{x} \cdot \mathbf{w} = \sum x_i w_i$, que corresponde, na verdade, à equação geral de um hiperplano, como será visto mais adiante neste capítulo.

## 4.2 Perceptron Simples

Nas seções anteriores foram descritas as aproximações lineares de uma única camada, caracterizadas por uma matriz de pesos $\mathbf{W}$, que mapeia a matriz $\mathbf{X}$ na matriz $\mathbf{Y}$ através da operação $\mathbf{XW} = \hat{\mathbf{Y}}$. Neste capítulo será descrito o Perceptron Simples [Ros58] que é caracterizado pela aplicação de uma não-linearidade, na forma da função de ativação $f(\cdot)$, sobre a transformação linear $\mathbf{XW}$, ou seja, a aproximação obtida corresponde a $f(\mathbf{XW}) = \hat{\mathbf{Y}}$. Neste capítulo serão estudadas as aproximações implementadas através da função degrau e de suas apro-




(a) Dados amostrados para as classes vermelha e azul.




(b) Classificador linear resultante.

Figura 4.1: Classificador linear de uma variável.

ximações sigmoidais unipolares e bipolares, realizadas, respectivamente, pelas funções logística e tangente hiperbólica.

Frank Rosenblatt descreveu o Perceptron Simples [Ros58] como uma rede neural de duas camadas, porém, o seu algoritmo de treinamento original era capaz de adaptar somente uma das camadas da rede. Como os ajustes de pesos de uma mesma camada de uma rede neural são independentes, treinar os neurônios de uma única camada equivale a treinar os seus neurônios individualmente. Por esta razão, o Perceptron Simples é frequentemente descrito na forma de um neurônio simples do tipo MCP [MP43], modelo utilizado por Rosenblatt. Assim, como há independência entre ajustes de pesos de neurônios de uma mesma camada, para efeitos de treinamento, o Perceptron Simples pode ser representado por um único neurônio MCP, como o descrito através da Equação 4.1. A implementação de uma função em R que recebe o vetores de pesos $\mathbf{w}$ e de entrada $\mathbf{x}$ e calcula a resposta do modelo Perceptron é apresentada na Listagem 4.1

$$f(u) = \begin{cases} 1 & u \geq \theta \\ 0 & u < \theta \end{cases} \tag{4.1}$$

onde $u = \sum_{i=1}^{n} x_i w_i$, $x_i$ é o valor do elemento $i$ do vetor de entrada $\mathbf{x} = [x_1, x_2, \cdots, x_i, \cdots, x_n]^T$,, $w_i$ o peso correspondente e $\theta$ o limiar de ativação do neurônio.

```
yp <- function(xvec,w)
# xvec: vetor de entrada
# w: vetor de pesos
# yp: resposta do Perceptron
{
    u<-xvec %*% w
    y<-1.0*(u>=0)

  return(as.matrix(y))
}
```

Listagem 4.1: Função **yp.R** que calcula a resposta de um perceptron simples.

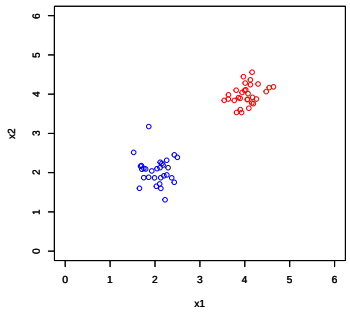
### 4.2.1 Separador Linear

De acordo com a Equação 4.1 o Perceptron Simples é um separador linear, já que sua equação de decisão $\sum_{i=1}^{n} w_i x_i = \theta$ corresponde à equação geral de um hiperplano. A linearidade de sua resposta pode ser mais facilmente observada para o caso bidimensional em que o vetor de entrada é composto por somente dois elementos: $\mathbf{x} = [x_1, x_2]^T$. Assim, a equação de decisão para o caso bidimensional se torna $w_1 x_1 + w_2 x_2 = \theta$, ou $x_2 = \frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}$, que corresponde a uma reta na forma $y = ax + b$ com inclinação $a = \frac{w_1}{w_2}$ e $b = \frac{\theta}{w_2}$, o ponto de intersecção no eixo $y$.

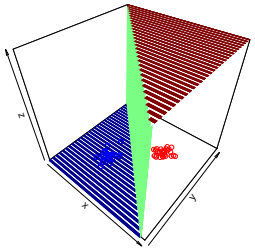
O limiar $\theta$ deve ser também um parâmetro resultante do treinamento, assim, a equação de decisão pode ser escrita como $\sum_{i=1}^{n} w_i x_i + (-\theta) = 0$ em que o parâmetro $-\theta$, multiplicado pela entrada fixa $+1$, passa a ser um termo adicional do somatório. A nova equação de decisão passa então a ser $\sum_{i=1}^{n+1} w_i x_i = 0$ para os vetores aumentados $\mathbf{x} = [+1, x_1, x_2, \cdots, x_n]^T$ e $\mathbf{w} = [-\theta, w_1, w_2, \cdots, w_n]^T$. Como o treinamento do Perceptron envolve encontrar os valores dos seus pesos ($w_1$, $w_2$ e $\theta$ no caso bidimensional), o processo de aprendizado, ou ajuste dos pesos, resulta no deslocamento da reta sobre o plano $x_1 \times x_2$ até que o critério de parada seja atingido. O critério de parada pode ser simplesmente um limite de tolerância de erro, por exemplo. Assim, o treinamento iterativo envolve deslocar a reta, ou hiperplano no caso mais geral, no espaço de entrada até que o critério seja atingido.

Com o objetivo de visualizar melhor o problema de separação bidimensional, considere os dados da Figura 4.2a, que representa um problema de separação de duas classes e duas variáveis $x_1$ e $x_2$. Para este problema, uma possível solução para a separação linear pode ser encontrada facilmente por inspeção. Um neurônio com pesos $w_0 = 6$, $w_1 = 1$ e $w_2 = 1$, em que $w_0$ corresponde ao termo de polarização $\theta$, é uma das infinitas soluções possíveis para este problema. O separador linear correspondente a estes valores de pesos possui equação $x_2 = -x_1 + 6$, a qual é capaz de fazer a separação das duas classes. A superfície de separação resultante da aplicação deste separador é apresentada na Figura 4.2b, onde pode ser observado que o classificador resultante atende ao critério de separação das amostras das duas classes. O trecho de código da Listagem 4.2 apresenta os passos em R para a geração das Figuras 4.2a e 4.2b.

```
rm(list=ls())
library('plot3D')
xc1<-matrix(0.3*rnorm(60)+2,ncol=2)
xc2<-matrix(0.3*rnorm(60)+4,ncol=2)
plot(xc1[,1],xc1[,2],xlim=c(0,6),ylim=c(0,6),xlab='x1',ylab='x2',
    col='blue')
par(new=TRUE)
plot(xc2[,1],xc2[,2],xlim=c(0,6),ylim=c(0,6),xlab='x1',ylab='x2',
    col='red')
seqx1x2<-seq(0,6,0.2)
npgrid<-length(seqx1x2)
M<-matrix(nrow=npgrid, ncol=npgrid)
ci<-0
w<-as.matrix(c(6,1,1))
for (x1 in seqx1x2)
  {
    ci<-ci+1
    cj<-0
    for (x2 in seqx1x2)
  {
```




(a) Dados amostrados para as classes vermelha e azul.



(b) Classificador linear resultante.

Figura 4.2: Classificador linear de duas variáveis.

```
    cj<-cj+1
    xin<-as.matrix(cbind(-1,x1,x2))
    M[ci,cj]<-1.0*((xin %*% w) >= 0) #yperceptron(xin,c(1.5,1,1),1)
  }
}

ribbon3D(seqx1x2,seqx1x2,xlim=c(0,6),ylim=c(0,6),M,colkey = F)
scatter3D(xc1[,1],xc1[,2],matrix(0,nrow = dim(xc1)[1]),add = T,col=
    'blue',colkey = F)
scatter3D(xc2[,1],xc2[,2],matrix(0,nrow = dim(xc1)[1]),add = T,col=
    'red',colkey = F)
```

Listagem 4.2: Trecho de código em R para implementar o exemplo da Figura 4.2.

### 4.2.2 Treinamento do Perceptron

O Perceptron é usualmente treinado por correção de erros, ou seja, o erro na saída do modelo resulta no ajuste do vetor de pesos com o objetivo de fazer a sua correção. Para o caso do Perceptron com função de ativação degrau, conforme descrito na Seção 4.2.1, só há duas situações possíveis em que pode haver erro na resposta do modelo, já que a sua saída pode assumir somente dois valores, 0 ou 1. É claro que não ocorrerá correção de pesos caso a saída do modelo corresponda ao valor desejado para a saída, ou seja, caso $\hat{y}_j = y_j$ para o vetor de entrada $\mathbf{x}_j$. A correção dos pesos para as duas situações possíveis em que $\hat{y}_j \neq y_j$ é descrita a seguir [BCL07]:

1. **Situação 1**: o rótulo $y_j$ do vetor de entrada $\mathbf{x}_j$ é 1 mas a resposta $\hat{y}_j$ do Perceptron para o vetor de pesos atual é 0 ($y_j = 1$ e $\hat{y}_j = 0$). Neste caso, considerando os vetores aumentados, conforme descrito na Seção 4.2.1, o limiar de ativação da saída é 0, o que implica que $u = \sum_{i=1}^{n+1} w_i x_i < 0$, já que $\hat{y}_j = 0$. Assim, como $\sum_{i=1}^{n+1} w_i x_i = |\mathbf{w}||\mathbf{x}| cos(\mathbf{w}, \mathbf{x})$, temos que $cos(\mathbf{w}, \mathbf{x}) < 0$, já que $|\mathbf{w}| \geq 0$ e $|\mathbf{x}| \geq 0$. Em outras palavras, a resposta $\hat{y}_j = 0$ resulta do fato de o cosseno do ângulo entre $\mathbf{w}$ e $\mathbf{x}$ ser negativo, ou seja $\frac{\pi}{2} < \alpha < \frac{3\pi}{2}$. Esta situação é mostrada na Figura 4.3a.

2. **Situação 2**: o rótulo $y_j$ do vetor de entrada $\mathbf{x}_j$ é 0 mas a resposta $\hat{y}_j$ do Perceptron para o vetor de pesos atual é 1 ($y_j = 0$ e $\hat{y}_j = 1$). Utilizando um raciocínio análogo ao apresentado para a Situação 1, neste caso o cosseno do ângulo entre $\mathbf{w}$ e $\mathbf{x}$ é positivo, situação mostrada na Figura 4.3b. Neste caso, o ângulo $\alpha$ entre $\mathbf{x}$ e $\mathbf{w}$ é menor que $\frac{\pi}{2}$ ($\alpha < \frac{\pi}{2}$).


(a) Situação 1 ($\hat{y} = 0$). (b) Situação 2 ($\hat{y} = 1$).

Figura 4.3: Representação esquemática das posições relativas dos vetores $\mathbf{w}$ e $\mathbf{x}$ para as duas respostas possíveis de um Perceptron.

Assim, para que haja mudança na resposta $\hat{y}_j$, ou seja para que $\hat{y}_j = y_j$ em ambas as situações de erro apresentadas na Figura 4.3, $\alpha$ deve ser reduzido na Situação 1 e aumentado na Situação 2. Como $\mathbf{w}$ é o vetor cujos elementos devem ser ajustados, este deve ser alterado para que o ângulo $\alpha$ atualizado resulte na inversão dos valores de saída $\hat{y}_j$ em ambas as situações. O vetor $\mathbf{x}$ fornece, portanto, a direção de ajuste nas duas situações de erro, conforme representado na Figura 4.4.


(a) Correção para Situação 1. (b) Correção para a Situação 2.

Figura 4.4: Representação esquemática da correção do vetor $\mathbf{w}$ para as duas situações de erro possíveis apresentadas na Figura 4.3.

Deve-se ter, portanto, o ajuste de pesos $\Delta\mathbf{w} \propto \mathbf{x}_j$ na primeira situação e $\Delta\mathbf{w} \propto -\mathbf{x}_j$ na segunda, mas como $e_j = y_j - \hat{y}_j = 1$ na primeira situação e $e_j = y_j - \hat{y}_j = -1$ na segunda, o erro já fornece a direção de ajuste. Chega-se assim a uma equação geral para o ajuste dos pesos, considerando-se o erro: $\Delta\mathbf{w} = \eta e_j \mathbf{x}_j$. A Equação 4.2 descreve de forma geral a regra de ajuste do vetor de pesos $\mathbf{w}$ na iteração $t$.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta e(t)\mathbf{x}(t) \tag{4.2}$$

em que $\mathbf{w}(t)$, $e(t)$ e $\mathbf{x}(t)$ representam, respectivamente, os valores do vetor de pesos, do erro e do vetor de entrada no instante $t$.

### 4.2.3 Implementação em R do treinamento do Perceptron

A Equação 4.2 apresenta, de forma geral, a regra de atualização dos pesos de um Perceptron simples, visando à minimização do erro de treinamento. Para o caso de problemas linearmente separáveis, esta regra de ajuste resulta na convergência do modelo em um número finito de passos, conforme será discutido nas seções seguintes. Em outras palavras, a aplicação iterativa da Equação 4.2 para todos os dados do conjunto de treinamento resulta em um vetor de pesos cujo hiperplano correspondente é uma solução para o problema de classificação binária representado nos dados. Assim, o algoritmo de treinamento do Perceptron envolve basicamente a aplicação da Equação 4.2 a todos os dados do conjunto de treinamento até que o erro global seja nulo, ou atinja um valor de tolerância pré-estabelecido. Uma implementação possível para o algoritmo, na forma de uma função de treinamento, é apresentada na Listagem 4.3.

```
treinap <- function(xin,yd,eta,tol,maxepocas,par)
# Rotina de treinamento do Perceptron simples.
# xin: matriz Nxn com os dados de entrada
# yd:  rótulos de saída (0 ou 1)
# eta: passo de treinamento
# tol: tolerância de erro
# maxepocas: número máximo de iterações
# par: parâmetro de entrada.
  # par=0 ==> xin tem dimensão n+1 e já inclui
  #           entrada correspondente ao termo
  #           de polarização.
  # par=1 ==> xin tem dimensão n e não inclui
  #           entrada correspondente ao termo de
  #           polarização, que deve ser adicionado
  #           dentro da função.

{
  dimxin<-dim(xin)        # Dimensões do conjunto de dados.
  N<-dimxin[1]            # Número de amostras.
  n<-dimxin[2]            # Dimensão de entrada.

  # Adiciona ou não termo de polarização ao vetor de
  # treinamento w.
  if (par==1){
    wt<-as.matrix(runif(n+1)-0.5)
    xin<-cbind(-1,xin)
  }
  else wt<-as.matrix(runif(n)-0.5)

  nepocas<-0    # Contador de épocas.
  eepoca<-tol+1 # Acumulador de erro de épocas.
  evec<-matrix(nrow=1,ncol=maxepocas) #  Vetor de erros.

  # Laço principal de treinamento
  while ((nepocas < maxepocas) && (eepoca>tol))
  {
    ei2<-0
    # Sequência aleatória de treinamento.
    xseq<-sample(N)
    for (i in 1:N)
    {
      # Amostra dado da sequência aleatória.
      irand<-xseq[i]
      # Calcula saída do Perceptron
      yhati<-1.0*((xin[irand,] %*% wt) >= 0)
```

```
46        ei<-yd[irand]-yhati
47        dw<-eta*ei*xin[irand,]
48        # Ajusta vetor de pesos.
49        wt<-wt+dw
50        # Acumula erro por época.
51        ei2<-ei2+ei*ei
52      }
53      # Incrementa número de épocas.
54      nepocas<-nepocas+1
55      evec[nepocas]<-ei2/N
56      # Armazena erro por época.
57      eepoca<-evec[nepocas]
58    }
59    # Retorna vetores de pesos e de erros.
60    retlist<-list(wt,evec[1:nepocas])
61    return(retlist)
62  }
```

Listagem 4.3: Função de treinamento de um perceptron simples.

### 4.2.4 Treinamento pelo Gradiente Descendente

A Equação de ajuste 4.2 foi obtida considerando-se que o modelo tenha saída binária unipolar (saída pode ser 0 ou 1). No entanto, nem sempre o Perceptron utilizará saída binária, particularmente em estruturas de redes de múltiplas camadas, para a qual a Equação 4.2 não se aplica diretamente. Nestas situações há uma grande de variedade de algoritmos de treinamento que se aplicam ao problema, para os quais é frequentemente necessário avaliar o gradiente da função de custo, soma dos erros quadráticos, por exemplo, à cada iteração. Assim, para que o gradiente seja avaliado, é necessário que a função de ativação do neurônio seja diferenciável, o que não é o caso da função degrau utilizada originalmente no Perceptron, devido à sua descontinuidade. Mantendo-se as características de separabilidade dos modelos tipo Perceptron, aproximações contínuas sigmoidais da função degrau, como a função logística ($f(u) = \frac{1}{1+e^{-\beta u}}$) e a tangente hiperbólica são usualmente utilizadas. Apesar de a separabilidade requerer uma separação rígida do espaço em duas regiões, o que é intrínseco para o caso da função degrau, esta pode ser também obtida por meio dos pontos de inflexão das funções sigmoidais, conforme Figura 4.5.

Para o caso geral, considere que, ao invés da Equação 4.1 a saída do Perceptron simples seja gerada por uma função genérica diferenciável $\hat{y}_j = f(u_j)$ e que o ajuste dos pesos seja obtido por meio da minimização do erro quadrático sobre todo o conjunto de treinamento, conforme Equação 4.3.

$$J = \frac{1}{2} \sum_j \left(y_j - f(u_j)\right)^2 \tag{4.3}$$

Considerando que o ajuste dos pesos seja feito individualmente por padrão, somente o termo referente ao mesmo afeta o valor de $J$ no somatório da Equação 4.3. Para o padrão arbitrário $\mathbf{x}_j$, o termo que afetará o gradiente é apresentado na Equação 4.4.

$$J = \frac{1}{2} \left(y_j - f(u_j)\right)^2 \tag{4.4}$$

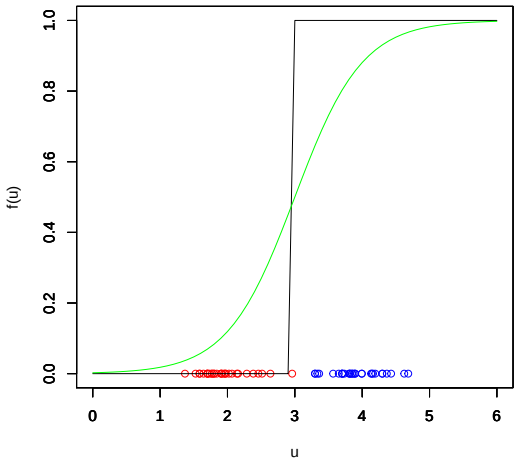



Figura 4.5: Aproximação contínua da função degrau ($f(u) = \frac{1}{1+e^{-2(u-3)}}$).

Assim, o ajuste $\Delta w_i$ visando à minimização da Equação 4.4 em direção contrária ao gradiente da função em cada ponto deve ser tal que $\Delta w_i \propto -\frac{\partial J}{\partial w_i}$. A derivada parcial de $J$ da Equação 4.4 em relação a um peso genérico $w_i$ é apresentada de forma genérica na Equação 4.5.

$$\frac{\partial J}{\partial w_i} = \frac{1}{2}\frac{\partial}{\partial w_i}\left(y_j - f(u_j)\right)^2 \tag{4.5}$$

Como, pela Regra da Cadeia, $\frac{\partial}{\partial w_i}\left(y_j - f(u_j)\right)^2 = 2\left(y_j - f(u_j)\right)\frac{\partial f(u_j)}{\partial u_j}\frac{\partial u_j}{\partial w_i}$ e $u_j = w_1x_1 + w_2x_2 + \cdots + w_ix_i + \cdots + w_nx_n$, temos que $\frac{\partial f(u_j)}{\partial u_j} = f'(u_j)$ e $\frac{\partial}{\partial w_i}(w_1x_1+w_2x_2+\cdots+w_ix_i+\cdots+w_nx_n) = x_i$. Chega-se, assim, à Equação 4.6 como expressão do gradiente da função apresentada na Equação 4.6 na dimensão $w_i$ do vetor de pesos $\mathbf{w}$ para o ajuste do erro relativo ao padrão $\mathbf{x}_j$.

$$\frac{\partial J}{\partial w_i} = -ef'(u_j)x_j \tag{4.6}$$

Assim, a expressão final para o ajuste de pesos pelo método do gradiente descendente para uma função de ativação genérica $f(u)$ é apresentada na Equação 4.7, já que $\Delta\mathbf{w}_i \propto -\frac{\partial J}{\partial w_i}$ e $w_i(t+1) = w_i(t) + \Delta\mathbf{w}_i(t)$, onde $t$ representa o número de iteração.

$$w_i(t+1) = w_i(t) + \eta e(t) f'(u_j)x_j(t) \tag{4.7}$$

Para o caso de a função de ativação utilizada ser a tangente hiperbólica, $tanh(u)$, cuja derivada é $sech^2(u)$, a equação de ajuste se torna simplesmente $w_i(t+1) = w_i(t) + \eta e(t) sech^2(u)x_j(t)$.

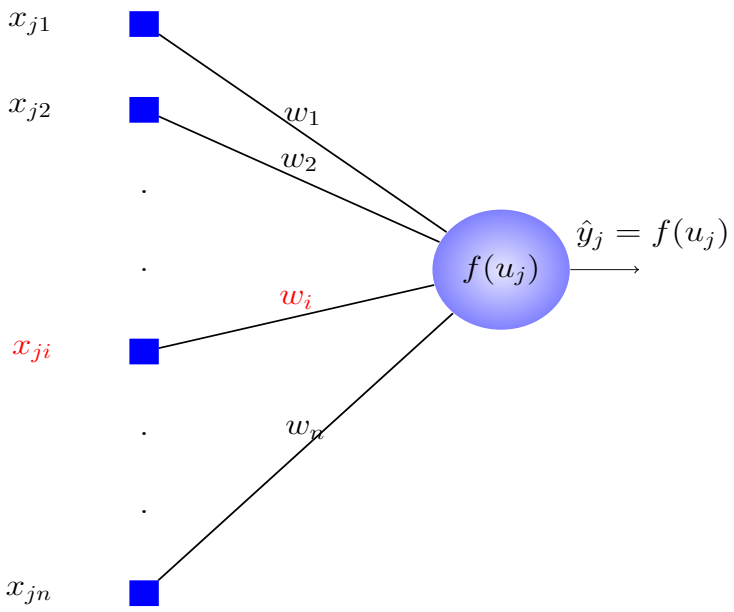



Figura 4.6: Neurônio de um Perceptron simples com $n$ entradas, representando a saída para um vetor de entrada arbitrário $\mathbf{x}_j = [x_{j1}, x_{j1}, \cdots, x_{jn}]^T$. O peso a ser ajustado $w_{ji}$ assim como a entrada correspondente $x_{ji}$ são indicados em vermelho.

### 4.2.5 Um Exemplo de Aplicação

O código a seguir apresenta uma aplicação do Perceptron ao problema de classificação da base de dados Iris, nativa ao R e bastante utilizada para testes de modelos de aprendizado de máquina. A base Iris foi descrita no trabalho de Edgar Anderson [And36] e é muito conhecida nas áreas de Estatística e de Aprendizado de Máquina, por ter sio utilizada por Ronald Fisher em seu trabalho pioneiro de discriminantes lineares, em 1936 [Fis36]. A leitura da base é feita diretamente com o comando `data(iris)`, conforme código a seguir. A base é composta por 50 amostras de cada uma das espécies *setosa*, *versicolo*r e *virginica* da flor Iris. Para cada uma das 150 amostras, foram colhidas medidas de comprimento e largura das suas sépalas e pétalas, resultando nas 4 variáveis Sepal.Length, Sepal.Width, Petal.Length e Petal.Width na base de dados.

```
> data(iris)
```

Exemplos de amostras de cada uma das espécies são apresentadas a seguir, as quais foram obtidas por meio do comando `print()`.

```
> print(rbind(iris[1,],iris[51,],iris[101,]))
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
51           7.0         3.2          4.7         1.4 versicolor
101          6.3         3.3          6.0         2.5  virginica
```

A Figura 4.7 mostra as 150 amostras das 3 classes (espécies da flor) em cores diferentes apresentadas aos pares de variáveis. Conforme pode ser observado, há uma separação espacial das amostras, que indica que o problema pode ser tratado com um separador linear, mesmo quando analisado no espaço bi-dimensional, como na figura.

```
> plot(iris,col=c("red","green3","blue")[unclass(iris$Species)])
```

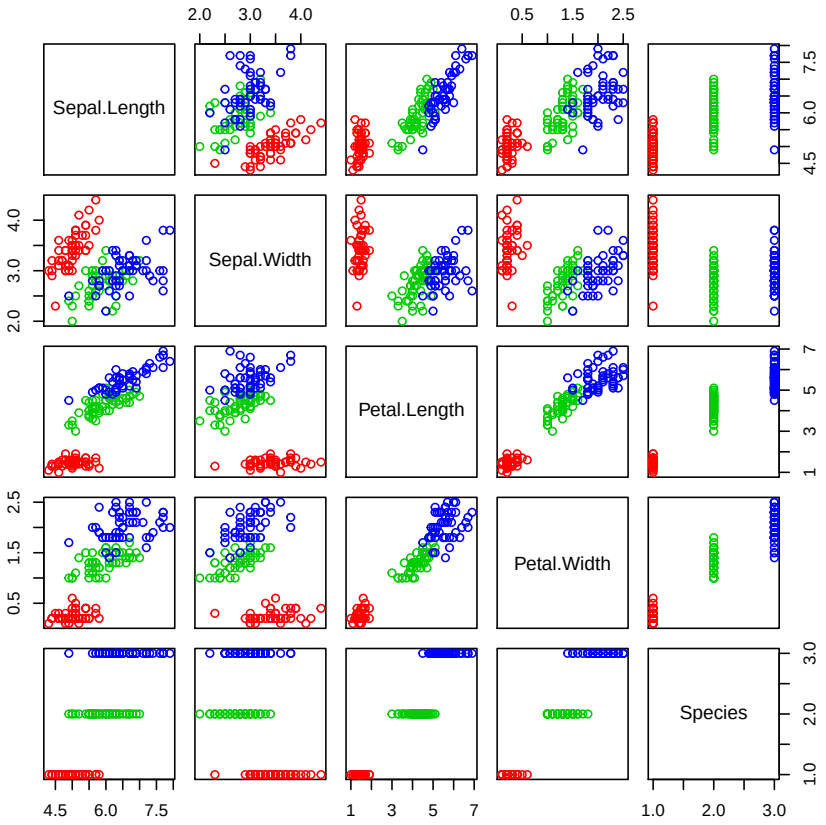



Figura 4.7: Amostras das 3 classes do conjunto de dados Iris para todos os pares de variáveis de entrada.

Como o problema da Iris tem 3 classes e, no momento, estamos estudando classificadores binários, o problema original será tratado como um problema de duas classes para a sua utilização com o Perceptron Simples. Assim, ao invés de discriminar as 3 classes, o nosso problema será aqui o de discriminar a classe setosa da classe vesicolor. A classe 1 do nosso problema será, então, caracterizada pelas 50 primeiras amostras e a classe 2 pelas 50 amostras seguintes, armazenadas nas matrizes $xc1$ e $xc2$, conforme linhas de código a seguir.

```
> xc1<-as.matrix(iris[1:50,1:4])
> xc2<-as.matrix(iris[51:100,1:4])
```

Conforme trecho de código anterior, o conjunto de amostras da classe 1 possui tamanho $N_1 = 50$ e da classe 2, $N_2 = 50$. Com objetivo de selecionar o mesmo número de amostras para cada classe, serão selecionadas 30 amostras aleatórias para cada uma das classes, conforme trecho de código que segue. Observe que os rótulos, ou seja, os valores de $y$, atribuídos por meio dos comandos `yc1treina<-matrix(0,nrow=ntrain)` e `yc2treina<-matrix(1,nrow=ntrain)`, já que os conjuntos das amostras de cada classe são conhecidos de antemão. Os vetores `seqc1` e `seqc2` contêm índices aleatórios para seleção de amostras das classes 1 e 2.

```
> ntrain<-30
> seqc1<-sample(50)
> xc1treina<-xc1[seqc1[1:ntrain],]
> yc1treina<-matrix(0,nrow=ntrain)
> seqc2<-sample(50)
> xc2treina<-xc2[seqc2[1:ntrain],]
> yc2treina<-matrix(1,nrow=ntrain)
```

Com o objetivo de avaliar o desempenho do modelo final obtido, um novo conjunto de testes é selecionado para cada amostra. A seleção das amostras restantes utiliza os índices aleatórios armazenados nas sequências `seqc1` e `seqc2`, conforme código a seguir.

```
> xc1teste<-xc1[seqc1[(ntrain+1):50],]
> yc1teste<-matrix(0,nrow=(50-ntrain))
> xc2teste<-xc2[seqc2[(ntrain+1):50],]
> yc2teste<-matrix(1,nrow=(50-ntrain))
```

Como as rotinas de treinamento não consideram os conjuntos separados por classe, as matrizes xc1treina, xc2treina, yc1treina, yc2treina, xc1teste, xc2teste, yc1teste e yc2teste devem ser concatenadas, o que é realizado por meio da função `rbind()` no código que segue.

```
> xin<-as.matrix(rbind(xc1treina,xc2treina))
> yd<-rbind(yc1treina,yc2treina)
> xinteste<-as.matrix(rbind(xc1teste,xc2teste))
> yteste<-rbind(yc1teste,yc2teste)
```

Treinamento foi realizado utilizando-se a função `trainperceptron()` com parâmetros obtidos experimentalmente, conforme trecho de código a seguir. O leitor pode alterar os parâmetros utilizados e observar de que forma eles influenciam no resultado final. A avaliação do modelo é feita por meio do comando `yt<-yperceptron(xinteste,wt,1)`, cujo resultado deve ser comparado com o vetor `yteste`. A acurácia final do modelo pode ser estimada por meio da operação $1 - \frac{y_{teste}^T y_t}{20}$, sendo 20 o número de amostras de teste, conforme comando `acuracia<-1-(t(yteste-yt)`%*% (yteste-yt) )/20 .

```
> retlist<-trainperceptron(xin,yd,0.1,0.01,100,1)
> wt<-retlist[[1]]
> yt<-yperceptron(xinteste,wt,1)
> acuracia<-1-(t(yteste-yt) %*% (yteste-yt) )/20
> print(acuracia)
```

```
     [,1]
[1,]    1
```

# Capítulo 5

# Máquinas de Aprendizado Extremo - ELM

## 5.1 Introdução

O tratamento independente das funções $\phi_h(\mathbf{x}_i, \mathbf{Z})$ e $\Phi_o(\mathbf{h}_i, \mathbf{w})$ surgiu inicialmente na literatura com a descrição das redes neurais do tipo RBF [BL88]. Embora tenham sido descritos algoritmos de treinamento visando a obter soluções conjuntas das duas funções, para este tipo de modelo usualmente a função $\phi_h(\mathbf{x}_i, \mathbf{Z})$ é obtida de maneira independente através de métodos de agrupamento de dados. Nesta forma de treinamento, cada função que compõe o mapeamento $\phi_h(\mathbf{x}_i, \mathbf{Z})$ responde por uma região do espaço de entrada. Na situação degenerada em que o número de regiões $c$ é igual ao número de amostras $N$, para cada vetor de entrada $\mathbf{x}_i$ uma função $h_i(\mathbf{x}, \mathbf{z}_i)$ é alocada na camada escondida, podendo resultar em um modelo sobre-ajustado. De uma maneira geral o treinamento da camada intermediária de redes RBF através de métodos de agrupamento de dados visa à ortogonalização das projeções dos agrupamentos no espaço da camada escondida. A resolução do problema linear na camada de saída é então obtida através de um hiperplano que passa pela origem do novo sistema de coordenadas e que separa as duas classes de entrada. A projeção, ou ortogonalização dos agrupamentos de entrada sem que isto resulte em um modelo sobre-ajustado, depende de parâmetros a serem ajustados muitas vezes *ad-hoc* pelo usuário sem garantia de solução global ótima.

Uma solução elegante para o problema, baseada nos princípios do Aprendizado Estatístico [Vap95], surgiu com a descrição das SVMs [BGV92] no início da década de 1990. Um dos princípios básicos das SVMs é a convexificação do problema de aprendizado através do mapeamento não-linear realizado por uma função de kernel $K(\mathbf{x}_i, \mathbf{x}_j)\ \forall i, j$, que executa o produto cruzado entre todos os padrões de entrada. Os produtos de kernel representam então o novo mapeamento no *espaço de características* onde existe uma solução ótima para o separador linear que minimiza o erro e maximiza a margem de separação entre os padrões mapeados. A obtenção da solução *ótima* de SVMs, no entanto, é condicionada a parâmetros de kernel e de margem que devem ser pré-estabelecidos pelo usuário. A solução é ótima para um dado conjunto de valores destes parâmetros os quais o usuário não sabe de antemão quais devem ser. O ajuste do

modelo é usualmente realizado através de buscas exaustivas como *grid-search* e validação cruzada.

As ELMs (*Extreme Learning Machines*) [HZS04], ou Máquinas de Aprendizado Extremo, se baseiam nos mesmos princípios dos modelos descritos nos parágrafos anteriores, porém, com uma interpretação particular do Teorema de Cover [Cov65]. Como no Teorema não há nenhuma restrição sobre a forma do mapeamento $\phi_h(\mathbf{x}_i, \mathbf{Z})$, o princípio das ELMs é que a matriz $\mathbf{Z}$ seja selecionada de forma aleatória e que o número de funções $h_i(\mathbf{x}, \mathbf{z}_i)$ seja suficientemente grande para garantir a separabilidade no espaço da camada intermediária. Uma vez garantida uma projeção linearmente separável, o problema volta a ser o de encontrar o separador linear, o que pode ser feito de maneira direta através da pseudo-inversa como será visto mais adiante.

No entanto, alguns princípios básicos devem ser satisfeitos para que a solução do problema através de ELMs seja plausível:

- A escolha aleatória da matriz de pesos $\mathbf{Z}$ deve garantir a separação linear no espaço da camada intermediária.
- Aumento da dimensão do espaço da camada intermediária para garantir a separabilidade não afeta diretamente o sobre-ajuste do modelo aos dados.
- Uma vez que a separabilidade esteja garantida pela expansão da camada intermediária, uma solução direta de erro mínimo e margem máxima pode ser obtida.

No restante deste capítulo as principais propriedades das ELMs serão apresentadas.

## 5.2 Treinamento de ELMs

Uma vez obtida a matriz de pesos $\mathbf{Z}$ de forma aleatória e então a matriz de mapeamento $\mathbf{H}$ através da operação $\mathbf{H} = \Phi_h(\mathbf{xZ})$ em que $\Phi_h(\cdot)$ é usualmente uma função sigmoidal como a tangente hiperbólica, o problema de treinamento consiste em encontrar a matriz de pesos $\mathbf{W}$ que satisfaça à Equação 5.1.

$$\Phi_o(\mathbf{HW}) = \mathbf{Y} \tag{5.1}$$

onde $\mathbf{W}$ é a matriz de pesos de dimensões $p \times m$ da camada de saída e $\Phi(\cdot)$ é a função que mapeia a camada intermediária à camada de saída.

A função $\Phi_o(\cdot)$ é tipicamente uma função de limiar para problemas de classificação, no entanto, a formulação de ELMs propõe linearizar a Equação 5.1 simplificando a sua solução ao reduzí-la ao sistema linear $\mathbf{HW} = \mathbf{Y}$, cuja solução pode ser obtida diretamente como $\mathbf{W} = \mathbf{H}^{-1}\mathbf{Y}$ ou na forma mais genérica, conforme representado pela Equação 5.2.

$$\mathbf{W} = \mathbf{H}^{+}\mathbf{Y} \tag{5.2}$$

onde $\mathbf{H}^{+}$ é a pseudo-inversa de $\mathbf{H}$.

De fato, a simplificação do problema de classificação resultante da solução da Equação 5.1 através da Equação 5.2 trata a superfície de erro correspondente a $(\mathbf{Y} - \Phi(\mathbf{HW}))^2$ como a sua aproximação $(\mathbf{Y} - \mathbf{HW})^2$, que é quadrática nos parâmetros da matriz $\mathbf{W}$.

## 5.3 Treinamento de ELMs

Para o exemplo de treinamento de ELMs desta seção serão utilizados dados sintéticos de um problema de classificação binária em que os dados das duas classes estão distribuídos na forma de "ou-exclusivo", conforme Figura 5.1.

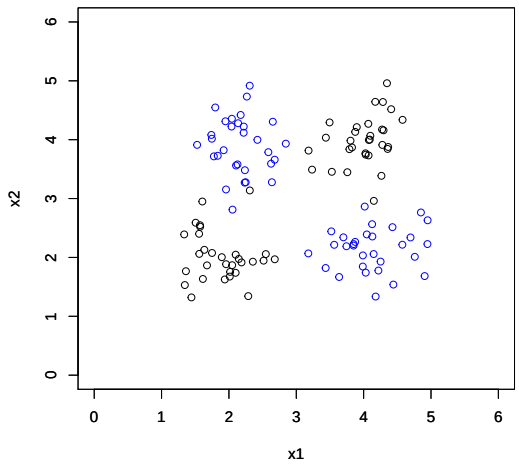



Figura 5.1: Problema de classificação binária em que as duas classes estão dispostas na forma de ou-exclusivo.

Uma vez lidos os dados de entrada, o próximo passo é gerar a matriz $\mathbf{Z}$, que contém os pesos dos neurônios da camada escondida, de forma aleatória. O trecho de código que se segue gera a matriz $\mathbf{Z}$ com pesos amostrados de uma distribuição uniforme definida no intervalo $[-0,5, +0,5]$ através do comando $replicate(p, runif(3, -0.5, 0.5))$, em que $p$ é o número de neurônios da camada escondida. É importante observar aqui que, apesar de o número de variáveis de entrada ser $n = 2$, a matriz é gerada com uma linha adicional para o termo de polarização, resultando na dimensão $(n + 1) \times p$.

```
> p<-20
> Z<-replicate(p, runif(3,-0.5,0.5))
```

A partir da matriz de pesos $\mathbf{Z}$, obtém-se então a projecão da matrix $\mathbf{X}$ na matriz $\mathbf{H}$ da camada escondida através da operação $\Psi_h(\mathbf{XZ}) = \mathbf{H}$ em que $Psi_h(\cdot)$ é usualmente uma função sigmoidal do tipo tangente hiperbólica. Novamente, para que o termo de polarização da camada escondida seja considerado, uma coluna é adicionada à matriz $\mathbf{X}$, gerando no código que se segue a matriz $\mathbf{X}_{aug}$ com dimensão $N \times (n + 1)$.

```
> Xaug<-cbind(replicate(nc*4,1),X)
> H<-tanh(Xaug %*% Z)
```

O vetor de pesos da camada de saída é então obtido através da operação $\mathbf{W} = \mathbf{H}^{+}\mathbf{Y}$ no código que se segue. Alternativamente, a pseudoinversa pode ser obtida

diretamente por meio da sua definição: `W<-solve(t(H)%*%H+0.1*diag(p))%*%t(H)%*%Y` .

```
> W<-pseudoinverse(H) %*% Y
```

Uma vez obtidos os parâmetros, representados pela matriz $\mathbf{Z}$ e pelo vetor $\mathbf{W}$ deseja-se avaliar o desempenho do modelo resultante tanto no conjunto de treinamento quanto no conjunto de testes. Inicialmente, para o conjunto de treinamento $\mathbf{X}$, como a matriz $\mathbf{H}$ já é conhecida, pode-se obter os valores de saída estimados pelo modelo através da operação $\hat{\mathbf{Y}} = sign(\mathbf{HW})$. A partir dos valores estimados $\hat{\mathbf{Y}}$ o erro do conjunto de treinamento pode então ser obtido, conforme trecho de código que se segue. O erro obtido, equivalente ao número de vetores classificados incorretamente, deve ser dividido por 4, já que $(y_i - \hat{y}_i)^2 = 4 \; \forall y_i \neq \hat{y}_i$.

```
> Yhat_train<-sign(H %*% W)
> e_train<-sum((Y-Yhat_train)^2)/4
> print(e_train)

[1] 3
```

Para se obter o erro do conjunto de testes, uma nova matriz de projeção $\mathbf{H}_t$ deve ser obtida para que as respostas do modelo ao novo conjunto de dados seja estimada. O arquivo **data2classXOR.txt** já contém dados de teste na matriz de dados $\mathbf{X}_t$, assim a matriz $\mathbf{H}_t$ e as respostas $\hat{\mathbf{Y}}_t$ ao conjunto $\mathbf{X}_t$ podem ser obtidas de maneira análoga à obtenção de $\hat{\mathbf{Y}}$ para o conjunto de treinamento, conforme código que se segue.

```
> Xaug_t<-cbind(replicate(nc*4,1),X_t)
> H_t<-tanh(Xaug_t %*% Z)
> Yhat_t<-sign(H_t %*% W)
> e_t<-sum((Y-Yhat_t)^2)/4
> print(e_t)

[1] 1
```

De forma a visualizar a resposta do modelo ELM obtido, o contorno da superfície de separação correspondente é apresentado na Figura 5.2. Como pode ser observado, a superfície não-linear obtida é capaz de fazer a separação das duas classes.

## 5.4 Exemplos de Aplicação

Com o objetivo de facilitar o uso geral das redes ELM, na Listagem 5.1 é apresentada uma função para o treinamento e na Listagem 5.2 outra função para o cálculo da saída de uma rede ELM, as quais serão utilizadas nos exemplos que seguem.

```
1 treinaELM<-function(xin, yin, p, par)
2 {
3   n<-dim(xin)[2]            # Dimensão de entrada.
4
```

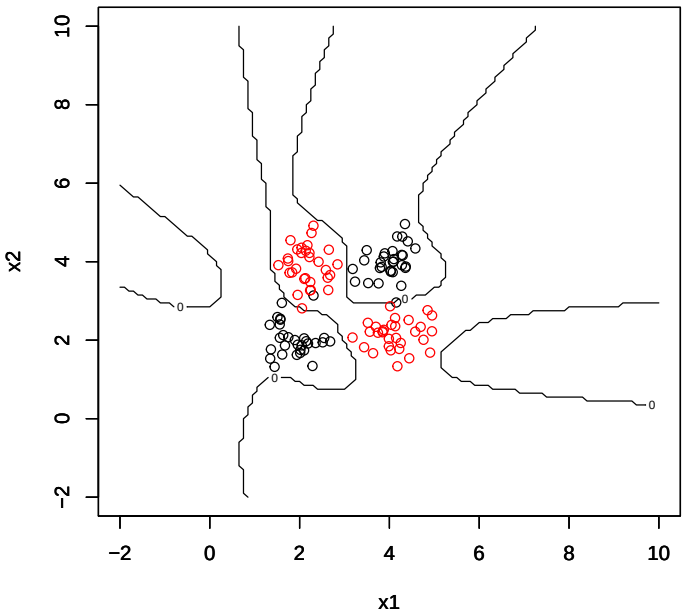



Figura 5.2: Linha de contorno da superfície de separação do problema da Figura 5.1 obtida com uma ELM.

```
  # Adiciona ou não termo de polarização
  if (par==1){
    xin<-cbind(1,xin)
    Z<-replicate(p, runif((n+1),-0.5,0.5))
  }
  else Z<-replicate(p, runif(n,-0.5,0.5))

  H<-tanh(xin %*% Z)

  # W<-pseudoinverse(H) %*% yin
  W<-(solve(t(H) %*% H) %*% t(H)) %*% yin

  return(list(W,H,Z))
}
```

Listagem 5.1: Função de treinamento de uma rede ELM.

é apresentada na Listagem 5.2.

```
YELM<-function(xin, Z, W, par)
{
  n<-dim(xin)[2]            # Dimensão de entrada.

  # Adiciona ou não termo de polarização
  if (par==1){
    xin<-cbind(1,xin)
  }

```

```
10   H<-tanh(xin %*% Z)
11   Yhat<-sign(H %*% W)
12
13   return(Yhat)
14 }
```

Listagem 5.2: Função que calcula a saída de uma rede ELM.

### 5.4.1 Conjunto da Iris

```
> rm(list = ls())
> source(file='/home/apbraga/PRINCIPAL/livros/LIVRO_R/rotinasGPL/treinaELM.R')
> source(file='/home/apbraga/PRINCIPAL/livros/LIVRO_R/rotinasGPL/YELM.R')
> library('RSNNS')
> data(iris)
> xseq<-sample(100)
> xall<-as.matrix(iris[xseq,1:4])
> yall<-(1*(iris$Species[xseq] == 'versicolor')-0.5)*2
> xyall<-splitForTrainingAndTest(xall,yall,ratio = 0.3)
> xin<-xyall$inputsTrain
> yd<-xyall$targetsTrain
> xinteste<-xyall$inputsTest
> yteste<-xyall$targetsTest
> retlist<-treinaELM(xin,yd,2,1)
> w<-retlist[[1]]
> H<-retlist[[2]]
> Z<-retlist[[3]]
> yt<-YELM(xinteste,Z,w,1)
> acuracia<-1-(t(yteste-yt) %*% (yteste-yt) )/30
> print(acuracia)

     [,1]
[1,]    1
```

# Capítulo 6

# Redes RBF

## 6.1 Introdução

As redes neurais do tipo RBF (*Radial Basis Functions Neural Networks* - RBFNNs) são caracterizadas pela utilização de funções radiais nos neurônios de sua única camada intermediária, cujas respostas são combinadas linearmente para a obtenção da saída da rede. A sua estrutura se assemelha a uma rede neural tipo MLP, porém, ao invés de neurônios com funções sigmoidais na camada intermediária, a rede RBF utiliza neurônios com funções radiais. Assim, de maneira geral, uma rede RBF pode ser descrita formalmente conforme Equação 6.1 e representada de maneira esquemática na Figura 6.1.

$$f(\mathbf{x}, \theta) = \sum_{i=1}^{p} w_i h_i(\mathbf{x}, \mathbf{z}_i) + w_0 \tag{6.1}$$

em que $\mathbf{w} = [w_0, w_1, w_2, \cdots, w_p]$ é o vetor de parâmetros do neurônio de saída da rede, $\mathbf{z}_i$ é o vetor que contém todos os parâmetros da função de ativação radial do neurônio $i$ da camada intermediária, $\theta = [\mathbf{w}, \mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_p]$ é o vetor que contém a concatenação de todos os vetores de parâmetros da rede.

Para o caso de redes RBF com funções Normais, os vetores de parâmetros $\mathbf{z}_i$ das funções radiais contêm os pontos centrais de cada função e as covariâncias entre as variáveis de entrada. Assim, os parâmetros $\mathbf{z}_i$ que caracterizam cada função radial são, neste caso, a matriz de covariâncias $\Sigma_i$ e o vetor de médias $\mu_i$, conforme Equações 6.2 e 6.3.

$$\mathbf{\Sigma}_i = \begin{bmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \cdots & \rho_{1n}\sigma_1\sigma_n \\ \rho_{21}\sigma_2\sigma_1 & \sigma_2^2 & \cdots & \rho_{2n}\sigma_2\sigma_n \\ \vdots & \vdots & \cdots & \vdots \\ \rho_{n1}\sigma_n\sigma_1 & \rho_{n2}\sigma_n\sigma_2 & \cdots & \sigma_n^2 \end{bmatrix} \tag{6.2}$$

$$\mu_i = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} \tag{6.3}$$

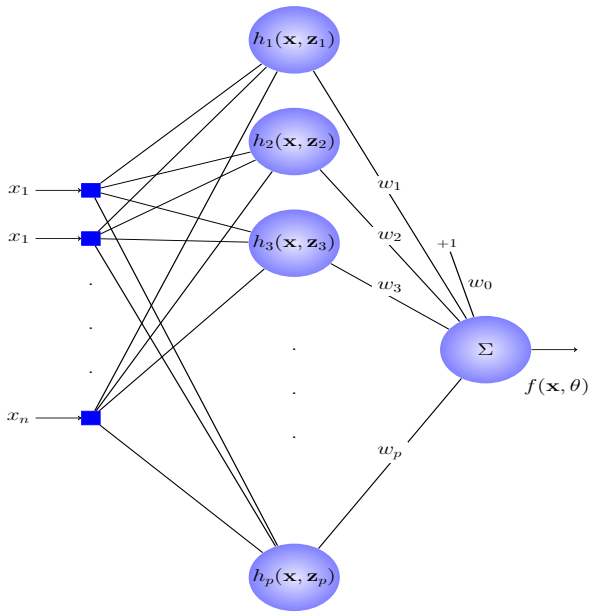



Figura 6.1: Representação esquemática de uma rede RBF.

O treinamento de redes RBF com funções de ativação normais envolve, portanto, encontrar as matrizes $\mathbf{\Sigma}_i$ e $\mu_i$ para cada uma das funções radiais e também o vetor de pesos $\mathbf{w}$ da função de saída. A expressão geral da função de densidade Normal de múltiplas variáveis para um neurônio arbitrário $i$ é apresentada da Equação 6.4.

$$h_i(\mathbf{x}, \mathbf{z}_i) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}_i|}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu_i)^{\mathrm{T}} \mathbf{\Sigma}_i^{-1} (\mathbf{x}-\mu_i)\right)} \tag{6.4}$$

A implementação das funções radiais segundo a Equação 6.4 requer a estimativa covariâncias de todos os pares de variáveis que definem o problema. No entanto, o problema pode ser simplificado quando é considerada independência entre as variáveis, o que leva $\mathbf{\Sigma}$ a se tornar diagonal. Para este caso particular, uma simplificação adicional pode ser feita considerando-se todos os desvios-padrão idênticos e iguais a $r$, chamado de raio da função. Assim, neste caso, o treinamento envolve encontrar o vetor $\mathbf{w}$, o raio $r$ e os centros $\mu_i = [\mu_1, \mu_2, \cdots, \mu_n]$ das funções radiais. Para este caso particular, as matrizes de covariâncias se reduzem a $\Sigma = \Sigma_1 = \cdots = \Sigma_p = r^2\mathbf{I}$ e a expressão geral das funções radiais pode ser descrita conforme Equação 6.5.

$$h_i(\mathbf{x}, \mathbf{z}_i) = \frac{1}{\sqrt{(2\pi)^n |r^2\mathbf{I}|}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu_i)^{\mathrm{T}} (r^2\mathbf{I})^{-1} (\mathbf{x}-\mu_i)\right)} \tag{6.5}$$

Considerando-se que o termo $\frac{1}{\sqrt{(2\pi)^n |r^2\mathbf{I}|}}$ é somente um fator de normalização comum a todas as funções e que não se deseja aqui estimar densidades, mas sim combinar funções não-lineares, ele pode ser desconsiderado, o que nos leva à Equação 6.6 como expressão final para as funções radiais.

$$h(\mathbf{x}, \mathbf{z}_i) = e^{-\frac{(\mathbf{x}-\mu_i)^2}{2r^2}} \tag{6.6}$$

Assim, do ponto de vista conceitual, o treinamento de redes RBF envolve a distribuição das funçoes radiais no espaço de entrada, a determinação de seus raios e a combinação linear de suas respostas. Como as funções radiais possuem um raio fixo $r$ que determina a sua abrangência, pode-se fazer a analogia destas funções com hiper-esferas. Assim, parte do problema de treinamento envolve a distribuição destas hiper-esferas no espaço de entrada, conforme representado esquematicamente na Figura 6.2 para um problema de duas variáveis.

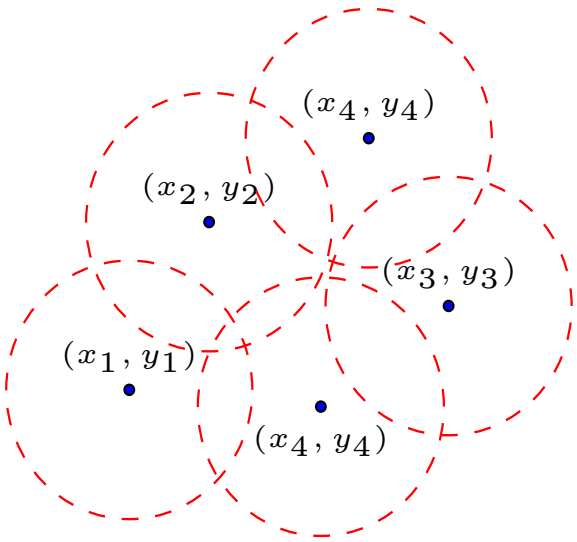



Figura 6.2: Representação esquemática das funções radiais de uma rede RBF como estruturas circulares distribuídas no espaço de entrada.

Uma vez determinados os parâmetros das funções radiais, define-se a projeção da matrix de dados $\boldsymbol{X}$ no espaço da camada intermediária por meio da matrix $\boldsymbol{H}$, apresentada na Equação 6.7. A saída do modelo RBF é obtida por meio da combinação linear destas projeções, conforme Equação 6.1, cujos coeficientes são os elementos do vetor de pesos $\mathbf{w}$, que também deve ser obtido como parte do aprendizado.

$$\mathbf{H} = \begin{bmatrix} h_1(\mathbf{x}_1, \mathbf{z}_1) & h_2(\mathbf{x}_1, \mathbf{z}_2) & \cdots & h_h(\mathbf{x}_1, \mathbf{z}_p) \\ h_1(\mathbf{x}_2, \mathbf{z}_1) & h_2(\mathbf{x}_2, \mathbf{z}_2) & \cdots & h_h(\mathbf{x}_2, \mathbf{z}_p) \\ \vdots & \vdots & \cdots & \vdots \\ h_1(\mathbf{x}_N, \mathbf{z}_1) & h_2(\mathbf{x}_N, \mathbf{z}_2) & \cdots & h_h(\mathbf{x}_N, \mathbf{z}_p) \end{bmatrix} \tag{6.7}$$

Uma vez obtida a matriz $\mathbf{H}$, a saída da rede RBF pode então ser calculada por meio da Equação 6.8.

$$\mathbf{Hw} = \mathbf{Y} \tag{6.8}$$

O vetor de pesos $\mathbf{w}$ pode então ser obtido por meio da resolução do sistema linear da Equação 6.8, cuja solução geral é apresentada na Equação 6.9.

$$\mathbf{w} = \mathbf{H}^{+}\mathbf{Y} \tag{6.9}$$

onde $\mathbf{H}^{+}$ é a pseudo-inversa de $\mathbf{H}$.

### 6.1.1 Exemplo simples de aproximação

Considere, por exemplo, o problema de aproximação da função $f(x) = seno(x)$ no intervalo de 0 a $2\pi$. Como neste momento o nosso objetivo é apenas mostrar a capacidade de aproximação das redes RBF, para efeitos deste exemplo, consideraremos a escolha dos centros por simples inspeção da função seno. Observa-se no gráfico da Figura 6.3 que parece ser razoável escolher os centros nos pontos $\mu_1 = \frac{\pi}{2}$ e $\mu_1 = \frac{3\pi}{2}$, já que a função tem forma radial em torno destes pontos. Assim, as expressões das duas funções radiais resultantes, em função do parâmetro $r$, são apresentadas nas Equações 6.10 e 6.11.

$$h_1(x, r) = e^{-\frac{(x-\frac{\pi}{2})^2}{2r^2}} \tag{6.10}$$

$$h_2(x, r) = e^{-\frac{(x-\frac{3\pi}{2})^2}{2r^2}} \tag{6.11}$$

O gráfico da Figura 6.3 mostra as duas funções $h_1(x, r)$ e $h_2(x, r)$ para $r = 0.5$, os pontos amostrados na função $f(x) = seno(x)$ e a função aproximadora resultante, considerando-se $w_1 = 1.3$, $w_2 = -1.3$ e $w_0 = 0$, o que resultou na função aproximadora $\hat{f}(x) = 1.3h_1(x, 0.5) - 1.3h_2(x, 0.5)$. A aproximação final no intervalo 0 a $2\pi$, obtida também por inspeção, é apresentada na Figura 6.3.

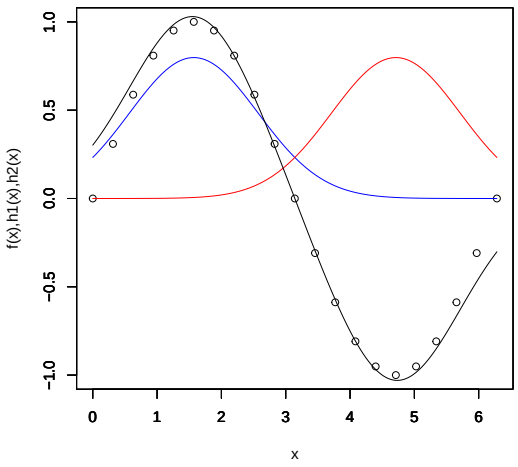



Figura 6.3: Exemplo de aproximação da função $f(x) = seno(x)$ por meio da combinação linear de duas funções radiais normais.

## 6.2 Alocação uniforme dos centros

No exemplo da Seção 6.1.1, os centros das funções radiais para aproximação da função $f(x) = seno(x)$ e os demais parâmetros da rede RBF foram alocados por inspeção. No exemplo desta seção, adotaremos a estratégia de distribuir os centros de maneira uniforme sobre o espaço de entrada, ou seja, sobre o domínio da variável $x$. O nosso objetivo será aproximar a função $f(x) = \frac{seno(x)}{x}$ no intervalo $[-10, 10]$. Para realizar a aproximação, os centros serão alocados uniformemente com raios fixos dentro dos limites de $x$, conforme mostrado na Figura 6.4, ou seja, as funções radiais estão centralizadas nos pontos $[-9, -7, -5, -3, -1, 1, 3, 5, 7, 9]$. Foi escolhido o valor arbitrário $r = 0.5$ para todas as funções radiais. Assim, com os valores dos centros e do raio definidos, pode-se obter a matriz $\mathbf{H}$ e então resolver a Equação 6.9 e realizar a aproximação com base na Equação 6.8. O resultado desta aproximação é apresentado na Figura 6.4.

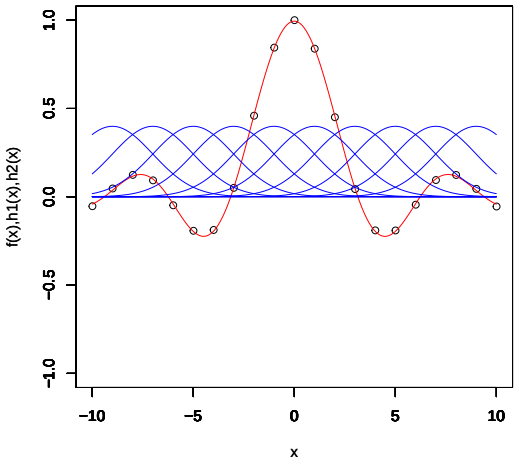



Figura 6.4: Aproximação da função $f(x) = \frac{seno(x)}{x}$ no intervalo $[-10, 10]$, por meio da superposição de 10 funções radiais normais.

## 6.3 Rotinas de Treinamento

```
1  treinaRBF<-function(xin, yin, p)
2  {
3
4    ######## Função radial Gaussiana
5    pdfnvar<-function(x,m,K,n)
6    {
7      if (n == 1)
8      {
9        r<-sqrt(as.numeric(K))
10       px<-(1/(sqrt(2*pi*r*r)))*exp(-0.5 * ((x-m)/(r))^2)
11     }
```

```
    else px<-((1/(sqrt((2*pi)^n*(det(K)))))*exp(-0.5*(t(x-m) %*% (
    solve(K)) %*% (x-m))))
    return(px)
  }
  ############################

  N<-dim(xin)[1] # número de amostras
  n<-dim(xin)[2] # dimensão de entrada (deve ser maior que 1)

  xin<-as.matrix(xin) # garante que xin seja matriz
  yin<-as.matrix(yin) # garante que yin seja matriz

  xclust<-kmeans(xin,p)

  # Armazena vetores de centros das funções.
  m<-as.matrix(xclust$centers)
  covlist<-list()

  # Estima matrizes de covariância para todos os centros.
  for (i in 1:p)
  {
    ici<-which(xclust$cluster == i)
    xci<-xin[ici,]
    if (n ==1)
      covi<-var(xci)
    else covi<-cov(xci)
    covlist[[i]]<-covi
  }

  H<-matrix(nrow = N, ncol = p)
  # Calcula matriz H
  for (j in 1:N)
  {
    for (i in 1:p)
    {
      mi<-m[i,]
      covi<-covlist[i]
      covi<-matrix(unlist(covlist[i]),ncol=n,byrow = T) + 0.001*
    diag(n)
      H[j,i]<-pdfnvar(xin[j,],mi,covi,n)
    }
  }

  Haug<-cbind(1,H)
  W<-(solve(t(Haug) %*% Haug) %*% t(Haug)) %*% yin


  return(list(m,covlist,W,H))
}
```

Listagem 6.1: Função de treinamento de uma rede RBF.

```
YRBF<-function(xin, modRBF)
{

  ######## Função radial Gaussiana
  pdfnvar<-function(x,m,K,n)
  {
    if (n == 1)
    {
      r<-sqrt(as.numeric(K))
      px<-(1/(sqrt(2*pi*r*r)))*exp(-0.5 * ((x-m)/(r))^2)
    }
```

```
12      else px<-((1/(sqrt((2*pi)^n*(det(K)))))*exp(-0.5*(t(x-m) %*% (
        solve(K)) %*% (x-m))))
13      return(px)
14    }
15    ############################
16
17    N<-dim(xin)[1] # número de amostras
18    n<-dim(xin)[2] # dimensão de entrada (deve ser maior que 1)
19    m<-as.matrix(modRBF[[1]])
20    covlist<-modRBF[[2]]
21    p<-length(covlist) # Número de funções radiais
22    W<-modRBF[[3]]
23
24    xin<-as.matrix(xin) # garante que xin seja matriz
25    # yin<-as.matrix(yin) # garante que yin seja matriz
26
27    H<-matrix(nrow = N, ncol = p)
28    # Calcula matriz H
29    for (j in 1:N)
30    {
31      for (i in 1:p)
32      {
33        mi<-m[i,]
34        covi<-covlist[i]
35        covi<-matrix(unlist(covlist[i]),ncol=n,byrow = T) +0.001*diag
        (n)
36        H[j,i]<-pdfnvar(xin[j,],mi,covi,n)
37      }
38    }
39
40    Haug<-cbind(1,H)
41    Yhat<-Haug %*% W
42
43    return(Yhat)
44  }
```

Listagem 6.2: Função que calcula a saída de uma rede RBF.

### 6.3.1 Aproximação de Funções

### 6.3.2 Conjunto de Dados da Iris

```
> rm(list = ls())
> source(file='/home/apbraga/PRINCIPAL/livros/LIVRO_R/rotinasGPL/treinaRBF.R')
> source(file='/home/apbraga/PRINCIPAL/livros/LIVRO_R/rotinasGPL/YRBF.R')
> data(iris)
> xseq<-sample(100)
> xall<-as.matrix(iris[xseq,1:4])
> yall<-as.matrix((1*(iris$Species[xseq] == 'versicolor')-0.5)*2)
> xin<-as.matrix(xall[1:70,])
> yd<-as.matrix(yall[1:70,])
> xinteste<-as.matrix(xall[(71:100),])
> yteste<-as.matrix(yall[(71:100),])
> modRBF<-treinaRBF(xin,yd,6)
> w<-modRBF[[3]]
> H<-modRBF[[4]]
> Yhat<-YRBF(xin,modRBF)
```

```
> rm(list = ls())
> source(file='/home/apbraga/PRINCIPAL/livros/LIVRO_R/rotinasGPL/treinaRBF.R')
> source(file='/home/apbraga/PRINCIPAL/livros/LIVRO_R/rotinasGPL/YRBF.R')
> xin<-matrix(seq(0,2*pi,0.01*pi),ncol=1)
> yin<-matrix(sin(xin),ncol=1)
> modRBF<-treinaRBF(xin,yin,2)
> xrange<-matrix(seq(0,2*pi,0.001*pi),ncol=1)
> yhat_teste<-YRBF(xrange,modRBF)
> plot(xin,yin,type='b',xlim=c(0,2*pi),ylim=c(-1,1),xlab="x",ylab="f(x),yhat")
> par(new=T)
> plot(xrange,yhat_teste,col='red',type='l',xlim=c(0,2*pi),ylim=c(-1,1),xlab='',ylab=
```

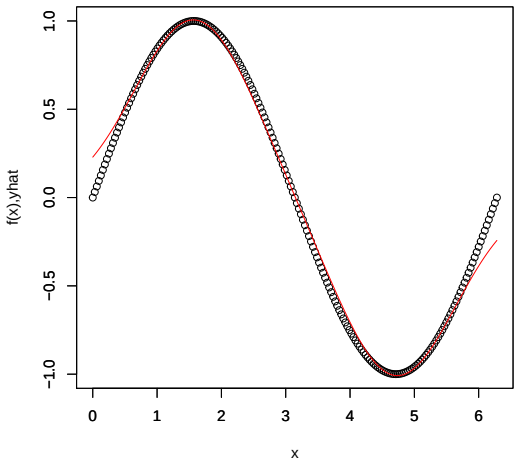



Figura 6.5: Aproximação da função $seno(x)$ por meio de redes RBF e seleção de centros usando o K-médias.

```
> yt<-(1*(Yhat >= 0) - 0.5)*2
> t(yd-yt) %*% (yd-yt)/(4*70)

           [,1]
[1,] 0.02857143

> sum(1*(yd!= yt))

[1] 2

> Yhat_teste<-YRBF(xinteste,modRBF)
> ytst<-(1*(Yhat_teste >= 0) - 0.5)*2
> t(yteste-ytst) %*% (yteste-ytst)/(4*30)

     [,1]
[1,]  0.2
```

```
> sum(1*(yteste != ytst))

[1] 6

>
```

# Referências Bibliográficas


[And36] E. Anderson. The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457–509, 1936.

[And68] J.A. Anderson. A memory model using spatial correlation functions. *Kybernetik*, 5:113–119, 1968.

[And70] J.A. Anderson. Two models for memory organization. *Mathematical Biosciences*, 8:137–160, 1970.

[And95] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[BCL07] Antônio Braga, André C. Carvalho, and Teresa B. Ludermir. *Redes Neurais Artificiais: Teoria e aplicações*. LTC Editora, 2007.

[BGV92] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

[BL88] D. S. Broomhead and D. Lowe. Multivariable function interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[BL13] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[Bra94] A. P. Braga. Predicting contradictions in the storage process of diluted recurrent boolean neural networks. *Electronics Letters*, 30:55–56, 1994.

[Cov65] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334, 1965.

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[CV95] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–279, 1995.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoid function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[DHS01] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley and Sons, 2001. 0-471-05669-3.

[Fis36] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[Heb49] D. O. Hebb. *The Organization of Behavior*. Wiley, 1949.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective properties. *Proc. Nat. Acad. Sci.*, 79:2554–8, 1982.

[HZS04] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990 vol.2, july 2004.

[Jam85] WILLIAM James. Psychology: The briefer course (g. allport, ed.). *Notre Dame, IN: University of Notre Dame Press.(publicado originalmente em 1892)*, 1985.

[KM15] S. K. Khaitan and J. D. McCalley. Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, 9(2):350–365, June 2015.

[Koh74] T. Kohonen. An adaptive associative memory principle. *IEEE Transactions on Computers*, C-23:444–445, 1974.

[Koh78] Zvi Kohavi. *Switching and finite automata theory*. McGraw Hill, 1978.

[Kos88] B. Kosko. Bidirectional associative memory. *IEEE Trans. SMC*, 18:49–60, 1988.

[Leo94] S. J. Leon. *Linear algebra with applications*. Prentice Hall, 1994.

[Lu17] Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1 – 10, 2017.

[LXZ+17] C. Lv, Y. Xing, J. Zhang, X. Na, Y. Li, T. Liu, D. Cao, and F. Y. Wang. Levenberg-marquardt backpropagation training of multilayer neural networks for state estimation of a safety critical cyberphysical system. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2017.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[MdCT06] P.A. Morettin and C.M. de Castro Toloi. *Análise de séries temporais*. ABE - Projeto Fisher. Edgard Blucher, 2006.

[MP43] W.S. McCulloch and W. Pitts. A logical calulus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MPRV87] R.J. McEliece, E.C. Posner, E.R. Rodemich, and S.S. Venkatesh. The capacity of the hopfield associative memory. *IEEE Transactions on Information Theory*, 33:461–482, 1987.

[NW06a] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[NW06b] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[OS75] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice–Hall, 1975.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65:386–408, 1958.

[SCAN16] A. Sargolzaei, C. D. Crane, A. Abbaspour, and S. Noei. A machine learning approach for fault detection in vehicular cyber-physical systems. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 636–640, Dec 2016.

[SGLW08] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang. Cyber-physical systems: A new frontier. In *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pages 1–9, June 2008.

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[WH60] B. Widrow and M.E. Hoff. Adaptative switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention*, 1960.

[Wol09] W. Wolf. Cyber-physical systems. *Computer*, 42(3):88–89, March 2009.

[WSZ+17] Y. Wang, B. Song, P. Zhang, N. Xin, and G. Cao. A fast feature fusion algorithm in image classification for cyber physical systems. *IEEE Access*, 5:9089–9098, 2017.

[WTH+17] J. Wang, W. Tu, L. C. K. Hui, S. M. Yiu, and E. K. Wang. Detecting time synchronization attacks in cyber-physical systems with machine learning techniques. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2246–2251, June 2017.

[XHL14] L. D. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014.