Roberto S. Bigonha

Fundamentos do Aprendizado de Máquina

1a Edição

Belo Horizonte

2025

1a Edição

Fundamentos do Aprendizado de Máquina

1^a Edição

Roberto S. Bigonha

2025

Belo Horizonte, MG

Roberto S. Bigonha: Pequisador Independente. PhD em Ciência da Computação pela Universidade da Califórnia, Los Angeles. Professor Titular Emérito da Universidade Federal Minas Gerais. Membro da Sociedade Brasileira de Computação. Áreas de interesse: Linguagens de Programação, Programação Modular, Estruturas de Dados, Compiladores, Semântica Formal.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

B594	Bigonha, Roberto S. Fundamentos do Aprendizado de Máquina / Roberto da Silva Bigonha — Belo Horizonte, MG, 2025, 100 p Bibliografia. ISBN 978-65-01-28912-0 1.Inteligência Artificial 2. Raciocínio Indutivo I. Título. CDD: 005.1 CDU 004.43

Índice para catálogo sistemático:
1. Inteligência Artificial

Última Atualizão: 16:08 de 16 de janeiro de 2025

Prefácio

Inteligência Artificial (IA) é a automação de atividades normalmente associadas a raciocínio, tais como resolver problemas, tomar decisões e aprender.

Na década de 1960, IA era uma área da Computação que buscava resolver problemas considerados difíceis para seres humanos, mas que poderiam ser relativamente mais fáceis para computadores.

Atualmente, o grande desafio de IA é automatizar a execução de tarefas que podem ser facilmente resolvidas por humanos, mas cujas soluções genéricas ainda são difíceis de ser descritas formalmente.

A solução desses problemas, e.g., reconhecimento de imagem, é difícil de ser implementada pelas técnicas tradicionais.

Este pequeno livro aborda as técnicas de treinamento de algoritmos que fundamentam o aprendizado de máquina baseado em redes neurais densas.

Agradecimentos

Agradeço aos colegas que fizeram a leitura dos primeiros manuscritos, apontaram erros, indicaram correções e também contribuíram com exemplos.

Particularmente, agradeço à professora Mariza Andrade da Silva Bigonha o cuidadoso trabalho de revisão do texto e ao professor Ivan Moura Campos, suas importantes observações para melhorar o texto.

Roberto S. Bigonha

Sumário

Capítulo 1

Inteligência Artificial

Inteligência Artificial é a automação de atividades normalmente associadas a raciocínio, tais como resolver problemas, tomar decisões e aprender.

Na década de 1960, IA era uma área da Computação que buscava resolver problemas considerados difíceis para seres humanos, mas que poderiam ser relativamente mais fáceis para computadores.

Atualmente, o grande desafio de IA é automatizar a execução de tarefas que podem ser facilmente resolvidas por humanos, mas cujas soluções genéricas ainda são difíceis de ser descritas formalmente. A solução desses problemas, e.g., reconhecimento de imagem, é difícil de ser implementada pelas técnicas tradicionais.

As principais áreas de pesquisa em IA são: Aprendizado de Máquina (*Machine Learning*), Sistemas Especialistas (*Expert Systems*), Visão Computacional (*Computer Vision*), Processamento de Linguagem Natural (*Natural Language Processing*) e Robótica (*Robotics*).

Todas essas áreas têm um papel muito importante

no desenvolvimento da Ciência da Computação moderna, mas, por uma questão prática, o foco deste livro é Aprendizado de Máquina.

1.1 Aprendizado de Máquina

Aprendizado de Máquina é uma técnica de programação de computadores baseada na detecção automática de padrões nos dados de um problema e à indução de conclusões a seu respeito.

Os algoritmos de aprendizado produzem programas a partir de exemplos que especificam as saídas ou *labels* para certas entradas ou *features*. A partir dos padrões identificados entre entrada e saída dos exemplos, produzem-se programas para interpretar novas entradas. Assim, esses programas são capazes de informar a saída mais provável para uma entrada que esteja ou não nos exemplos dados.

Essencialmente, Aprendizado de Máquina é a ciência de tornar máquinas capazes de tomar decisões sem a intervenção humana. E a ideia principal é capacitar computadores a aprender e interpretar padrões em imagens, sons e outros dados estruturados a partir de padrões identificados em exemplos similares.

Os programas de computadores que simulam tomadas de decisão precisam ser treinados, via a definição de seus parâmetros, os quais governam seu funcionamento.

1.2 Um Método Simples de Treinamento

Um método muito simples de treinamento de um programa consiste em, por exemplo, dado o conjunto de pares para treinamento:

$$\{(x_i, y_i), \textbf{ para } 1 \leq i \leq N_a\}$$

onde x_i é chamado de atributo (*feature*) e y_i, de resultado (*label*), deseja-se treinar um programa para que dado um novo x ele seja capaz de sugerir um y, tal que $y = h(x)$, para alguma função h.

Em um treinamento bem simples, pode-se supor que h seja definido por $h(x) = mx+b$, e o problema central é a determinação dos parâmetros m e b, a partir da amostra de pares (x, y) dada, ou seja, graficamente:

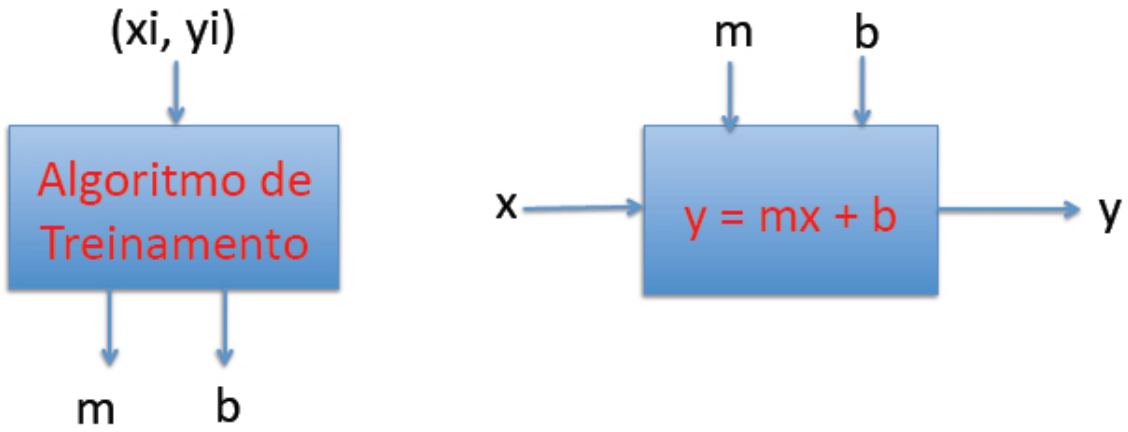


Figura 1.1: Modelo de Aprendizado

de forma que o mapeamento de valores de x a y possa ser inferido pelo algoritmo treinado como mostrado no diagrama à direita na Fig. 1.1.

Quando y é um conjunto contínuo de valores, esse método de treinamento é chamado de regressão. E quando y é discreto, tem-se uma regressão logística.

Esse processo é denominado **Aprendizado de Máquina**.

1.3 Tipos de Aprendizado de Máquina

Aprendizado de Máquina pode ser subdividido em quatro tipos[1]:

- Supervised Learning: dados um arranjo de atributos e resultados associados, treina-se o agente para predizer a melhor saída para um novo atributo.
- Unsupervised Learning: dado um arranjo de atributos, sem os respectivos resultados, descobrem-se padrões dentro desse arranjo.
- Semi-Supervised: dados um arranjo de atributos e uma quantidade limitada de resultados associados, treina-se o agente para predizer resultados para os demais atributos.
- Reinforcement Learning: dado um objetivo, treina-se um agente artificial, via tentativa e erro, sem o auxílio de operador humano.

E as técnicas para Aprendizado de Máquina comumente usadas são assim denominadas:

[1]Nomes em inglês ligados à área de IA não foram algumas vezes aqui traduzidos para facilitar sua identificação na literatura científica.

- Linear Regression
- Dense Neural Networks
- Support Vector Machines
- Naive Bayes
- Decision Trees
- Convolutional Networks
- Recurrent Neural Networks
- Attention Networks
- Bidirectional Transformers

As palavras-chave em inglês mais recorrentes e ligadas à área de aprendizado de máquina e que estão definidas ao longo deste texto são:

- backpropagation
- bias
- convolution
- cross-validation
- deep learning
- dense
- forward propagation
- generalization
- gradient descent
- inference
- induction
- least squares
- linear regression
- machine learning
- mean squared error
- naive bayes
- neural network
- overfitting
- perceptron
- regularization
- ReLU
- sigmoid
- softmax
- training
- underfitting
- weight

Leitura Recomentada

- Hal Daumé III, A Course in Machine Learning, disponível na Internet, Version 0.8, 189 páginas, August 2012.

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press, 799 páginas, 2016.
- Kevin P. Murphy, Machine Learning - A Probabilistic Perspective, The MIT Press, Cambridge-Massachusetts, London-England, 1098 páginas, 2012 .
- Michael A. Nielsen, Neural Networks and Deep Learning, on-line book, 395 páginas, 2015.
- Andrew Ng, Machine Learning Yearning - draft, 118 páginas, free book, 2018.
- Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, 946 páginas, Prentice Hall, Englewood Cliffs, New Jersey, 1995.
- Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, Dive Into Deep Learning, pdf Release 0.7.1, 912 páginas, jan 25, 2020.

***

Capítulo 2

Silogismo

Segundo o grego Aristóteles (Aristóteles-300AC), silogismo é uma forma de raciocínio que consiste em duas proposições: a primeira, chamada premissa maior, a segunda, premissa menor, e a conclusão. Pelo silogismo, admitida a coerência das premissas, a conclusão se infere da maior por intermédio da menor. Por exemplo:

- premissa maior: todo ser humano é mortal
- premissa menor: eu sou um ser humano
- conclusão: eu sou mortal

Esse processo de raciocínio é chamado de inferência, ou seja, inferência são passos do raciocínio que mapeia premissa em consequência lógica e pode ser classificada em dedução ou indução.

No caso de dedução, conclusões lógicas verdadeiras são derivadas de premissas conhecidas e verdadeiras, usando regras de inferência válidas. Nesse processo, o raciocínio parte do geral para derivar um resultado mais específico.

Em dedução, a conclusão deve ser aceita com resultado derivado das premissas via regras formais da Lógica. Assim, o método de inferência usado tem base científica sólida e garante a validade da conclusão. Portanto, a conclusão é assim um fato comprovado logicamente.

Por outro lado, com indução, as conclusões são derivadas de premissas particulares, não necessariamente demonstradas verdadeiras. Nessa inferência, o raciocínio parte do específico para derivar um resultado mais geral, sendo portanto uma generalização.

Dado que neste caso a conclusão deriva de premissas empiricamente, não há garantia de que seja verdadeira em todos os casos. Isto é, a conclusão não é logicamente válida, há apenas uma probabilidade que seja correta para a maioria dos casos. Quanto mais casos analisados, maior é essa probabilidade, e a validade em todos os casos possíveis exige que se tenha informação sobre todos eles.

2.1 Exemplo de Dedução X Indução

No caso de dedução, parte-se de uma premissa maior, mais geral, e.g., de que todos os seres humanos são mortais, e dado que um indivíduo chamado Aristóteles é um ser humano, então infere-se um resultado específico que diz que Aristóteles é mortal.

No caso de indução, a premissa maior é um dado específico, e.g., o citado Aristóteles morreu, e dado a premissa menor de que Aristóteles era um ser humano, induz-se um resultado mais geral, sem o devido fundamento, que diz que todos os seres humanos seriam mortais.

2.2 Dedução X Indução X IA

O mecanismo de inferência em Inteligência Artificial pode ser dedutivo ou indutivo.

Em um processo dedutivo, o programa de inferência implementa um raciocínio dentro de uma dada teoria com base em regras de inferência da lógica matemática e deriva uma conclusão específica.

Por exemplo, pode-se definir um conjunto de axiomas da forma:

$$\text{pai(antônio, josé)} \wedge \text{pai(josé, pedro)} \wedge$$
$$\text{pai(josé, joão)} \wedge ...$$

que define que antônio é pai de josé, josé é pai de pedro e de joão, etc.

E além desses axiomas, definem-se regras de inferência do tipo:

$$\text{pai(A,B)} \wedge \text{pai(B,C)} \Longrightarrow \text{neto(C,A)}$$

para identificar precisamente quem é neto de quem.

Esse mecanismo é de fato matematicamente preciso, mas demanda implementações específicas e de genera-

lização difícil.

Com um processo indutivo, parte-se de um conhecimento específico e busca uma conclusão geral, por meio de treinamento do algoritmo a partir de uma massa de dados, e.g., $\{(x_i, y_i), \textbf{para } 1 \leq i \leq N_a\}$ que relaciona entradas e resultados específicos.

A premissa maior, nesse caso, é um conhecimento específico dado por uma tabela como a da Fig. 2.1, que relaciona valores de x a de y, para $1 \leq x \leq 22$. E a premissa menor é que há uma reta $y = mx + b$ que generaliza esse mapeamento.

A partir dessas premissas, conclui-se que o melhor modelo, usando o método dos Mínimos Quadrados, é a reta $y = mx + b$, para $m = 23$ e $b = -91$.

x	y	mx+b	x	y	mx+b
1	2	-68,0	12	145	185,0
2	5	-45,0	13	170	208,0
3	10	-22,0	14	197	231,0
4	17	1,0	15	226	254,0
5	26	24,0	16	257	277,0
6	37	47,0	17	290	300,0
7	50	70,0	18	325	323,0
8	65	93,0	19	362	346,0
9	82	116,0	20	401	369,0
10	101	139,0	21	442	392,0
11	122	162,0	22	485	415,0

Figura 2.1: indução

Ou seja, a partir da premissa maior, e.g., a tabela (x, y) da Fig. 2.1, e por intermédio da premissa menor de que o mapeamento de x a y seja linear, induz-se

a conclusão de que esse mapeamento é modelado pela reta $y = 23x - 91$, para todo x.

Observe que essa conclusão depende da validade das premissas nas quais o raciocínio foi baseado, e que o mapeamento de x a y é válido somente para os valores da tabela dada. Para outros valores de x, apenas existe uma probabilidade de que o mapeamento seja correto, e essa probabilidade será maior quanto maior for a tabela (x, y), mas, nesse caso, nunca chegará a 100%, pois a tabela foi, a título de exemplo, construída com o mapeamento $y = x^2 + 1$.

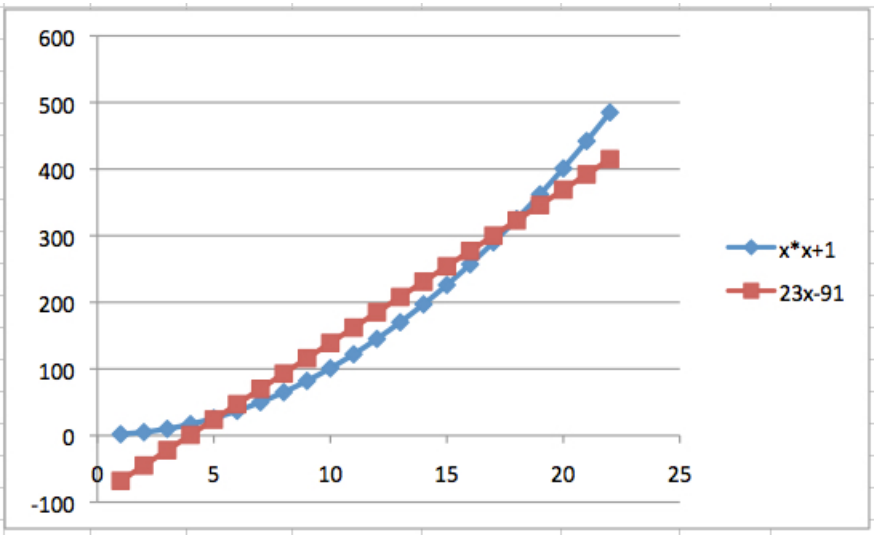


Figura 2.2: Aprendizado

Em termos do Aprendizado de Máquina, o processo de determinação dos valores de m e b de uma equação como $y = mx + b$ é chamado de treinamento. No exemplo acima, baseando-se na tabela dada, o algoritmo de treinamento induziu a função $y = 23x - 91$, para ma-

peiar novos x aos y correspondentes.

A premissa maior do método para derivação do modelo é um fato específico representado por uma amostra (x_i, y_i), $1 \leq i \leq N_a$, que define um mapeamento particular de entradas (*features*) x_i a resultados (*labels*) y_i.

A premissa menor, aceita como verdadeira, embora sem provas, é a existência de uma função $y = f(x)$ que mapeia *features* a *labels*. Essa função, $y = f(x)$, pode ser uma reta ou uma função mais complexa, como polinômios de grau $n > 1$.

A partir da premissa maior e por intermédio da menor, usando um método de treinamento, seja ele uma regressão, um processo de descida pelo gradiente usando *back-propagation* ou qualquer outra técnica, infere-se a expressão da função geral $y = f(x)$ que permite informar o resultado associado a cada entrada, dentro ou fora da amostra.

De acordo com Aristóteles, trata-se de uma inferência por indução.

Leitura Recomentada

- Hal Daumé III, A Course in Machine Learning, disponível na Internet, Version 0.8, 189 páginas, August 2012.
- Stuart J. Russell and Peter Norvig, Artificial In-

telligence: A Modern Approach, 4th Edition, 946 páginas, Prentice Hall, Englewood Cliffs, New Jersey, 1995.

- Wikipedia, Silogismo, https://pt.wikipedia.org/wiki/Silogismo, acessado em 2024.

***

Capítulo 3

Revisão Matemática

Nos capítulos a seguir, a fundamentação dos algoritmos que implementam o Aprendizado de Máquina são apresentados e muitos termos técnicos ligados à Matemática são usados. Assim, de forma a uniformizar o seu entendimento, a definição dos termos mais importantes é aqui relembrada.

3.1 Vetores e Matrizes

Um **escalar** é um número, representado por uma letra minúscula, como em $s \in \mathbb{R}$ para escalares reais, ou $n \in \mathbb{N}$ para escalares naturais.

Um **vetor** é um arranjo unidimensional de escalares, organizados em uma certa ordem, podendo seus elementos serem identificados pelo nome do vetor e o índice de sua posição dentro do vetor. Por exemplo:

$$x_i \in \mathbb{N}, 1 \leq i \leq n,$$

denota o i-ésimo elemento do vetor $x \in \mathbb{N}^n$.

Uma **matriz** é um arranjo bidimensional de escala-

res, identificado por uma letra maiúscula, sendo cada elemento identificado pelo nome da matriz com dois índices. Por exemplo:

$$A_{i,j} \in \mathbb{R}, \textbf{ para } 1 \leq i \leq m \textbf{ e } 1 \leq j \leq n$$

denota o elemento da linha i e coluna j da matriz $A \in \mathbb{R}^{mXn}$.

3.2 Tensores

Um **tensor** é um arranjo de escalares com mais de duas dimensões, sendo seus elementos indexados conforme cada dimensão. Por exemplo, $T_{i,j,k} \in \mathbb{N}$ denota o elemento de coordenadas $1 \leq i \leq m$, $1 \leq j \leq n$ e $1 \leq k \leq p$ do tensor $T \in \mathbb{N}^{mXnXp}$.

Um **produto escalar**, **produto interno** ou **dot product** $\odot$ é uma soma dos produtos dos elementos correspondentes de dois vetores de mesmo tamanho. Por exemplo, $a \odot b = \sum_{i=1}^{n} a_i b_i$, **para** $a, b : \mathbb{R}^n$.

3.3 Mean Squared Error

A *Mean Squared Error* (MSE) é uma medida da grandeza do erro entre os valores estimados (z_i) e os valores esperados (y_i), sendo medida como o valor médio dos quadrados dos erros observados:

$$\text{MSE} = \frac{1}{N_a} \sum_{i=1}^{N_a} (z_i - y_i)^2$$

MSE é sempre valor positivo, pois é a média dos quadrados dos erros e trata-se de uma medida da qualidade da estimativa, quanto menor melhor.

3.4 Regra da Cadeia do Cálculo

Suponha que $u = f(x_1, x_2, \cdots, x_n)$ seja uma função diferenciável, onde cada $x_j = g_j(t_1, t_2, \cdots, t_m)$, para $j : 1..n$, também seja uma função diferenciável. Então pela regra da cadeia:

$$\frac{\partial u}{\partial t_i} = \frac{\partial u}{\partial x_1}\frac{\partial x_1}{\partial t_i} + \frac{\partial u}{\partial x_2}\frac{\partial x_2}{\partial t_i} + \cdots + \frac{\partial u}{\partial x_n}\frac{\partial x_n}{\partial t_i}, \textbf{ para } i : 1..m$$

3.5 Gradiente ∇

Suponha que $u = f(x_1, x_2, \cdots, x_n)$ seja uma função diferenciável, então:

$$\nabla f(x_1, x_2, \cdots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$$

3.6 Distribuição de Probabilidades

Distribuição de probabilidades é uma descrição dos valores possíveis de uma variável aleatória. A forma de descrição da distribuição depende de as variáveis aleatórias serem discretas ou contínuas.

Uma função de probabilidade é uma função que associa a cada possível ocorrência de uma variável aleatória

uma probabilidade. Essa função pode ser:

- função massa de probabilidade $P(x)$
- função densidade de probabilidade $p(x)$

A função $P(x)$ de massa de probabilidade descreve a distribuição de variáveis aleatórias discretas e tem as seguintes propriedades:

1. Domínio de P é o conjunto X dos estados possíveis de x
2. $\forall x \in X,\ 0 \leq P(x) \leq 1$
3. $\sum_{x \in X} P(x) = 1$

A função $p(x)$ de densidade de probabilidade descreve a distribuição de variáveis aleatórias contínuas e possui as seguintes propriedades:

1. Domínio de p é o conjunto de todos os estados possíveis de x
2. $\forall x \in X, p(x) \geq 0$
3. Não se requer que $p(x) \leq 1$
4. $\int p(x)dx = 1$
5. A probabilidade $p(x)$ de x estar no intervalo $[a, b]$ é dada por $\int_a^b p(x)dx$.

3.7 Conceito de Convolução

Em Matemática, convolução é uma operação que combina duas funções, permitindo medir a soma do produto de uma função por outra deslocada e invertida. A convolução pode ser discreta ou contínua de acordo com o domínio das duas funções envolvidas. Representa-se por $(f * g)(t)$ a convolução das funções f e g.

Se $f, g \in [0, n-1] \to \mathbb{R}$, $h(t) = (f * g)(t)$ representa a convolução discreta de f e g, tal que

$$h(t) = \sum_{i=0}^{t} f(i)g(t-i), \textbf{ para } t \in \mathbb{Z} \textbf{ e } t : 0..n-1$$

Tipicamente, tem-se $h(t) =$

$$f(0)g(t)+f(1)g(t-1)+\cdots+f(t-1)g(1)+f(t)g(0)$$

ou seja, tem-se um somatório em que os elementos de f são multiplicados pelos de g deslocados e invertidos.

No caso de $f, g \in \mathbb{R} \to \mathbb{R}$ serem funções integráveis, $h(x) = (f * g)(x)$ representa a convolução contínua de f e g, tal que

$$h(x) = \int_{-\infty}^{\infty} f(u)g(x-u)du, \textbf{ para } x, u \in \mathbb{Z}.$$

Para exemplificar uma aplicação de convolução, seja $p(t)$, $p : \mathbb{Z} \to \mathbb{R}$, a posição de uma espaçonave no instante t, fornecida por um sensor a laser e supostamente o sensor não é muito preciso devido a ruídos.

Uma estimativa menos ruidosa da posição da espaçonave no momento t seria a média ponderada das t

medidas mais recentes:

$$s(t) = (\sum_{a=1}^{t} p(a)w(t-a))/(\sum_{a=1}^{t} w(a))$$

onde os pesos $w : \mathbb{N} \to \mathbb{R}$ são definidos de forma que

$$\sum_{a=1}^{t} w(a) = 1$$

Assim, a posição estimada da espaçonave é dada pela convolução de p e w:

$$s(t) = \sum_{a=1}^{t} p(a)w(t-a).$$

Leitura Recomentada

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press, 799 páginas, 2016.
- Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, Dive Into Deep Learning, pdf Release 0.7.1, 912 páginas, jan 25, 2020.

***

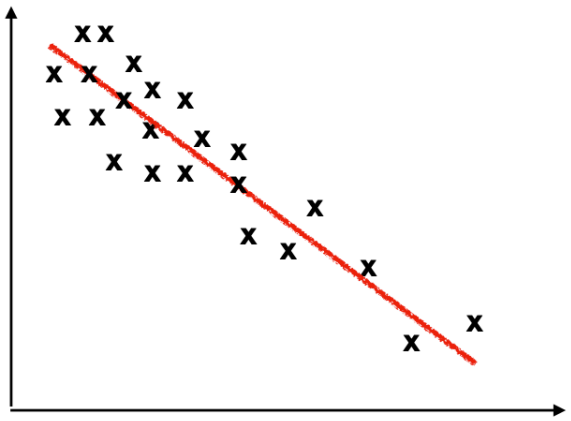
Capítulo 4

Descida pelo Gradiente

Um exemplo de treinamento de um algoritmo de IA é feito por meio do uso de regressão linear, a qual parte de uma amostra de dados, como uma tabela do tipo

$$\{(x_i, y_i), \textbf{para } 1 \leq i \leq N_a\}$$

e computa a equação linear $y = m * x + b$, que relaciona a entrada x ao resultado y para algum m e b, ou seja, a regressão interpola uma reta aos pontos da amostra dada. Graficamente:



Essa interpolação pode ser obtida, algébrica ou iterativamente, via o método dos mínimos quadrados[1].

[1] do inglês *Least Square Method*

4.1 Mínimos Quadrados

O método dos mínimos quadrados, definido por Gauss (1795) e Legendre (1805), consiste nos seguintes passos:

- Seleção da amostra para treinamento:
  $$\{(x_i, y_i) \textbf{ para } 1 \leq i \leq N_a\}$$
- Definição da forma da função a ser ajustada à amostra, que para simplificar este texto será usada a reta
  $$h(x) = mx + b.$$
- Equacionamento do indicador C de erro de ajustamento (MSE) em relação à amostra definido por:
  $$C(m, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} (h(x_i) - y_i)^2$$
  que no caso de ajuste de reta, torna-se:
  $$C(m, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} (mx_i + b - y_i)^2$$
- Computação dos pesos m e b usados para definir custo C mínimo, os quais podem ser determinados analítica ou iterativamente.

4.2 Determinação Analítica dos Pesos

Na formulação analítica, o erro C de ajustamento de uma reta em relação à amostra é dado por:

$$C(m, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} (mx_i + b - y_i)^2$$

e os pesos m e b devem ser calculados para definir o valor mínimo do erro de ajustamento C, conforme as

derivadas:

$$\frac{\partial C(m,b)}{\partial m} = \frac{1}{N_a}\sum_{i=1}^{N_a}(mx_i + b - y_i)x_i = 0$$

$$\implies \boxed{m = \frac{(\sum_{i=1}^{N_a} x_i)(\sum_{i=1}^{N_a} y_i) - N_a(\sum_{i=1}^{N_a} x_i y_i)}{(\sum_{i=1}^{N_a} x_i)^2 - N_a(\sum_{i=1}^{N_a} x_i^2)}}$$

$$\frac{\partial C(m,b)}{\partial b} = \frac{1}{N_a}\sum_{i=1}^{N_a}(mx_i + b - y_i) = 0$$

$$\implies \boxed{b = \frac{(\sum_{i=1}^{N_a} y_i) - m(\sum_{i=1}^{N_a} x_i)}{N_a}}$$

O método dos Mínimos Quadrados é muito conveniente para obter a regressão linear $y = mx + b$, quando temos apenas uma variável independente, mas no caso geral, de ajuste de grandes masssas de dados e de uso de funções de ajuste mais elaboradas que a reta, os cálculos indicados podem ser demasiadamente elaborados.

Mesmo quando tem-se $y = \sum_{i=1}^{n} mx_i + b$, mas n é muito grande, pode ser impraticável a obtenção das fórmulas-fechada de m, e o custo das derivadas da função C pode ser superior a $O(n^3)$.

Nesses casos, uma solução iterativa, como o Gradiente Descendente, pode ser computacionalmente muito mais simples.

Na descrição do Gradiente Descendente apresentada a seguir, usa-se apenas uma única variável independente de forma a facilitar o entendimento do processo. A

generalização para n variáveis independentes será vista à frente.

4.3 Gradiente Descendente com uma Variável

O método Gradiente Descendente foi inventado por Augustin Cauchy em 1847 com o objetivo de definir uma forma para achar os valores de $x_1, \ldots, x_n$ que satisfazem a equação $u = 0$, quando $u = f(x_1, \ldots, x_n)$.

Gradiente é uma uma grandeza vetorial que representa a inclinação da tangente do gráfico da função considerada e **dá a direção de sua maior taxa de variação**. Gradiente ∇u é um vetor com as derivadas de u em relação a x_i. Assim, se $u = f(x_1, x_2, \cdots, x_n)$ for uma função diferenciável, então

$$\nabla f(x_1, x_2, \cdots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$$

Na formulação de Cauchy, u não precisa ser uma função linear.

No caso de gradiente descendente com apenas uma variável independente, parte-se da amostra para treinamento $\{(x_i, y_i), \textbf{para } 1 \leq i \leq N_a\}$ e arbitram-se aleatoriamente valores iniciais para m e b da equação $h(x) = mx+b$, e para seus passos de decréscimo $\alpha > 0$.

Formula-se a equação do custo C pelo método dos mínimos quadrados:

$$C(m, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} (h(x_i) - y_i)^2$$

E determinam-se suas derivadas:

$\frac{\partial C}{\partial m} = \frac{1}{N_a}\sum_{i=1}^{N_a}(mx_i + b - y_i)x_i$

$\frac{\partial C}{\partial b} = \frac{1}{N_a}\sum_{i=1}^{N_a}(mx_i + b - y_i)$

Com base nas equações de custo de C e de suas derivadas, repita os passos abaixo enquanto o custo C estiver reduzindo-se, para calcular os valores de m e b que minimizam esse custo:

1. calcule o gradiente $\nabla C(m, b) = (\frac{\partial C}{\partial m}, \frac{\partial C}{\partial b})$, para toda a amostra (x_i, y_i), $1 \leq i \leq N_a$, e valores correntes de m e b
2. atualize $m := m - \alpha\frac{\partial C}{\partial m}$ **e** $b := b - \alpha\frac{\partial C}{\partial b}$
3. recalcule o custo $C(m, b)$ e compare com o anterior.

Há duas variantes importantes desse método iterativo para determinar os parâmetros m e b que minimizam a função de custo C:

- Modo *Batch*: quando usa-se toda a amostra em cada ajuste de m e b.
- Modo Estocástico: quando particiona-se a amostra em subamostras, denominadas *mini-batch*, e fazem-se ajustes de m e b para cada subamostra em separado.

4.4 Formulação do Modo *Batch*

A partir da amostra para treinamento:

$A = \{(x_i, y_i),$ **para** $1 \leq i \leq N_a\}$,

no modo *batch*, a determinação dos parâmetros m e b também baseia-se na seguinte formulação:

- Função de custo para toda a amostra A:
  $C(m,b) = \frac{1}{2N_a}\sum_{i=1}^{N_a}(h(x_i) - y_i)^2$,
  onde $h(x_i) = mx_i + b$
- Gradiente da função de custo:
  $\nabla C(m,b) = (\frac{\partial C}{\partial m}, \frac{\partial C}{\partial b})$
  onde $\frac{\partial C}{\partial m} = \frac{1}{N_a}\sum_{i=1}^{N_a}(mx_i + b - y_i)x_i$
  $\frac{\partial C}{\partial b} = \frac{1}{N_a}\sum_{i=1}^{N_a}(mx_i + b - y_i)$

E o algoritmo de descida do gradiente nesse caso:

- arbitre valores para m, b e α
- calcule o custo inicial $C(m,b)$ de toda a amostra A
- repita os passos a seguir até atingir convergência, i.e., até a função de custo C não mais apresentar melhoras significativas:
  - calcule o gradiente $\nabla C(m,b) = (\frac{\partial C}{\partial m}, \frac{\partial C}{\partial b})$, com os valores correntes de m e b
  - atualize $m := m - \alpha\frac{\partial C}{\partial m}$ **e** $b := b - \alpha\frac{\partial C}{\partial b}$
  - recalcule o custo $C(m,b)$ e compare com o valor anterior

Esse método é considerado muito caro, pois o custo total é reavaliado a cada ajuste de m e b e sempre considera toda a amostra. É preferível o modo estocástico que realiza menos cálculos.

4.5 Formulação no Modo Estocástico

Considerando a amostra de treinamento:

$$A = \{(x_i, y_i), \textbf{ para } 1 \leq i \leq N_a\}$$

e dada a função de custo para toda a amostra A:

$$C(m, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} (h(x_i) - y_i)^2,$$
$$\textbf{onde } h(x_i) = mx_i + b,$$

pode-se, inicialmente, dividir a amostra em subamostras *mini-batch* A_K de tamanho s, e.g., $50 \leq s \leq 256$:

$$A_K = \{(x_i, y_i), \textbf{ para } K \leq i \leq (K + s)\},$$
$$\textbf{onde } K \in \{1, s + 1, 2s + 1, \cdots, (N_a/s - 1)s + 1\}.$$

A função de custo para A_K é:

$$C_K(m, b) = \frac{1}{2s} \sum_{i=K}^{K+s} (h(x_i) - y_i)^2.$$

E o gradiente da função de custo da subamostra K:

$$\nabla C_K(m, b) = \left(\frac{\partial C_K}{\partial m}, \frac{\partial C_K}{\partial b} \right)$$
$$\textbf{onde } \frac{\partial C_K}{\partial m} = \frac{1}{s} \sum_{i=K}^{K+s} (mx_i + b - y_i)x_i$$
$$\frac{\partial C_K}{\partial b} = \frac{1}{s} \sum_{i=K}^{K+s} (mx_i + b - y_i).$$

Com base no particionamento da amostra e da formulação apresentada, tem-se o seguinte algoritmo de descida do gradiente no modo estocástico:

- arbitre valores para m, b, α e s
- particione a amostra A em $N_a/s - 1$ *mini-batches* de s elementos
- calcule o custo inicial $C(m, b)$ de toda a amostra A

- repita os passos abaixo, que formam uma epoch[2], até C convergir:
  - para cada mini-batch K selecionado:
    - calcule seu gradiente $\nabla C_K(m,b) = \left(\frac{\partial C_K}{\partial m}, \frac{\partial C_K}{\partial b}\right)$, usando os valores correntes de m e b
    - atualize $m := m - \alpha\frac{\partial C_K}{\partial m}$ **e** $b := b - \alpha\frac{\partial C_K}{\partial b}$
  - recalcule $C(m,b)$ e compare com valor anterior

O modo estocástico é mais eficiente que o *batch*, pois agora o cálculo do custo total C somente é realizado $N_a/s-1$ vezes, e não N_a vezes.

4.6 Gradiente Descendente com n Variáveis

No caso de se ter $n > 1$ variáveis independentes na amostra de trabalho, tem-se a seguinte reformulação:

- Amostra para treinamento:
  $\{(x^{(i)}, y^{(i)}), \textbf{para } 1 \leq i \leq N_a\}$, **onde**
  $x = (x_1, \cdots, x_n)$, $m = (m_1, \cdots, m_n)$ **e** $n \geq 1$
- Hipótese para h:
  $h(x^{(i)}) = \sum_{j=1}^{n} m_j x_j + b, \textbf{para } 1 \leq i \leq N_a$
- Indicador C de erro da hipótese h em relação à amostra agora é:
  $C = \frac{1}{2N_a}\sum_{i=1}^{N_a}\left[h(x^{(i)}) - y^{(i)}\right]^2$

[2] O termo *epoch* representa uma passagem por toda a amostra de treinamento, enquanto o termo iteração corresponde à atualização de um parâmetro do modelo.

- Afinação dos pesos m_k, $k : 1..n$, e b para minimizar C, conforme a formulação

  $\frac{\partial C}{\partial m_k} = \frac{1}{N_a} \sum_{i=1}^{N_a} (\sum_{j=1}^{n} m_j x_j^{(i)} + b - y^{(i)}) x_k^{(i)}$

  $\frac{\partial C}{\partial b} = \frac{1}{N_a} \sum_{i=1}^{N_a} (\sum_{j=1}^{n} m_j x_j^{(i)} + b - y^{(i)})$

  faça $m_k := m_k - \alpha \frac{\partial C}{\partial m_k}$ **e** $b := b - \alpha \frac{\partial J}{\partial b}$

  onde α é a *taxa de aprendizado*

- E repetem-se as afinações de m e b até C parar de diminuir.

4.7 Regressão Logística

Uma regressão é chamada de logística quando o resultado da regressão linear $y = h(x)$ é modelado a ter apenas valores discretos, tipicamente $y \in \{0, 1\}$ ou $y \in \{\text{false}, \text{true}\}$ ou $y \in \{\text{sim}, \text{não}\}$, etc.

Tecnicamente, para obter $y = h(x)$ em regressão logística, deve-se executar os passos:

- faz-se uma regressão linear clássica $y' = h'(x)$
- calcula-se $y'' = \sigma(y')$, onde $\sigma(t) = \frac{1}{1+e^{-t}}$ é a função sigmoid, que retorna valores $\in [0, 1]$, também chamada de logística
- discretiza-se y'' para obter o resultado final

$$y = s(y''), \textbf{ onde } s(t) = \begin{cases} 0 \text{ se } t \leq 0.5 \\ 1 \text{ se } t > 0.5 \end{cases}$$

4.8 ALERTA!

Regressão não estabelece nem identifica relação de causa e efeito. Se essa relação existir na massa de dados, regressão apenas permite determinar o efeito dado a causa.

Assim, regressão pode levar à conclusão errônea caso a relação de causa e efeito não exista entre os dados da amostra, como ocorre quando se deseja estabelecer uma relação de causa e efeito entre o comprimento do cabelo de uma pessoa, independentemente do sexo, e a ocorrência de câncer de próstata, como sugere a Fig. 4.1.

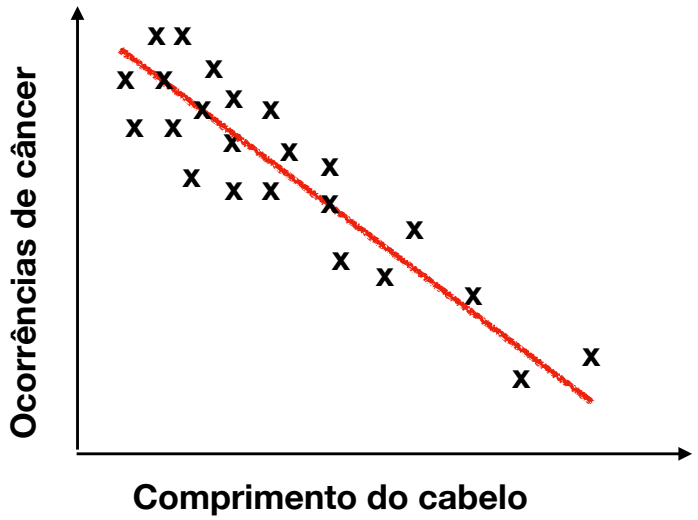


Figura 4.1: Causa e Efeito

A modelagem desse fenômeno por uma equação linear autoriza pensar que basta deixar o cabelo crescer para evitar essa doença?

Leitura Recomentada

- Louis Augustin Cauchy, Méthode Générale pour la Résolution des Systèms d'Équations Simultanées, Compte Rendu à l'Academie de Sciences Paris, 25:536-538, 1847 [citado por Claude Lemaréchal].
- Frederico Ferreira Campos, filho, Algoritmos Numéricos, Livros Técnicos e Científicos, Editora S.A., 383 páginas, 2001.
- Claude Lemaréchal, Cauchy and the Gradient Descent Method, Documenta Mathematica, Extra Volume ISMP, 251-254, 2012.
- Michael A. Nielsen, Neural Networks and Deep Learning, on-line book, 395 páginas, 2015.

***

Capítulo 5

Aprendizado de Máquina

Aprendizado de máquina (*machine learning*) é o processo de descobrir uma função-objetivo f que mapeie as entradas (*features*) x nos respectivos resultados (*labels*) y, i.e., $y = f(x)$. E esse processo de aprendizado é baseado nos dados (x_i, y_i) da amostra de treinamento.

Quando a forma geral da função f a ser aprendida é pré-fixada, o algoritmo de aprendizado é dito parametrizado, e sua função é inferir os parâmetros ou coeficientes que compõem a função f.

Mas, quando não se faz suposição quanto à forma da função f, o algoritmo de aprendizado é dito não-parametrizado. Nesse caso, pode ser necessário testar diferentes alternativas para escolher a melhor forma da função e os valores de seus parâmetros.

5.1 Aprendizado Parametrizados

Como foi dito, a forma da função $y = f(x)$ é arbitrada, e o conjunto de parâmetros ou coeficientes da função

são prefixados e não dependem do tamanho da amostra de treinamento.

Esses algoritmos de treinamento envolvem os seguintes passos:

- selecione uma amostra de treinamento e outra de validação
- selecione um modelo ou forma de função-objetivo
- faça o treinamento para determinar os valores dos coeficientes da função
- faça a validação do aprendizado.

As vantagens dos algoritmos paramétricos são:

- simplicidade: os métodos são fáceis de entender
- eficiência: aprendizado é rápido
- amostras menores: treinamento não requer amostras grandes
- genericidade: funciona mesmo se o ajuste da função à amostra não for perfeito

E as desvantagens dos algoritmos paramétricos são:

- riscos: a prefixação da forma da função limita a qualidade do aprendizado, sendo possível a ocorrência de *underfitting*[1]
- escalabilidade: algoritmos limitados a problemas mais simples.

[1] *Undefitting* é uma situação em que os erros nos resultados computados ainda são insatisfatórios, mas não consegue reduzi-los.

5.2 Aprendizado Não-Parametrizados

Nos algoritmos do aprendizado não-parametrizados, a forma da função $y = f(x)$ não é prefixada, devendo ser determinada pelo algoritmo de treinamento. Em consequência, os parâmetros da forma geral e seus valores também devem ser inferidos pelo algoritmo de aprendizado. Assim, sem prefixação da forma da função, o algoritmo é livre para aprender qual é a melhor forma funcional a partir da amostra de treinamento.

Essa abordagem elimina os riscos de prefixação da forma da função e deve ser usada quando se têm grandes volumes de dados amostrais e pouco conhecimento de seu padrão.

De fato, a literatura da área cita modelos que são baseados em bilhões de parâmetros, quantidade que inviabiliza o uso de algoritmos paramétricos.

As vantagens dos algoritmos não-paramétricos são:

- flexibilidade: capazes de fazer ajustes de muitas formas funcionais
- aplicabilidade: não têm limitação quanto à forma da função a ser aprendida
- desempenho: apresentam grande capacidade de generalização.

E as desvantagens são:

- demanda de dados: necessidade de grandes amostras de treinamento para induzir a função desejada

- custo: treinamento mais trabalhoso, pois pode requerer determinação de muitos parâmetros de treinamento
- riscos: observa-se maior risco de *overfitting*[2].

5.3 Generalização da Amostra

Aprendizado supervisionado de máquina visa inferir a função-objetivo $y = f(x)$ a partir da amostra de treinamento (x_i, y_i). Consequentemente, o ponto mais importante do aprendizado é que a função f seja capaz de mapear corretamente entradas que não estejam na amostra em valores válidos. Ou seja, o aprendizado visa generalizar o mapeamento descrito pela amostra para quaisquer dados dentro do domínio do problema.

Generalização é indispensável, pois tipicamente amostras de treinamento podem ser incompletas e imprecisas. E generalização está fortemente vinculada à distribuição dos dados da amostra de treinamento.

Certamente, se essa distribuição dos dados mudar, deve-se proceder com um retreinamento para ajustar f à nova distribuição. Pois se as novas entradas apresentadas à função f inferida pelo algoritmo de aprendizado não seguirem a distribuição dos dados da amostra de treinamento, o modelo pode não funcionar adequa-

[2]Diz que há *overfitting* quando o algoritmo gerado produz resultados precisos para dados da amostra, mas é incapaz de generalizar para novos dados

damente, implicando na necessidade de retreinamento com a ampliação da distribuição de dados para incorporar novas entradas.

O mecanismo de inferência usado pelo mecanismo de generalização é o de indução, no sentido em que trata-se de aprender conceitos gerais a partir de exemplos específicos.

Note que isso difere de dedução, a qual busca aprender conceitos específicos a partir de regras gerais.

5.4 Resumo dos Passos de um Aprendizado

1. Identifique uma amostra de treinamento A:
   $\{(x^{(i)}, y^{(i)}), i : 1..N_a\}$,
   onde cada *feature* tem a forma $x = (x_1, \cdots, x_m)$
2. Arbitre uma função $y = g(w, x)$, que mapeie as entradas x a resultados y, com desvio ou custo médio dado por
   $C(w) = \frac{1}{2N_a} \sum_{i=1}^{N_a} \left[g(w, x^{(i)}) - y^{(i)}\right]^2$
   sendo w um vetor de constantes cujos valores devem ser aprendidos pelo algoritmo de treinamento
3. Aplique o algoritmo da descida pelo gradiente para identificar (ou aprender) o valor do vetor $\overline{w}$ das constantes em w que minimiza o custo médio $C(w)$

A generalização do mapeamento descrito pela amostra A é então dado pela equação $y = h(x)$, onde a função $h(x) = g(\overline{w}, x)$.

5.5 Redes Neurais X Aprendizado de Máquina

A definição da forma mais adequada da função citada $g(w, x)$ é essencial para se ter um bom aprendizado. Haja vista que a forma escolhida para essa função tem efeito direto na qualidade da generalização da amostra de treinamento.

Em princípio, a definição da forma de $g(w, x)$ é arbitrária e empírica, sendo feita por tentativa e erro. Por outro lado, um método sistemático de definição da forma geral de g, ainda que empírico, como as redes neurais, melhora os resultados.

Assim, o caminho mais praticado, em vez de arbitrar a expressão de uma função de generalização com coeficientes a ser aprendidos, é arbitrar a estrutura de uma rede neural com pesos e *biases* a serem determinados.

Leitura Recomentada

- Hal Daumé III, A Course in Machine Learning, disponível na Internet, Version 0.8, 189 páginas, August 2012.
- Michael A. Nielsen, Neural Networks and Deep Learning, on-line book, 395 páginas, 2015.

***

Capítulo 6

Redes Neurais Densas

Em neurociência, um neurônio humano é uma estrutura orgânica que pode receber informações em seus terminais denominados dentritos e transmitem comandos via os seus terminais chamados de axônios.

Um neurônio humano recebe sinais elétricos, modulados em certas quantidades, em alguns de seus dentritos, e dispara um sinal de saída nos seus axônios em que a somatória dos sinais de entrada exceder um dado limiar.

Se x_i, $i : 1..n$, representam os terminais de neurônios que recebem impulsos elétricos, e y_i, $i : 1..m$, terminais que transmitem impulsos elétricos a outros neurônios de uma rede neural biológica, então pode-se ter um conjunto de neurônios com interconexões dinâmicas dos axônios de um neurônio a dentritos de outro.

Cada interconexão instalada é chamada de sinapse, por meio da qual sinais são transmitidos de um neurônio a outro. Sempre que a somatória dos sinais recebidos por um neurônio ultrapassa certo limiar ou viés, uma ação é propagada para outros neurônios via alguns de

seus terminais axônios.

As sinapses no neurônio biológico são controladas pela amplitude dos sinais recebidos, os quais combinados internamente pelos pesos associados à conexões, como no modelo matemático de um neurônio biológico da Fig. 6.1, produz uma decisão. Os sinais de entrada nesse modelo são valores numéricos inteiros, e o resultado é a soma ponderada das entradas x_1, ..., x_n, limitada por um viés $-b$. Esse resultado produz uma decisão que pode ou não ser transmitida a outro neurônio.

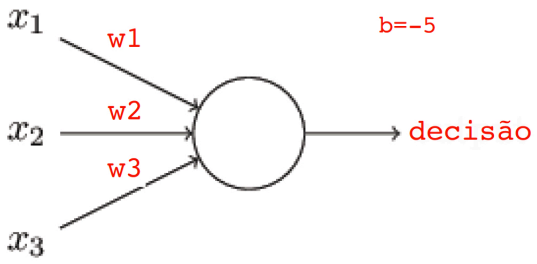


Figura 6.1: Neurônio Matemático

Por exemplo, se o neurônio da Fig. 6.1 modela a decisão de eu ir ou não ao Mineirão ver um jogo, supõe-se que x_1, x_2 e x_3 sejam condições que levam à decisão:

- x_1 indica se o jogo é de meu time (peso $w_1 = 4$)
- x_2 indica se minha mulher vai comigo (peso $w_2 = 1$)
- x_3 indica se tenho carona assegurada (peso $w_3 = 2$)
- b é chamado de limiar (*bias*) ($b = -5$, onde 5 é o *threshold*)

A decisão de ir ou não ao jogo pode ser computada da seguinte forma:

$$\text{decisão} = f(\overbrace{x_1w_1 + x_2w_2 + x_3w_3 + b}^{a})$$

$$\text{decisão} = \begin{cases} 0 \text{ se } a \leq 0 \Longrightarrow \neg \text{ sinapse} \\ 1 \text{ se } a > 0 \Longrightarrow \quad \text{sinapse} \end{cases}$$

Alternativamente, se forem usados os pesos $(4, 3, 1)$, a entrada x_2 passa a definir se haverá sinapse ou não.

6.1 Neurônios Perceptrons

Perceptron é um modelo matemático de um neurônio biológico de n entradas, em que os sinais elétricos recebidos nos seus dendritos são representados por valores binários $\in \{0, 1\}$, indicando presença ou não de sinal. A saída também é um valor binário, que indica se houve sinapse. Dado o neurônio da Fig. 6.2:

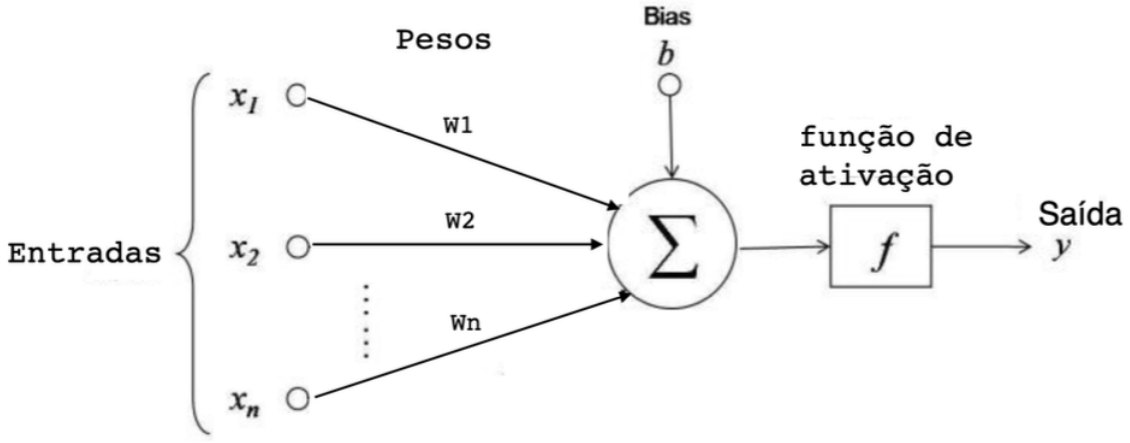


Figura 6.2: Neurônio Perceptron

- **onde** $x_i \in \{0, 1\}$ **e** $\Sigma = \sum_{i=1}^{n} w_i x_i + b$
  $x_i = 0$: sem estímulo na entrada x_i

$$x_i = 1\text{: com estímulo na entrada } x_i$$
$$y = f(\Sigma) = \begin{cases} 0 \text{ se } \Sigma \leq 0 \Longrightarrow \text{sem sinapse} \\ 1 \text{ se } \Sigma > 0 \Longrightarrow \text{gera sinapse} \end{cases}$$

a força dos sinais recebidos é obtida via uma soma Σ ponderada pelos pesos w_i, $i : 1..n$, indicados no modelo. E ao valor obtido adiciona-se o valor do *bias* $b < 0$ dado. O resultado final é então submetido a uma função de ativação que o normaliza aos valores desejados, conforme detalha a Fig. 6.2.

6.2 *Modus Operandi* de um Perceptron

O modelo de decisão de um perceptron depende de seus pesos e *biases*, haja vista que perceptron é um dispositivo que toma decisões ponderando evidências conforme a entrada fornecida.

Cada entrada $x_i \in \{0, 1\}$ indica se a evidência associada deve contribuir ou não para a decisão final. E uma saída y com valor 1 indica uma decisão que deve ser propagada para outros perceptrons, assim realizando uma sinapse.

O peso w_i de cada evidência define seu poder de influência no processo de ponderação para a decisão final. O *threshold*, *bias* ou limiar, define o valor mínimo que influências das evidências devem atingir para permitir uma decisão positiva, sendo assim uma medida da propensão de o perceptron produzir a saída 1.

O exemplo da Fig. 6.3 mostra um perceptron que simula o circuito básico NAND. Nesse perceptron, tem-se $z = x_1 w_1 + x_2 w_2 + 3$ **e** $y = f(z)$.

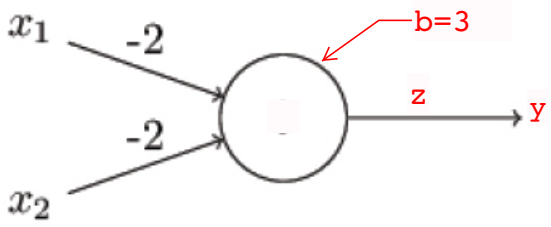


Figura 6.3: NAND

E a tabela a seguir comprova que o resultado final é de fato $y = x_1$ NAND x_2.

x_1	x_2	z	y
0	0	0 * (-2) + 0 * (-2) + 3 = 3	1
0	1	0 * (-2) + 1 * (-2) + 3 = 1	1
1	0	1 * (-2) + 0 * (-2) + 3 = 1	1
1	1	1 * (-2) + 1 * (-2) + 3 = -1	0

Pois, a tabela da verdade de um NAND mostra o mesmo resultado.

Como portas NANDs, rede de *perceptrons* podem computar qualquer função lógica! Isso permite dizer que *perceptron* é um elemento universal para computação, principalmente porque uma rede com muitas camadas permite realizar diferentes e complexas operações.

Q = A NAND B

Tabela da Verdade

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

6.3 Redes de Perceptrons

Perceptrons podem ser organizados em redes de várias camadas como a Fig. 6.4, na qual destacam-se a camada de entrada, duas camadas internas ou ocultas, e a camada de saída.

Na camada de entrada, *perceptrons* são ativados com os estímulos de entrada. Nas ocultas, ponderam-se os resultados da camada anterior em um nível mais complexo e mais abstrato, e, na camada de saída, o *perceptron* faz a ponderação para a decisão final. E todas as conexões têm pesos para definir sua contribuição.

Quando todos os neurônios de uma camada estão conectados a todos da camada seguinte, diz-se que a rede é densa.

O exemplo a seguir define uma rede de *perceptron*

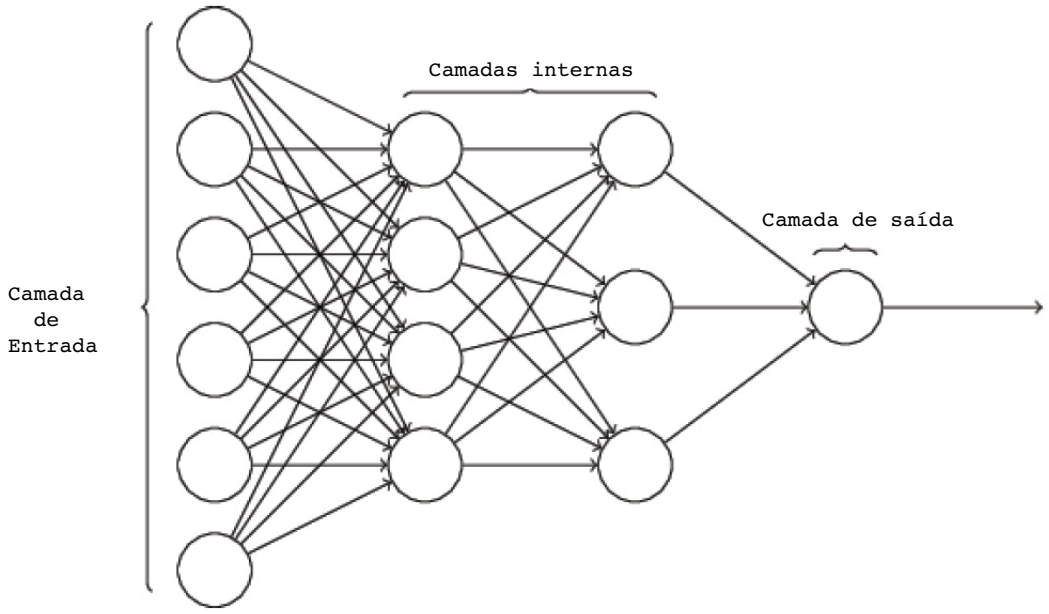


Figura 6.4: Redes de Perceptions

que efetua a soma de dois bits, da mesma forma que é realizada pelo circuito de NANDs da Fig. 6.5.

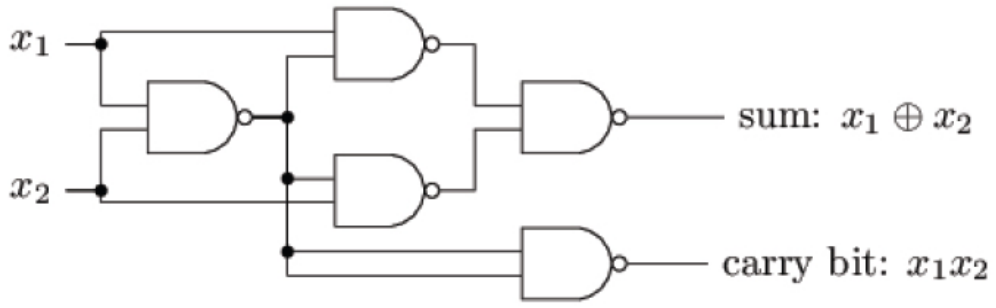


Figura 6.5: Somador

E a rede de *perceptrons*, com $bias = 3$, correspondente pode ser:

6.4 Neurônios Sigmoids

Um neurônio sigmoid de n entradas é um neurônio derivado de um perceptron no qual, entre outras pequenas

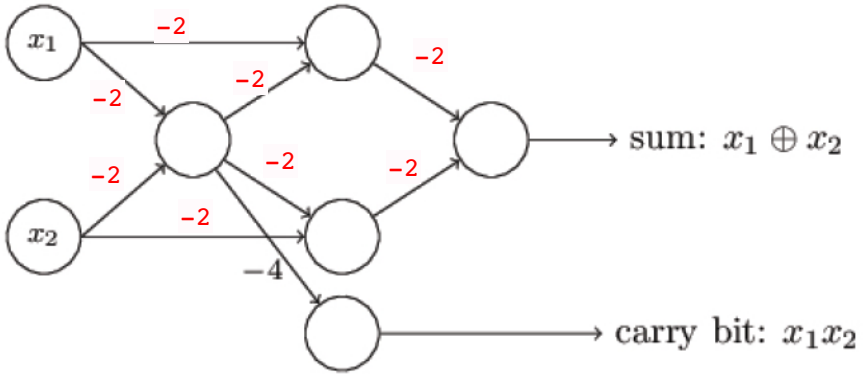


Figura 6.6: Redes de *Perception* 2

diferenças, a função de ativação f é a função *sigmoid* σ, definida por $\sigma(t) = 1/(1 + e^{-t})$, a qual transforma um número real em um do intervalo $[0.0, 1.0]$.

Graficamente, dado o seguinte neurônio

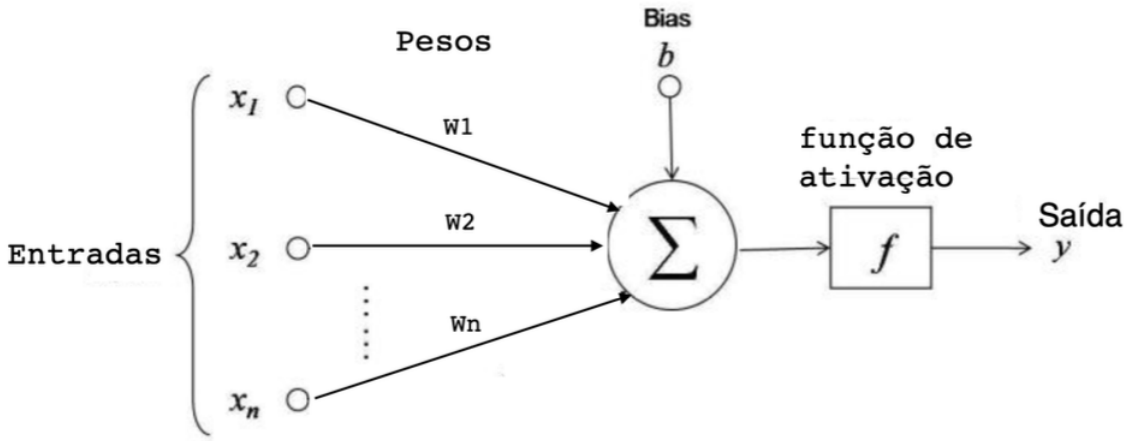


Figura 6.7: Neurônio Sigmoid

onde:

- as entradas $x_i \in [0, 1]$ agora são valores em $\mathbb{R}$
- $\Sigma = \sum_{i=1}^{n} w_i x_i + b$
- a função de ativação $f = \sigma : \mathbb{R} \to \mathbb{R}$ é a função sigmoid definida por $\sigma(t) = 1/(1 + e^{-t})$
- $y = f(\Sigma)$, tal que $y \in \mathbb{R}$ e $y \in [0, 1]$.

os algoritmos de aprendizado de máquina consistem em afinar pesos e *biases* da rede de neurônios artificiais para obter a saída correta. Para isso, é indispensável que pequenas modificações nos pesos e *biases* provoquem apenas pequenas mudanças no valor de saída:

$$\Delta\text{saída} \approx \sum_j \frac{\partial\ \text{saída}}{\partial w_j}\Delta w_j + \frac{\partial\ \text{saída}}{\partial b}\Delta b$$

Todos os pesos e *biases* devem ser afinados iterativamente por meio de algoritmos a serem apresentados oportunamente.

Como a saída de todo perceptron é binária, não há como obter *pequenas* mudanças em sua saída, i.e., ou se tem 0 ou se tem 1. Assim, não é óbvio como fazer uma rede de perceptrons aprender!

Por outro lado, em neurônios sigmoids, as entradas e saídas são valores reais no intervalo $[0.0, 1.0]$. Assim, aprendizado de máquina funciona melhor com a função σ (sigmoid), pois, pequenas mudanças nos pesos e *biases* geram pequenas alterações no resultado final $y \in [0.0, 1.0]$.

Após a afinação, para gerar ou não a sinapse, pode-se interpretar a saída $y \leq 0.5$ como false e $y > 0.5$ como true.

6.5 Treinamento de um Neurônio Sigmoid

O treinamento de um neurônio sigmoid, inicia-se com a definição de sua estrutura como a da figura a seguir,

onde estão definidas as entradas e os parâmetros como pesos das conexões e *bias*:

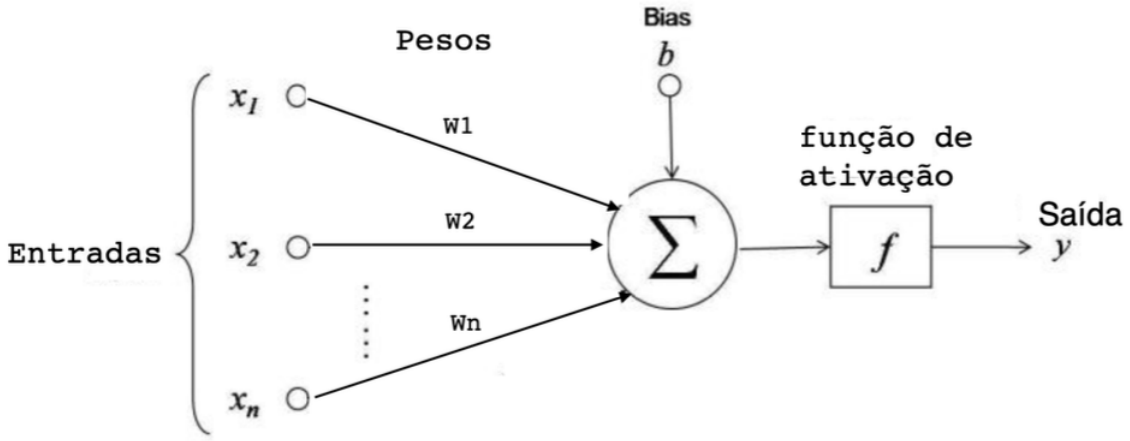


Figura 6.8: Neurônio Sigmoid

Se a amostra de treinamento for $(x^{(i)}, y^{(i)})$, $i : 1..N_a$, sendo $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ e f, a função sigmoid σ, computa-se o erro C do treinamento por:

$$C(w_1, \cdots, w_n, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} \left[\sigma(\sum_{k=1}^{n} w_k x_k^{(i)} + b) - y^{(i)}\right]^2.$$

E em seguida, pode-se determinar o valor do *bias* b e dos pesos w_k, $k : 1..n$, pelo algoritmo de descida pelo gradiente, sendo a função a ser interpolada é a provida pelo somatório computado pelo neurônio, não sendo assim, necessário definir a expressão analítica da função a ser interpolada.

Um exemplo mais elaborado é exibido pela Fig. 6.8, extraída do livro [12], a qual mostra um exemplo de uma rede neural mais complexa, usada para realizar reconhecimento de dígitos decimais escritos a mão e re-

presentado digitalmente como um matriz 28X28 bits ou um vetor de 784 posições:

Se f for a função sigmoid σ de ativação, pode-se escrever:

- $y_k^{(1)} = f(\sum_{i=1}^{784} w_i^{(1)} x_i + b^{(1)})$ **para** $k : 1..5$
- $y_k^{(2)} = f(\sum_{i=1}^{5} w_i^{(2)} y_i^{(1)} + b^{(2)})$ **para** $k : 1..10$
- $y_k^{(3)} = f(\sum_{i=1}^{10} w_i^{(3)} y_i^{(2)} + b^{(3)})$ **para** $k : 1..4$

A função de ativação sigmoid, definida por

$$\sigma(t) = \frac{1}{1+e^{-t}},$$

e cuja derivada é:

$$\sigma'(t) = \sigma(t)(1 - \sigma(t)),$$

confina valores de t no intervalo $[0.0, 1.0]$ das probabilidades e não necessariamente forma uma distribuição de probabilidade. Essa função tem papel fundamental na definição de uma rede densa de neurônios sigmoids.

Outras funções de ativação populares, não abordadas neste texto, são:

- Função ReLU (*Rectified Linear Unit)*:
  - despreza valores negativos
  - extremamente simples
- Função Tangente Hiperbólica (*tanh*):
  - semelhante à função sigmoid
  - confina valores no intervalo $[-1.0, 1.0]$
- Função *softmax*:
  - usada em redes neurais de classificação

– confina valores no intervalo $[-1.0, 1.0]$

6.6 Função Modelada pela Rede

Os algoritmos de treinamento para formulação de aprendizado de máquina necessitam que seja arbitrada uma função a qual deve-se ajustar a amostra de dados. Certamente, quanto mais elaborada for essa função, melhor pode ser o aprendizado.

Escolher uma boa função para essa modelagem não é tarefa fácil, mas isso pode ser automatizado, tomando para o aprendizado a função que mapeia as entradas de uma rede de neurônios em sua saída. E a tarefa de definir uma rede neural é bem mais simples do que produzir analiticamente uma boa função de modelagem.

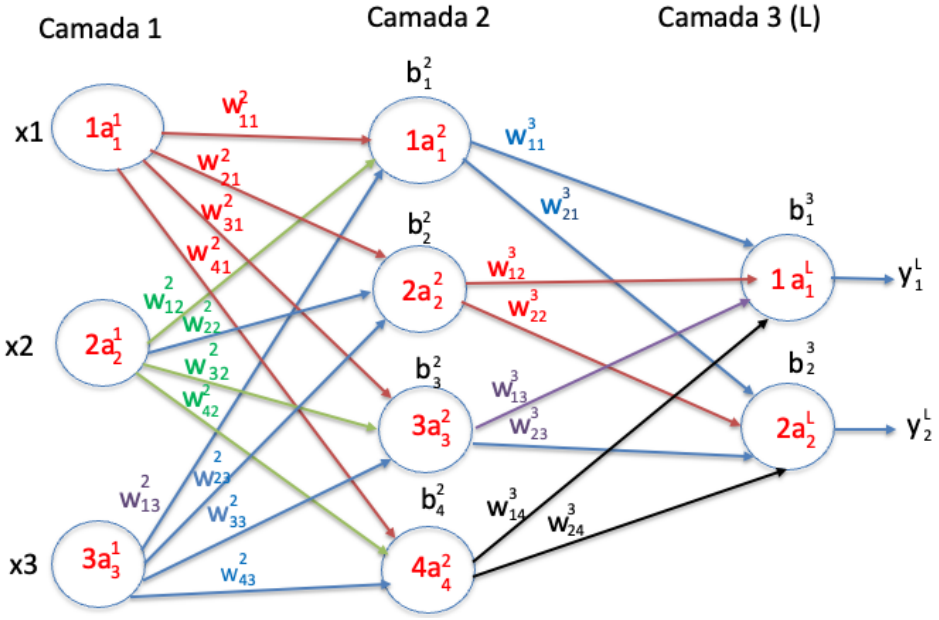


Figura 6.9: Uma Rede de 3 Camadas

E o mais importante: fornecer a expressão analítica dessa função não é necessário para o algoritmo de aprendizado, basta que se forneça a rede, por exemplo.

Além disso, redes neurais densa encapsulam funções polinomiais de alto grau, o que confere mais precisão ao processo, conforme ilustra a rede da Fig. 6.9, onde f é a função de ativação de cada neurônio:

- $a_i^1 = x_i$ **para** $i : 1..3$
- $a_1^2 = f(b_1^2 + \sum_{k=1}^{3} w_{1k}^2 a_k^1)$
- $a_1^2 = f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$
- $a_2^2 = f(b_2^2 + \sum_{k=1}^{3} w_{2k}^2 a_k^1)$
- $a_2^2 = f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$
- $a_3^2 = f(b_3^2 + \sum_{k=1}^{3} w_{3k}^2 a_k^1)$
- $a_3^2 = f(b_3^2 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3)$
- $a_4^2 = f(b_4^2 + \sum_{k=1}^{3} w_{4k}^2 a_k^1)$
- $a_4^2 = f(b_4^2 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3)$
- $a_1^3 = f(b_1^3 + \sum_{k=1}^{4} w_{1k}^3 a_k^2)$
- $a_1^3 = f(b_1^3 + w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + w_{13}^3 a_3^2 + w_{14}^3 a_4^2)$
- $a_1^3 = f(b_1^3 + w_{11}^3 f(b_1^2 + w_{11}^2 a_1^1 + w_{12}^2 a_2^1 + w_{13}^2 a_3^1)$
  $+ w_{12}^3 f(b_2^2 + w_{21}^2 a_1^1 + w_{22}^2 a_2^1 + w_{23}^2 a_3^1)$
  $+ w_{13}^3 f(b_2^3 + w_{31}^2 a_1^1 + w_{32}^2 a_2^1 + w_{33}^2 a_3^1)$
  $+ w_{14}^3 f(b_2^4 + w_{41}^2 a_1^1 + w_{42}^2 a_2^1 + w_{43}^2 a_3^1))$

- $a_1^3 = f(b_1^3 + w_{11}^3 f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$
  $+ w_{12}^3 f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$
  $+ w_{13}^3 f(b_2^3 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3)$
  $+ w_{14}^3 f(b_2^4 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3))$

- $a_2^3 = f(b_2^3 + \sum_{k=1}^{4} w_{2k}^3 a_k^2)$
- $a_2^3 = f(b_2^3 + w_{21}^3 a_1^2 + w_{22}^3 a_2^2 + w_{23}^3 a_3^2 + w_{24}^3 a_4^2)$
- $a_2^3 = f(b_2^3 + w_{21}^3 f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$
  $+ w_{22}^3 f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$
  $+ w_{23}^3 f(b_2^3 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3)$
  $+ w_{24}^3 f(b_2^4 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3))$

Em resumo, a forma da função modelada é:

- $a_i^1 = x_i$ **para** $i : 1..3$
- $a_1^2 = f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$
- $a_2^2 = f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$
- $a_3^2 = f(b_3^2 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3)$
- $a_4^2 = f(b_4^2 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3)$
- $a_1^3 = f(b_1^3 + w_{11}^3 f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$
  $+ w_{12}^3 f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$
  $+ w_{13}^3 f(b_2^3 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3)$
  $+ w_{14}^3 f(b_2^4 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3))$

- $a_2^3 = f(b_2^3 + w_{21}^3 f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$
  $+ w_{22}^3 f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$
  $+ w_{23}^3 f(b_2^3 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3)$
  $+ w_{24}^3 f(b_2^4 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3))$

O resultado final é o seguinte:

- Função $f = \sigma$, **onde** $\sigma(t) = 1/(1 + e^{-t})$
- $a_2^L = g(x_1, x_2, x_3) = f(b_2^3 +$
  $w_{21}^3 f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3) +$
  $w_{22}^3 f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3) +$
  $w_{23}^3 f(b_2^3 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3) +$
  $w_{24}^3 f(b_2^4 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3))$
- $y_2^L = a_2^L$

A definição de a_2^L possui dois componentes: um linear em relação às *features* x_1, x_2 e x_3 e aos pesos das conexões e um outro, que pode ser não-linear.

Assim, uma rede neural tipicamente é uma representação geral de funções que têm um componente linear e outro não-linear. E isso permite realizar aprendizado com funções de generalização mais elaboradas, sem que se tenha de arbitrar analiticamente sua expressão matemática. Os valores dos parâmetros de rede neural são determinados pelos lgoritmos de aprendizado.

6.7 *Deep Learning*

A rede neural usada no algoritmo de aprendizado de máquina é constituída por uma camada de entrada com tantos neurônios quanto o tamanho da tupla de entrada da amostra, por uma camada de saída com tantos neurônios quanto o tamanho da tupla de resultados esperados e por zero ou mais camadas ocultas com um número variado de neurônios determinado experimentalmente.

Recomenda-se que o número de neurônios de camadas ocultas esteja entre 2/3 do número de neurônios nas camadas de entrada e de saída e o dobro do tamanho da camada de entrada. E também deve ter pelo menos o mesmo número de neurônios da camada de entrada.

As camadas ocultas determinam o poder de aprendizado da rede, por exemplo: zero camadas permite que a rede represente somente funções lineares, uma camada para redes capazes de representar qualquer função de um espaço finito a outro também finito. Mais camadas representa qualquer função contínua.

O desafio da IA é resolver problemas cujas soluções são muito difíceis de serem descritas formalmente. A ideia é resolver problemas compondo soluções de problemas mais simples, já resolvidos experimentalmente.

Soluções são coletadas a partir de experiências, dispensando a necessidade de especificação formal do co-

nhecimento necessário, de forma que a hierarquia de conceitos permita que o computador aprenda conceitos mais complexos a partir de conceitos mais simples.

E uma rede neural com apenas uma camada oculta é dita *shallow*, e uma rede neural com duas ou mais camadas ocultas é dita *deep*.

Deep learning é um aprendizado com redes neurais com pelo menos quatro camadas (entrada, duas ocultas, saída). Nesse tipo de aprendizado de máquina, o computador é treinado a realizar tarefas típicas de humanos, como reconhecimento de fala ou de imagens.

Com *deep learning*, o computador é submetido a um conjunto de dados que permite determinar a relevância de cada tipo de dado fornecido, de forma que sua composição em várias camadas, tipicamente de 5 a 10 camadas ocultas, possa produzir para novos dados respostas coerentes com as da amostra.

Desta forma, *deep learning* requer alto poder computacional, pois é de natureza iterativa, envolvendo uma quantidade elevada de dados, e realiza composição de resultados intermediários em muitas camadas.

6.8 Uma Rede para Reconhecimento de Dígitos

A rede neural *deep* da Fig. 6.10, extraída do livro [12], para reconhecimento de dígitos possui 784 entradas e 10 saídas.

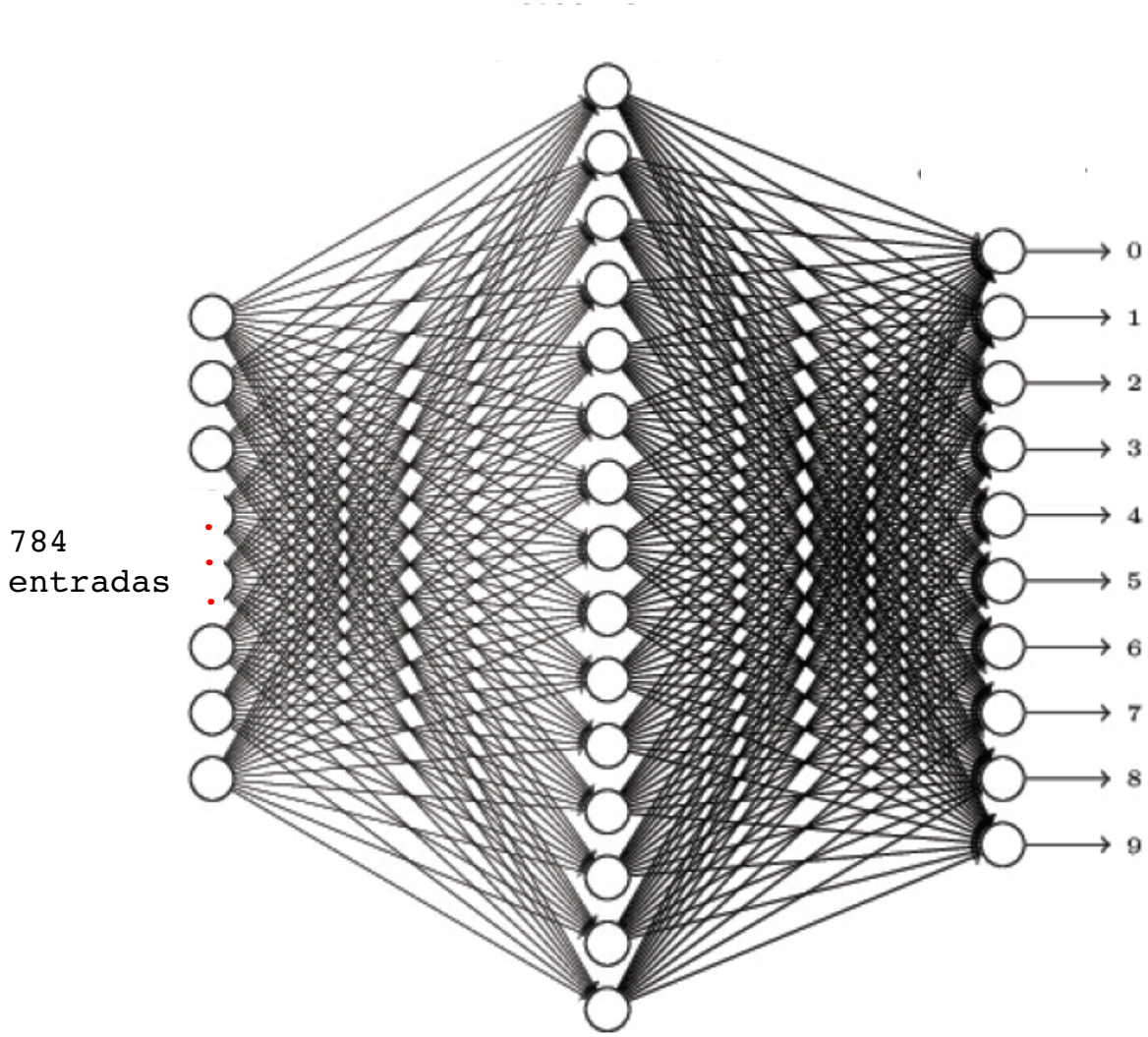


Figura 6.10: Reconhecimento de Dígitos (extraída de [12])

Se a imagem de um dígito tiver 28X28 = 784 pixels, esses pixels formam a entrada da rede e a saída da rede é um vetor que identifica o dígito reconhecido.

Seu uso é bastante simples: o usuário define a arquitetura da rede neural e fornece as amostras de treinamento. A partir daí, o algoritmo de treinamento, descrito no próximo capítulo, descobre os pesos e *biases* que asseguram o reconhecimento desejado.

Leitura Recomentada

- Hal Daumé III, A Course in Machine Learning, disponível na Internet, Version 0.8, 189 páginas, August 2012.
- Michael A. Nielsen, Neural Networks and Deep Learning, on-line book, 395 páginas, 2015.
- Andrew Ng, Machine Learning Yearning - draft, 118 páginas, free book, 2018.
- Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, Dive Into Deep Learning, pdf Release 0.7.1, 912 páginas, jan 25, 2020.

***

Capítulo 7

Treinamento de Redes Neurais Densas

O treinamento de uma rede neural compreende as seguintes atividades, executadas nessa ordem:

1. Definição da arquitetura de uma rede de neurônios (L, n, w, b), com L camadas, n_l neurônios na camada $l \in [1, L]$, com matriz de pesos w e vetor de *biases* b a ser ajustados pelo treinamento.
2. Preparação da amostra $A = \{(x, y)\}$ para treinamento.
3. Dedução da função de custo $C(x, y, w, b)$, que fornece a grandeza do erro de treinamento.
4. Seleção do método de minimização da função de erro C da Rede.
5. Formulação dos mecanismos de propagação de sinais na Rede.
6. Ajuste a rede (L, n, w, b) para garantir um C mínimo.

E o **resultado** final é uma rede (L, n, w, b) treinada com a amostra A, tal que, se iniciada com uma nova

entrada, produz uma saída compatível com o padrão dos resultados identificados na amostra A.

7.1 Definição da Arquitetura da Rede

Inicialmente é necessário definir a arquitetura de uma rede neural (L, n, w, b), a partir da qual, far-se-á o aprendizado desejado e nomear-se-á adequadamente todos os seus componentes, por exemplo, se a rede escolhida for

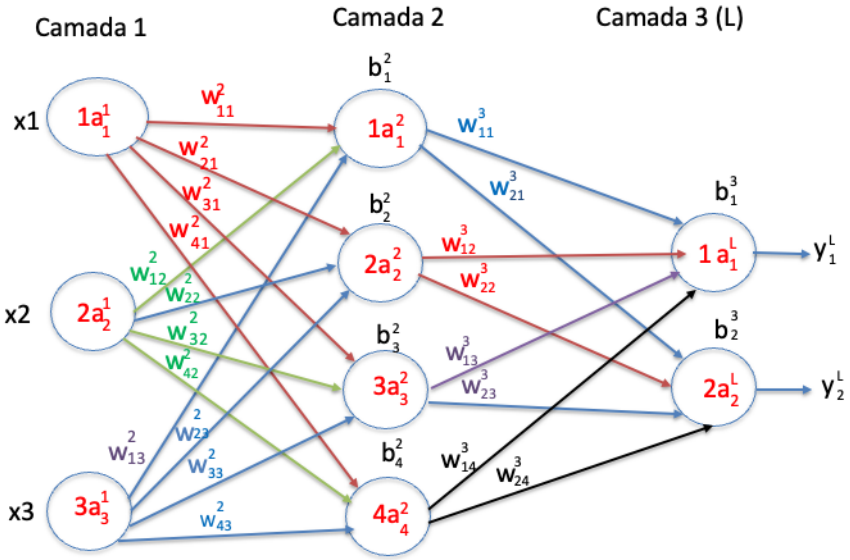


Figura 7.1: Arquitetura Escolhida da Rede

A camada de entrada desse exemplo é identificada por a_j^1, $j \in [1, n_1]$, e a de saída por a_j^L, $j \in [1, n_L]$.

A ativação na saída do neurônio j é designada a_j^l, para $l \in [1, L]$, α é a taxa de aprendizado (*learning rate*) e b é um arranjo contendo todos os *biases* b_j^l, i.e., b_j^l é o valor do *bias* do neurônio $j \in [1, n_l]$ da camada $l \in [2, L]$.

Continuando esse processo de identificação da rede, w^l_{jk} denota o peso da conexão do neurônio k da camada $l-1$ ao neurônio j da camada $l \in [2..L]$.

Cada neurônio $k : 1..n_{l-1}$ da camada $l-1$ está conectado a todos os neurônios $j : 1..n_l$ da camada l, e o peso de cada uma dessas conexões é w^l_{jk}, sendo w um arranjo contendo todos os pesos w^l_{jk}.

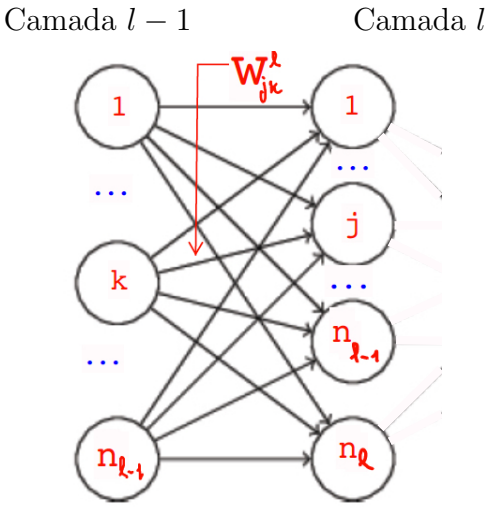


Figura 7.2: Pesos de Conexões w^l_{jk}

Deve-se então iniciar os parâmetros de (L, n, w, b) da rede da seguinte forma:

- Defina o número de camadas $L \geq 2$, no caso, $L = 3$.
- Inicialize vetor n com o número de neurônios por camada, que na Fig. 7.1 é $n = (3, 4, 2)$.
- Inicialize as matrizes w e b, atribuindo aos pesos e *biases* valores arbitrários, observando-se que:
  - w^l_{jk}: peso da conexão do neurônio k da camada $l-1$ ao j da camada $l \in [2, L]$

  - b_j^l: *bias* do neurônio j da camada $l \in [2, L]$

- E também deve-se inicializar a *learning rate* α com um valor definido experimentalmente.

No desenvolvimento a seguir, $a_j^l(i)$ denota uma ativação do neurônio j da camada l com a i-ésima entrada da amostra A, e σ é a função logística (sigmoid)

$\sigma(t) = \frac{1}{1+e^{-t}}$,

sendo sua derivada $\sigma'(t) = \sigma(t)(1 - \sigma(t))$.

7.2 Preparação da Amostra

A amostra de treinamento $A = \{(x, y)\}$, de tamanho N_a, onde x é uma matriz N_a X n_1 contendo todas as entradas (*features*) da amostra, e y, a matriz N_a X n_L contendo as saídas previstas, pode ser reescrita como:

$$A = \left\{ \left((x_1^{(i)}, \cdots, x_{n_1}^{(i)}), (y_1^{(i)}, \cdots, y_{n_L}^{(i)}) \right), i \in [1, N_a] \right\}$$

Se as escalas dos valores das entradas x forem de grandezas muito distintas, suas escalas devem ser normalizadas para que sejam da mesma ordem de grandeza, preferencialmente no intervalo $[0, 1]$ em $\mathbb{R}$. Por exemplo, supondo faixa de valores distintos como

$x_1^{(i)} \in [50, 10.000]$, $x_2^{(i)} \in [1, 10]$, $i : 1..N_a$,

recomenda-se definir fatores de normalização F_j como

$F_j := \max(x_j^{(i)})$, **para** $i : 1..N_a$

Caso a normalização das *features* não seja necessária, basta fazer $F_j := 1$, **para** $j : 1..n_1$.

7.3 Dedução da Função de Custo C

A Fig. 7.3 exibe função de custo ou desvio C da rede (L, n, w, b):

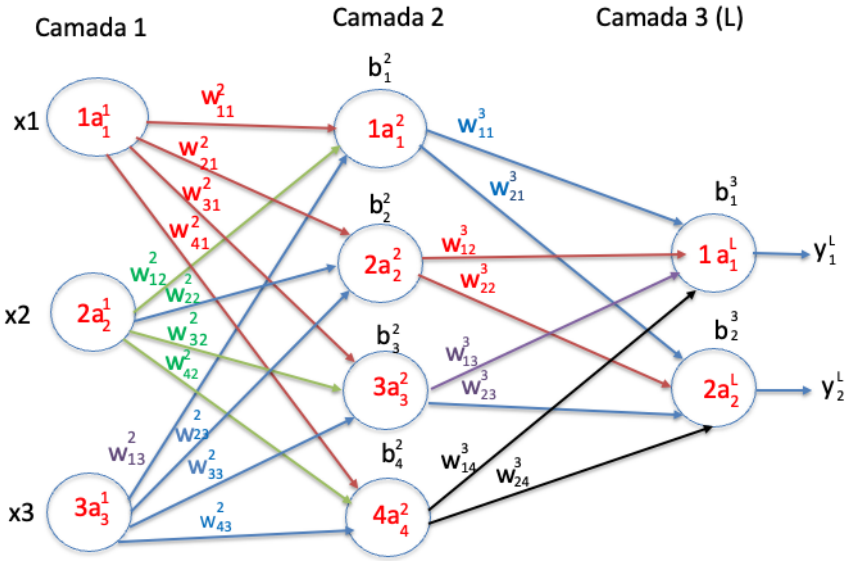


Figura 7.3: Rede (L, n, w, b)

dado o fator de normalização F_j, e que denota o erro da amostra $A = \{(x, y)\}$, de tamanho N_a é

$$C(x, y, w, b) = \frac{1}{N_a} \sum_{i=1}^{N_a} C^{(i)}(x, y, w, b),$$

$$\textbf{onde } C^{(i)}(x, y, w, b) = \sum_{j=1}^{n_L} C_j^{(i)}(x, y, w, b)$$

denota o erro acumulado de todos neurônios da camada L para item i.

Cada parcela do somatório acima,

$$C_j^{(i)}(x, y, w, b) = \frac{1}{2} \left(a_j^L(i) - y_j^{(i)} \right)^2, \; j \in [1, n_L],$$

é o erro do neurônio j da camada L.

A entrada da rede para cada item i, devidamente normalizada, é dada por

$$a_j^1(i) = x_j^{(i)} / F_j, \; j : 1..n_1,$$

e a ativação do neurônio, $j \in [1, n_l]$, das demais cama-

das, $l \in [2, L]$, para item i é $a_j^l(i) = \sigma(z_j^l(i))$, **onde**

$$z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l.$$

7.4 Minimização da Função de Erro

Por um método analítico, a minimização da função de medida de erro pode ser definida pela equação:

$$C(x, y, w, b) = \frac{1}{N_a} \left(\sum_{i=1}^{N_a} \sum_{j=1}^{n_L} C_j^{(i)}(x, y, w, b) \right)$$

a qual foi obtida analiticamente resolvendo as equações:

$$\frac{\partial C(x,y,w,b)}{\partial w_{jk}^l} = 0, \textbf{ onde } l : 2..L,\ j : 1..n_l \textbf{ e } k : 1..n_{l-1}$$

$$\frac{\partial C(x,y,w,b)}{\partial b_j^l} = 0, \textbf{ para } l : 2..L \textbf{ e } j : 1..n_l$$

Entretanto, caso o número de pesos e *biases* seja muito elevado, o uso de processos iterativos, e.g., Gradiente Descendente, pode ser mais interessante para encontrar o mínimo de C em relação a todos os pesos w_{jk}^l e *biases* b_j^l da rede neural.

O ajuste iterativo é feito diretamente no gráfico da rede, portanto dispensa o desenvolvimento das fórmulas, que pode ser muito complexo.

No ajuste iterativo de pesos e *biases*, observa-se que pequenas alterações nos valores dos pesos e *biases* causam pequenas alterações no valor de C, i.e.,

$$\Delta\text{saída} \approx \sum_j \frac{\partial \text{ saída}}{\partial w_j} \Delta w_j + \frac{\partial \text{ saída}}{\partial b} \Delta b$$

Assim, de acordo com o algoritmo do Gradiente Descendente, as alterações necessárias nos pesos e *biases* são:

- **para** $l : 2..L$, $j : 1..n_l$ **e** $k : 1..n_{l-1}$,
  faça $w^l_{jk} := w^l_{jk} - \alpha \frac{\partial C(x,y,w,b)}{\partial w^l_{jk}}$
- **para** $l : 2..L$ e $j : 1..n_l$,
  faça $b^l_j := b^l_j - \alpha \frac{\partial C(x,y,w,b)}{\partial b^l_j}$,

  onde α é o *learning rate*.

O método iterativo parte do entendimento inicial do problema de que o custo C de treino de toda a amostra A pode ser dado por:

$$C(x,y,w,b) = \frac{1}{N_a}\left(\sum_{i=1}^{N_a} C^{(i)}(x,y,w,b)\right),$$

$$\textbf{onde}\quad C^{(i)}(x,y,w,b) = \sum_{j=1}^{n_L} C^{(i)}_j(x,y,w,b)$$

$$C^{(i)}_j(x,y,w,b) = \frac{1}{2}\left(a^L_j(i) - y^{(i)}_j\right)^2$$

$$a^L_j(i) = \sigma(z^l_j(i)),\ \textbf{onde}\ \sigma(t) = \frac{1}{1+e^{-t}}$$

$$z^l_j(i) = \sum_{k=1}^{n_{l-1}} w^l_{jk} a^{l-1}_k(i) + b^l_j,\ \textbf{para}\ l : 2..L$$

$$a^1_j(i) = x^{(i)}_j / F_j\ \textbf{para}\ j \in [1, n_1].$$

Finalmente, considerando que

$$\begin{aligned} C &\equiv C(x,y,w,b) \\ C^{(i)} &\equiv C^{(i)}(x,y,w,b) \\ C^{(i)}_j &\equiv C^{(i)}_j(x,y,w,b) \end{aligned}$$

tem-se que $C = \frac{1}{N_a}\left(\sum_{i=1}^{N_a} C^{(i)}\right)$,

onde $C^{(i)} = \sum_{j=1}^{n_L} C_j^{(i)}$ **e** $C_j^{(i)} = \frac{1}{2}\left(a_j^L(i) - y_j^{(i)}\right)^2$,

e deseja-se calcular as derivadas

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{1}{N_a}\sum_{i=1}^{N_a} \frac{\partial C^{(i)}}{\partial w_{jk}^l},$$

para $l\colon 2..L$, $j\colon 1..n_l$ **e** $k\colon 1..n_{l-1}$

$$\frac{\partial C}{\partial b_j^l} = \frac{1}{N_a}\sum_{i=1}^{N_a} \frac{\partial C^{(i)}}{\partial b_j^l}, \text{ para } l\colon 2..L \text{ e } j\colon 1..n_l.$$

pelo método *backward propagation*, o qual é uma forma de propagar para trás a saída de uma rede neural com o objetivo de ver a contribuição de cada nodo de rede para o resultado final nesse processo de atualizar os pesos das transações, dando aos nodos responsáveis por maior erro um menor peso, de forma a minimizar os erros de ajuste da função interpolada aos dados da amostra.

7.5 Formulação do Backpropagation

O erro na camada de saída para o item i da amostra é dado por: $C^{(i)} = \frac{1}{n_L}\sum_{j=1}^{n_L} C_j^{(i)}$

O método *backpropagation* calcula as derivadas parciais dessa função, a qual define a saída de uma rede neural em relação a seus pesos e *biases*, de tal forma que essas derivadas permitam identificar como alterações nesses pesos e *biases* modificam o custo ou erro na saída de rede para cada item da amostra.

Nesse sentido, *backpropagation* calcula os componentes do gradiente local, que são dados por

$$\delta_j^l(i) \equiv \partial C^{(i)}/\partial z_j^l(i),\ l{:}1..L \text{ e } j{:}1..n_l,$$

que é a medida do erro associado a todos os neurônios da rede para a i-ésima entrada da amostra.

Com os $\delta_j^l(i)$ computados, determinam-se as derivadas do custo de cada item individual i da amostra de treinamento. E a partir dessas derivadas, calculam-se as derivadas $\partial C/\partial w_{jk}^l$ e $\partial C/\partial b_j^l$ para toda amostra.

No cálculo das derivadas do custo para um item, considera-se que

$$z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$$
$$\frac{\partial z_j^l(i)}{\partial w_{jk}^l} = a_k^{l-1}(i) \textbf{ e } \frac{\partial z_j^l(i)}{\partial b_j^l} = 1$$

E as derivadas do custo para o i-ésimo item da amostra são:

$$\frac{\partial C^{(i)}}{\partial w_{jk}^l} = \underbrace{\frac{\partial C^{(i)}}{\partial z_j^l(i)}}_{\delta_j^l(i)} \underbrace{\frac{\partial z_j^l(i)}{\partial w_{jk}^l}}_{a_k^{l-1}} = a_k^{l-1}(i)\delta_j^l(i),$$

para $l{:}2..L$, $j{:}1..n_l$ **e** $k{:}1..n_{l-1}$

$$\frac{\partial C^{(i)}}{\partial b_j^l} = \underbrace{\frac{\partial C^{(i)}}{\partial z_j^l(i)}}_{\delta_j^l(i)} \underbrace{\frac{\partial z_j^l(i)}{\partial b_j^l}}_{1} = \delta_j^l(i),$$

para $l : 2..L$ **e** $j : 1..n_l$.

As derivadas do custo de toda a amostra relativas a pesos e *biases* são as médias das respectivas derivadas

obtidas com os exemplos individuais de treinamento, dadas por:

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{1}{N_a}\sum_{i=1}^{N_a} a^{l-1}_k(i)\delta^l_j(i),$$

$$\textbf{para } l : 2..L,\ j : 1..n_l \textbf{ e } k : 1..n_{l-1}$$

$$\frac{\partial C}{\partial b^l_j} = \frac{1}{N_a}\sum_{i=1}^{N_a} \delta^l_j(i), \textbf{ para } l : 2..L \textbf{ e } j : 1..n_l$$

onde

$$\delta^l_j(i) \equiv \frac{\partial C^{(i)}}{\partial z^l_j(i)}, \textbf{ para } l : 1..L \textbf{ e } j : 1..n_l$$

7.6 Formulação do *Forward Propagation*

Dada a amostra A de treinamento:

$$A = \left\{((x^{(i)}_1, \cdots, x^{(i)}_{n_1}), (y^{(i)}_1, \cdots, y^{(i)}_{n_L})), \text{for } i \in [1, N_a]\right\}$$

e considerando a rede

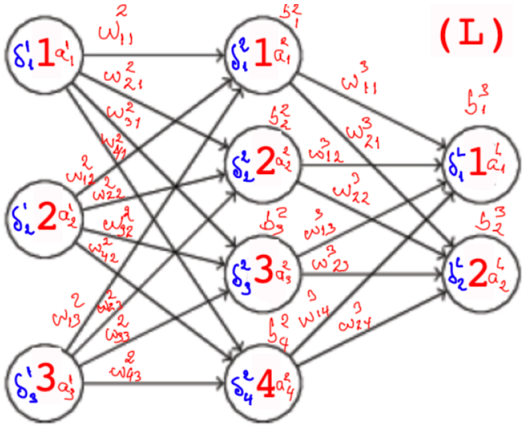


Figura 7.4: Forward Propagation

a ativação da entrada do item i é dada por:

$$a^1_j(i) = x^{(i)}_j / F_j,\ j : 1..n_1,$$

a ativação dos neurônios, **para** $l:2..L$ **e** $j:1..n_l$ por:

$a_j^l(i) = \sigma(z_j^l(i))$

$z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$

e o custo dos neurônios na saída do item i por:

$C_j^{(i)} = \frac{1}{2}\left(a_j^L(i) - y_j^{(i)}\right)^2$, **para** $j:1..n_L$.

Para a formulação de $\delta_j^L(i)$ da Camada de Saída, tem-se que, para $j:1..n_L$:

$C^{(i)} = \sum_{k=1}^{n_L} C_k^{(i)}$

$C_k^{(i)} = \frac{1}{2}\left(a_k^L(i) - y_k^{(i)}\right)^2$

$a_k^L(i) = \sigma(z_k^L(i))$

Então, o erro na saída j é:

$$\delta_j^L(i) = \frac{\partial C^{(i)}}{\partial z_j^L(i)} = \sum_{k=1}^{n_L} \frac{\partial C_k^{(i)}}{\partial a_k^L(i)} \frac{\partial a_k^L(i)}{\partial z_j^L(i)}$$

$$= \frac{\partial C_j^{(i)}}{\partial a_j^L(i)} \frac{\partial a_j^L(i)}{\partial z_j^L(i)} = \frac{\partial C_j^{(i)}}{\partial a_j^L(i)} \sigma'(z_j^L(i))$$

$$\implies \boxed{\delta_j^L(i) = (a_j^L(i) - y_j^{(i)})\ \sigma'(z_j^L(i))}$$

onde $\sigma'(t) = \sigma(t)(1 - \sigma(t))$

Similarmente, formula-se $\delta_j^l(i)$ das demais camadas, partindo-se de:

$C^{(i)} = \sum_{k=1}^{n_L} C_k^{(i)}$

$z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$, $l:2..L$ **e** $j:1..n_l$

E **para** $l:1..L-1$ **e** $k:1..n_l$, tem-se que:

$$z_k^{l+1}(i) = \sum_{j=1}^{n_l} w_{kj}^{l+1}\sigma(z_j^l(i)) + b_k^{l+1}$$

$$\frac{\partial z_k^{l+1}(i)}{\partial z_j^l(i)} = \underbrace{\cdots}_{0} + w_{kj}^{l+1}\sigma'(z_j^l(i)) + \underbrace{\cdots}_{0}$$

Então:

$$\delta_j^l(i) = \frac{\partial C^{(i)}}{\partial z_j^l(i)} = \sum_{k=1}^{n_L} \frac{\partial C_k^{(i)}}{\partial z_k^{l+1}(i)} \frac{\partial z_k^{l+1}(i)}{\partial z_j^l(i)}$$

$$= \sum_{k=1}^{n_L} \delta_k^{l+1}(i) \frac{\partial z_k^{l+1}(i)}{\partial z_j^l(i)}$$

$$\implies \boxed{\delta_j^l(i) = \sum_{k=1}^{n_L} w_{kj}^{l+1}\delta_k^{l+1}(i)\sigma'(z_j^l(i))}$$

onde $\sigma'(t) = \sigma(t)(1-\sigma(t))$

Para finalizar, computam-se a seguir as taxas de alteração de pesos e *biases*, onde para simplificar assume-se $\sigma(t) = t$, e tem-se:

$$\delta_j^L(i) = (a_j^L(i) - y_j^{(i)}), \textbf{ para } j : 1..n_L,$$

$$\delta_j^l(i) = \sum_{k=1}^{n_{l+1}} w_{kj}^{l+1}\delta_k^{l+1}(i),$$
$$\textbf{para } l : 1..L-1 \textbf{ e } j : 1..n_l$$

$$a_j^l(i) = \sigma(z_j^l(i)), \text{ para } l : 2..L \text{ e } j : 1..n_l,$$
$$\textbf{onde } z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$$

Pode-se então calcular as derivadas desejadas:

$$\frac{\partial C^{(i)}}{\partial b_j^l} = \delta_j^l(i), \textbf{ para } l : 1..L \textbf{ e } j : 1..n_l$$

$$\frac{\partial C^{(i)}}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l(i), \textbf{ para } l : 2..L, \; k : 1..n_{l-1} \textbf{ e}$$
$$j : 1..n_l$$

7.7 Erro do Neurônio j da Camada l

A expressão $\delta_j^l(i)$ designa o elemento de δ que dá o erro do neurônio j da camada l para o item i da amostra. Para determinar sua fórmula, considera-se o item i da amostra A e a rede da Fig. 7.5:

$$A = \left\{((x_1^{(i)}, \cdots, x_{n_1}^{(i)}), (y_1^{(i)}, \cdots, y_{n_L}^{(i)})), \textbf{ para } i \in [1, N_a]\right\}$$

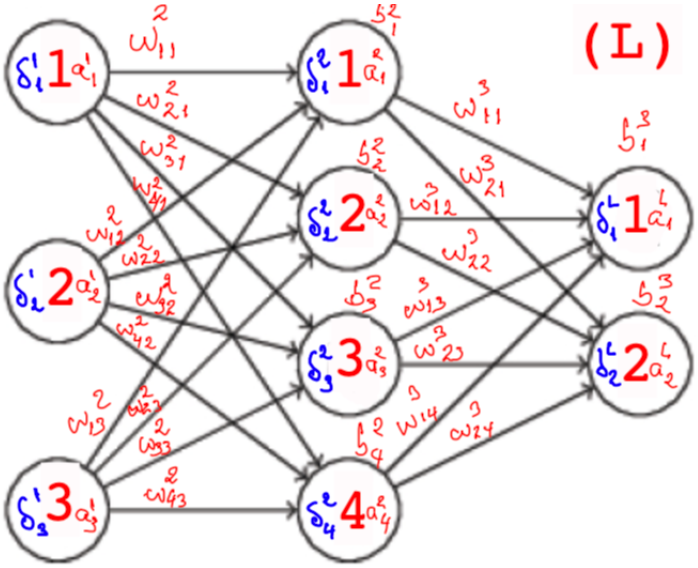


Figura 7.5: Forward and Backward

E a decorrente formulação:

$$a_j^1(i) = x_j^{(i)}/F_j,\ j : 1..n_1$$
$$\delta_j^L(i) = a_j^L(i) - y_j^{(i)},\ j : 1..n_L$$

para $l : 2..L$ **e** $j : 1..n_l$,

$$a_j^l(i) = \sigma(z_j^l(i))$$
$$z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$$

E **para** $l : L-1..1$ **e** $j : 1..n_l$

$$\delta_j^l(i) = \sum_{k=1}^{n_l} w_{jk}^{l+1} \delta_j^{l+1}(i)$$

Portanto,

$$\boxed{\frac{\partial C^{(i)}}{\partial b_j^l} = \delta_j^l(i)} \text{ e } \boxed{\frac{\partial C^{(i)}}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l(i)}$$

Assim, pode-se calcular o erro $\delta_j^l(i)$ desejado, sendo o custo do treinamento do item i:

$C^{(i)} = \sum_{k=1}^{n_L} C_k^{(i)}$

E **para** $j : 1..n_L$, tem-se:

$$C_k^{(i)} = \frac{1}{2}\left(a_k^L(i) - y_k^{(i)}\right)^2$$

$a_k^L(i) = \sigma(z_k^L(i))$, onde $\sigma(t) = t$

$$\delta_j^L(i) = \frac{\partial C_j^{(i)}}{\partial z_j^L(i)} = \sum_{k=1}^{n_L} \frac{\partial C_k^{(i)}}{\partial a_k^L(i)} \frac{\partial a_k^L(i)}{\partial z_j^L(i)}$$

$$= \frac{\partial C_j^{(i)}}{\partial a_j^L(i)} \frac{\partial a_j^L(i)}{\partial z_j^L(i)}$$

$$= \frac{\partial C_j^{(i)}}{\partial a_j^L(i)} \sigma'(z_j^L(i))$$

Portanto, tem-se o erro na saída da rede:

$$\boxed{\delta_j^L(i) = (a_j^L(i) - y_j^{(i)})\sigma'(z_j^L(i)}$$

Para os erros dos demais neurônios, tem-se

para $l : 1..L$ **e** $j : 1..n_l$:

$$a_j^l(i) = \sigma(z_j^l(i))$$

$$z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$$

para $l : 1..L-1$ **e** $j : 1..n_l$:

$$z_k^{l+1}(i) = \sum_{j=1}^{n_l} w_{kj}^{l+1} a_j^l(i) + b_j^{l+1}$$

$$= \sum_{j=1}^{n_l} w_{kj}^{l+1} \sigma(z_j^l(i)) + b_k^{l+1}$$

Finalmente, o erro é dado por

$$\delta_j^l(i) = \frac{\partial C_j^{(i)}}{\partial z_j^l(i)} = \sum_{k=1}^{n_{l+1}} \frac{\partial C_k^{(i)}}{\partial z_k^{l+1}(i)} \frac{\partial z_k^{l+1}(i)}{\partial z_j^l(i)}$$

$$= \sum_{k=1}^{n_{l+1}} \delta_k^{l+1}(i) \frac{\partial z_k^{l+1}(i)}{\partial z_j^l(i)}$$

$$\frac{\partial z_k^{l+1}(i)}{\partial z_j^l(i)} = w_{jk}^{l+1} \sigma'(z_j^l(i))$$

$$\boxed{\delta_j^l(i) = \sum_{k=1}^{n_{l+1}} w_{kj}^{l+1} \delta_k^{l+1}(i) \sigma'(z_j^l(i))}$$

7.8 Ajuste do Custo Mínimo

Para o ajuste com erro mínimo, executa-se o seguinte algoritmo de treinamento de uma rede neural:

1. defina a rede neural $R = (L, n, w, b)$
2. compute fator F de normalização das *features*
3. compute matrizes a e δ
4. compute o custo $C(x, y, w, b)$
5. repita os passos abaixo até $C(x, y, w, b)$ convergir:
   - calcule a matriz $\partial C / \partial b$
   - calcule a matriz $\partial C / \partial w$
   - compute os ajustes dos pesos e *biases*
   - recompute a e δ
   - recompute $C(x, y, w, b)$

7.9 Glossário

A seguinte lista, em quase ordem alfabética, introduz a notação extensamente usada na formulação das bases dos algoritmos de treinamento de redes neurais densas:

- A: amostra de treinamento
- a: matriz tridimensional das ativações dos neurônios da rede para toda a amostra
- $a(i)$: matriz bidimensional das ativações dos neurônios da rede para o item i da amostra
- $a_j^l(i)$: elemento de a denotando o valor da ativação do neurônio j da camada l para o item i de A
- b: matriz bidimensional de *biases* b_j^l, que dá a medida da propensão de o neurônio produzir saída ativa
- b_j^l: valor do *bias* do neurônio j da camada $l \in [2, L]$
- C: função de custo que fornece o erro de treinamento
- *epoch*: uma passagem por toda a amostra de treinamento
- F: vetor de tamanho n_1 com fatores para normalizar as entradas da rede neural, i.e.,
$$a_j^1(i) = x_j^{(i)}/F_j,\ \forall i : 1..N_a \text{ e } j : 1..n_1$$
- L: número de camadas da rede neural
- (L, n, w, b): rede de neurônios de L camadas, com n_l neurônios na camada $l \in [1, L]$, matriz de pesos w, matriz de *biases* b
- n: vetor contendo o número de neurônios em cada

camada

- n_l: número de neurônios da camada l
- N_a: tamanho da amostra a ser usada no treinamento
- N_r: número de neurônios da rede neural
- $T_a(Na, N_r)$: custo do cálculo da matriz a
- $T_c(Na, N_r)$: custo do cálculo do erro de aprendizado
- $T_\delta(Na, N_r)$: custo do cálculo da matriz δ
- $T_{\partial b}(Na, N_r)$: custo de $\partial C/\partial w$
- $T_{\partial w}(Na, N_r)$: custo de $\partial C/\partial w$a
- $T_f(Na, N_r)$: custo do fator de normalização
- $T_p(Na, N_r)$: custo dos ajustes de pesos e *biases*
- $T_t(Na, N_r)$: custo final do aprendizado
- w: matriz tridimensional de pesos w^l_{jk}
- w^l_{jk}: peso da conexão do neurônio k da camada $l-1$ ao j da camada $l \in [2, L]$
- threshold: valor limite acima do qual gera-se sinapse
- $x^{(i)}$: i-ésima *feature* ou atributo de entrada
  $(x^{(i)}_1, x^{(i)}_2, \cdots, x^{(i)}_{n_1})$
  da amostra de treinamento
- $(x^{(i)}, y^{(i)})$: i-ésimo item da amostra de treinamento, $1 \leq i \leq N_a$
- $y^{(i)}$: resultado observado para a entrada $x^{(i)}$
- α: *learning rate* usada no algoritmo de ajuste de pesos e *biases*
- δ: matriz tridimensional de erros dos neurônios, i.e., de desvios entre o valor esperado e o calculado pela rede neural para toda a amostra

- $\delta(i)$: matriz bidimensional de erros dos neurônios, i.e., de desvios entre o valor esperado e o calculado pela rede neural para item i da amostra
- $\delta_j^l(i)$: elemento de δ denotando o erro do neurônio j da camada l para o item i da amostra
- $\partial C/\partial w$: matriz tridimensional contendo as médias das derivadas em relação aos pesos das conexões da rede neural, sendo essas derivadas relativas a cada item da amostra
- $\partial C/\partial w_{jk}^l$: elemento da matriz $\partial C/\partial w$ contendo o valor médio das derivadas em relação ao peso w_{jk}^l da conexão entre o neurônio k da camada $l-1$ ao neurônio j da camada l da rede neural
- $\partial C^{(i)}/\partial w$: matriz tridimensional contendo as derivadas em relação aos pesos das conexões da rede neural no treinamento com o item i da amostra
- $\partial C^{(i)}/\partial w_{jk}^l$: elemento da matriz $\partial C^{(i)}/\partial w$ contendo o valor da derivada em relação ao peso w_{jk}^l da conexão entre o neurônio k da camada l–1 ao neurônio j da camada l da rede neural, considerando apenas o item i da amostra
- $\partial C/\partial b$: matriz bidimensional contendo a média das derivadas em relação aos *biases* dos neurônios de cada item i a amostra
- $\partial C/\partial b_j^l$: elemento da matriz bidimensional $\partial C/\partial b$ contendo o valor médio das derivadas do custo em

relação ao *bias* do neurônio j da camada l considerando o treinamento de toda a amostra

- $\partial C^{(i)}/\partial b$: matriz bidimensional contendo as derivadas em relação aos *biases* dos neurônios durante o treinamento com item i
- $\partial C^{(i)}/\partial b^l_j$: elemento da matriz $\partial C^{(i)}/\partial b$ contendo o valor da derivada em relação ao *bias* do neurônio j da camada l da rede neural
- σ: função sigmoid ou logística $\sigma(t) = \frac{1}{1+e^{-t}}$
- σ': derivada logística $\sigma'(t) = \sigma(t)(1 - \sigma(t))$

Leitura Recomentada

- Hal Daumé III, A Course in Machine Learning, disponível na Internet, Version 0.8, 189 páginas, August 2012.
- Michael A. Nielsen, Neural Networks and Deep Learning, on-line book, 395 páginas, 2015.

***

Capítulo 8

Validação da Amostra

Uma boa amostra de treinamento é aquela que é representativa de toda a população de dados, e se a distribuição dos dados da população for dinâmica, novas amostras devem ser organizadas periodicamente, e o aprendizado submetido a retreinamentos.

Uma distribuição de dados de uma amostra é uma função ou uma tabela que mostra todos os valores (ou intervalos) possíveis dos dados e sua frequência de ocorrências no conjunto considerado.

A amostra de validação, que se destina apenas para avaliar a capacidade de generalização do algoritmo usado, deve ser distinta da amostra de treinamento, e ambas devem seguir a mesma distribuição de probabilidades.

Para a validação da amostra ou do aprendizado usa-se uma técnica, cujos procedimentos básicos consiste em retirar da amostra dada uma subamostra para teste, fazer o devido treinamento com a amostra resultante, avaliar o treinamento usando a subamostra de teste e deve-se repetir esse processo com diferentes subamos-

tras de teste, até que o erro seja mínimo.

8.1 Tipos de Validação

A validação da amostra de treinamento pode ser feita por processos denominados *Holdout*, *K-Fold*, *Stratified K-Fold* e *Leave-P-Out*.

No *Holdout*, remove-se da amostra uma subamostra de tamanho típico de 15% da amostra de treinamento e faz-se uma validação.

Com *K-Fold*, particiona-se a amostra de treinamento em k subconjuntos de teste, aplica o *holdout* para cada um deles e considera-se o erro final apurado a média desses k testes.

No *Stratified K-Fold*, removem-se subamostras de teste que tenham o mesmo perfil da amostra de treinamento e faz-se uma validação.

E com *Leave-P-Out*, retiram-se subamostras da amostra treinamento e faz-se o treinamento seguido de sua validação com essas subamostras.

8.2 Underfitting e Overfitting

O algoritmo de aprendizado a partir de uma amostra de dados deve ser encerrado quando ocorrer uma das seguintes situações:

- *Just Right*: quando os erros entre o esperado e o computado são mínimos e o método é generalizável.
- *Underfitting*: quando os erros nos resultados computados ainda são insatisfatórios, mas não se consegue reduzi-los.
- *Overfitting*: quando o aprendizado gera algoritmo que produz resultados precisos para dados da amostra, mas é incapaz de generalizar para novos dados.

Se o modelo usado ajusta-se igualmente bem com as amostras de treinamento e de validação, espera-se um *overfitting* mínimo. E no caso de um baixo desempenho do algoritmo de *machine learning* tem-se *overfitting* ou *underfitting* dos dados ao modelo apresentado. Por outro lado, situações em que há um bom ajuste da amostra de treinamento e um mau ajuste de validação apontam para *overfitting*.

Em resumo, *underfitting* refere-se a um modelo que nem modela os dados da amostra de treinamento nem permite generalização para novos dados e *underfitting* é uma indicação clara que o algoritmo de aprendizado não serve. A solução é buscar algoritmos alternativos.

Overfitting ocorre quando ruídos ou flutuações dos dados da amostra impactam negativamente o desempenho do modelo com novos dados que não trazem os mesmos detalhes ou ruídos, ou seja, ruídos e flutuações particulares de uma dada amostra de treinamento são

erroneamente tratados pelo algoritmo como conceitos relevantes e assim afetam a distribuição dos dados.

Esse problema surge porque esses conceitos normalmente não aparecem nos novos dados e assim impactam negativamente o processo de generalização.

Overfitting é o maior problema em rede neurais com um número elevado de pesos e *biases*. E isso demanda um mecanismo para detectá-lo e ter seu efeito reduzido, por exemplo, se os erros de aprendizado da amostra para de reduzir, o treinamento deve ser encerrado.

8.3 Regularização

Pode-se evitar *overfitting* aumentando o tamanho da amostra, reduzindo o tamanho da rede, escolhendo uma melhor função de custo, inicializando com mais cuidados os pesos da rede, aplicando heurísticas para escolha de bons hiperparâmetros, como taxa de aprendizado, etc, ou então aplicando técnicas de regularização.

Reduzir tamanho da rede ou da amostra é indesejável, pois redes maiores são mais poderosas, e amostra maior fornece mais precisão.

Uma boa técnica de regularização deve permitir reduzir *overfitting* sem afetar o tamanho da rede ou da amostra. As principais técnicas são denominadas Regularização L1, Regularização L2 e *Dropout*.

Com a Regularização L1, adiciona-se a parcela desta-

cada à função de custo: $C = C_0 + \frac{\lambda}{N_a}\sum_w |w|$, **onde** λ tem o papel de definir preferências, sendo que um pequeno $\lambda > 0$ implica que o custo mínimo depende majoritariamente de C_0 e que com um grande λ, o custo mínimo passa a depender prioritariamente dos valores dos pesos w.

Intuitivamente, o efeito da regularização é controlar, via o parâmetro de regularização λ, a preferência por pesos menores. Assim, regularização é um compromisso entre o uso de pesos de pequenos valores, que geram melhores resultados, mas necessitam elevado processamento, e a qualidade de minimização da função de custo.

Na Regularização L2, também adiciona-se uma parcela à função de custo: $C = C_0 + \frac{\lambda}{2N_a}\sum_w w^2$, onde o parâmetro $\lambda > 0$ tem o mesmo significado que em L1.

Adicionalmente, observa-se que quando $\Delta w > 1$, tem-se que $|\Delta w| < \Delta w^2$, e isso implica que a afinação com L2 encolhe os pesos mais rapidamente que L1, e quando $\Delta w < 1$, tem-se que $|\Delta w| > \Delta w^2$, e a afinação com L1 encolhe os pesos mais rapidamente que L2.

A regularização *dropout* inicia-se apagando aleatória e temporariamente a metade dos neurônios de camadas ocultas e realizando *forward* e *backward propagations*. Em seguida, atualizam-se os pesos e *biases* com os gradientes computados e restauram-se os neurônios apagados, reavaliando o custo da rede.

O processo é então repetido até que se obtenham pesos e *bias* satisfatórios.

Dropout assegura funcionamento mesmo com falta de dados!!!

Leitura Recomentada

- Hal Daumé III, A Course in Machine Learning, disponível na Internet, Version 0.8, 189 páginas, August 2012.
- Michael A. Nielsen, Neural Networks and Deep Learning, on-line book, 395 páginas, 2015.

***

Capítulo 9

Epílogo

Deep Learning trouxe uma enorme contribuição para a área de Inteligência Artificial. Entretanto, ainda há um custo muito alto para isso, porque para funcionar adequadamente, necessita-se uma imensa massa de dados e alta capacidade de processamento.

Isso explica porque *deep learning* somente tornou-se popular agora, quando o poder computacional expandiu-se expressivamente.

Reduzir a dependência de *deep learning* em grandes massas de dados é talvez uma das prioridades das atuais pesquisa em IA.

***

Referências Bibliográficas

[1] Daniel Adiwardana, Towards Conversational Agent that Can Chat about ··· Anything, https://ai.googleblog.com/2020/01/towards-conversational-agent-that-can.html, Google AIBlog, posted on january, 28, 2020, acessado em 06/02/2020.

[2] Jason Brownlee,Overfitting and Underfitting with Machine Learning Algorithms,Internet Post, 2019.

[3] Jason Brownlee, Parametric and Nonparametric Machine Learning Algorithms, Internet Post, 2019.

[4] Frederico Ferreira Campos, filho, Algoritmos Numéricos, Livros Técnicos e Científicos, Editora S.A., 383 páginas, 2001.

[5] Louis Augustin Cauchy, Méthode Générale pour la Résolution des Systèms d'équations simultanées, Compte Rendu à l'Academie de Sciences Paris, 25:536-538, 1847 [citado por Claude Lemaréchal].

[6] Luiz Ferrara de Almeida Cunha, Comunicação Pessoal, jan/2020.

[7] Hal Daumé III, A Course in Machine Learning, disponível na Internet, Version 0.8, 189 páginas, August 2012.

[8] Marc Peter Deisenroth, A. Aldo Faisal and Cheng Soon Ong, Mathematics for Machine Learning, 398 pages, Cambridge University Press, 2020.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press, 799 páginas, 2016.

[10] Claude Lemaréchal, Cauchy and the Gradient Descent Method, Documenta Mathematica, Extra Volume ISMP, 251-254, 2012.

[11] Kevin P. Murphy,Machine Learning - A Probabilistic Perspective, The MIT Press, Cambridge-Massachusetts, London-England, 1098 páginas, 2012 .

[12] Michael A. Nielsen, Neural Networks and Deep Learning, on-line book, 395 páginas, 2015. $\Longleftarrow$ recomendado.

[13] Andrew Ng, Machine Learning Yearning - draft, 118 páginas, free book, 2018. $\Longleftarrow$ recomendado.

[14] Joseph C. Pitt, Chapter II: Rules of Inference, Induction, and Ampliative Frameworks, in Book: Pictures, Images, and Conceptual Change, 24

páginas, D. Reid Publishing Company, Dorrecht, Holland, 1981.

[15] David E. Rumelhart, Geoffrey E. Hilton, Ronald J. Williams,Learning Representation by Back-Propagating Errors, Nature, vol 323, pp 533-536, 9/october/1986. ⟸ recomendado.

[16] Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, 946 páginas, Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[17] Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning - From Theory to Algorithms, Cambridge University Press, free pdf version, 449 páginas, 2014.

[18] Wikipedia,Inference, https://en.wikipedia.org/wiki/Inference, acesso em 11/janeiro/2020.

[19] Wikipedia, Training, Validation, and Test Sets, https://en.wikipedia.org/wiki/Training,_validation_and_test_sets, acesso em 01/fevereiro/2020.

[20] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, Dive Into Deep Learning, pdf Release 0.7.1, 912 páginas, jan 25, 2020. ⟸ recomendado.


fim