Fundamentos de Aprendizagem de Máquina

Roberto S. Bigonha

Belo Horizonte

2020

FUNDAMENTOS DE APRENDIZAGEM DE MÁQUINA

VERSÃO 5.0

Roberto da Silva Bigonha

UFMG

13 de maio de 2020
08:41

All rigths reserved

Belo Horizonte - Minas Gerais

SUMÁRIO

- Fontes de Consulta e Referências 2
- Introdução 8
- Silogismo de Aristóteles 16
- Revisão de Matemática 25
- Regressão Linear 38
  - Solução Algébrica 40
  - Solução Iterativa 45
- Aprendizagem de Máquina 65
- Redes Neurais Densas 75
  - Neurônios Perceptrons 79
  - Neurônios Sigmoids 85
- Função Modelada pela Rede 91
- Deep Learning 103
- Treinamento de Redes Densas 109
- Análise de Complexidade 142
- Cross-Validation 161
- Regularização 171
- Redes Neurais Convolucionais 177
- Recurrent Networks 197
- Ferramentas 199
- Conclusões 201

FONTE DE CONSULTAS E REFERÊNCIAS

Bibliografia Recomendada

1. David E. Rumelhart, Geoffrey E. Hilton, Ronald J. Williams, Learning Representation by Back-Propagating Errors, Nature, vol 323, pp 533-536, 9/october/1986. $\Longleftarrow$ recomendado

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

3. 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. $\Longleftarrow$ recomendado

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

Bibliografia Adicional...

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

6. Daniel Adiwardana, Towards Conversational Agent that Can Chat about$\cdots$ 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.

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

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

...Bibliografia Adicional...

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

10. Jason Brownlee, Overfitting and Underfitting with Machine Learning Algorithms, Internet Post, 2019.

11. Jason Brownlee, Parametric and Nonparametric Machine Learning Algorithms, Internet Post, 2019.

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

...Bibliografia Adicional...

13. 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.

14. 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].

15. Luiz Ferrara de Almeida Cunha, Comunicaçao Pessoal, jan/2020.

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

...Bibliografia Adicional

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

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

20. Wikipedia, Training, Validation, and Test Sets, https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets, acesso em 01/fevereiro/2020.

INTRODUÇÃO

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

Principais Áreas de Pesquisa em IA

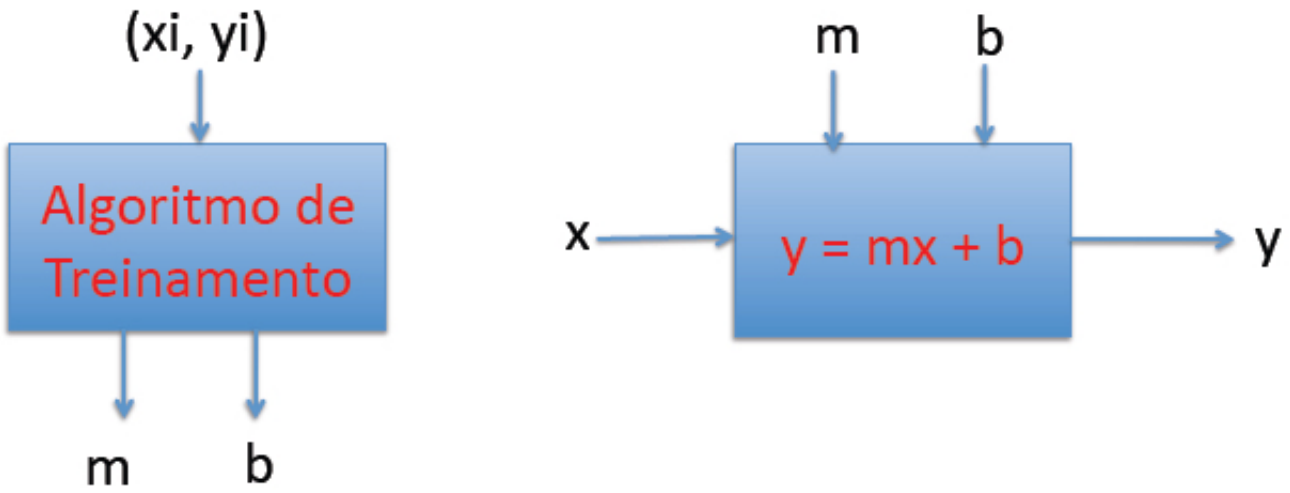
- Machine Learning
- Expert Systems
- Computer Vision
- Natural Language Processing
- Robotics

Machine Learning

- Aprendizagem 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
- Algoritmos de aprendizagem produzem programas a partir de exemplos que especificam as saídas 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
- A ideia é capacitar computadores a aprender e interpretar padrões em imagens, sons e outros dados estruturados

Um Método Simples de Treinamento

- **Amostra para treinamento:** $\{(x_i, y_i),$ **para** $1 \leq i \leq N_a\}$
- **Máquina treinada: para um novo** x**, sugere** y**, dado por** $y = h(x)$
- **Suposição para o treinamento:** $h(x) = mx + b$
- **Graficamente:**



- **Quando** y **é contínuo** $\Longrightarrow$ **regressão**
- **Quando** y **é discreto** $\Longrightarrow$ **regressão logística (classificação)**
- **Processo denominado** Aprendizado de Máquina

Tipos de Aprendizado de Máquina

Aprendizado de Máquina é subdividido em quatro tipos:

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

Técnicas para Aprendizado de Máquina

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

Palavras-Chave

- backpropagation
- bias
- convolution
- cross-validation
- deep learning
- dense
- forwardpropagation
- 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

SILOGISMO DE ARISTÓTELES

Silogismo

- **Silogismo [Michaelis]:**
  Segundo Aristóteles, silogismo é uma forma de raciocínio que consiste em duas proposições: a primeira, chamada premissa maior, a segunda, premissa menor, e a conclusão.
- **Admitida a coerência das premissas, a conclusão se infere da maior por intermédio da menor**
- **Por exemplo:**
  - **todos os homens são mortais (premissa maior)**
  - **eu sou um homem (premissa menor)**
  - **logo, eu sou mortal (conclusão)**

Conceito de Inferência

- Inferência são passos do raciocínio que mapeiam premissas em consequências lógicas
- Inferência é classificada em dedução e indução (Aristóteles-300AC)
- Dedução:
  - conclusões lógicas verdadeiras são derivadas de premissas conhecidas e verdadeiras, usando regras de inferência válidas
  - o raciocínio parte do geral para derivar um resultado mais específico
- Indução:
  - conclusões são derivadas de premissas particulares
  - conclusões não necessariamente demostradas verdadeiras
  - o raciocínio parte do específico para derivar um resultado mais geral

Dedução X Indução

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

- **Em indução,**
  - a conclusão deriva de premissas empiricamente, portanto não há garantia de que seja verdadeira em todos os casos
  - a conclusão não é logicamente válida, há apenas uma **probabilidade** que seja correta para a maioria dos casos
  - validade em todos os casos exige que se tenha informação de todos eles

Exemplo de Dedução X Indução

- **Exemplo de Dedução:**
  - todos os homens são mortais (premissa maior) $\Leftarrow$ geral
  - Aristóteles é um homem (premissa menor)
  - logo, Aristóteles é mortal (conclusão) $\Leftarrow$ específico
- **Exemplo de Indução:**
  - Aristóteles é mortal (premissa maior) $\Leftarrow$ específico
  - Aristóteles é um homem (premissa menor)
  - logo, todos os homens são mortais (conclusão) $\Leftarrow$ geral

Dedução X Indução X IA

- **Processo Dedutivo (deductive inference):**
  Raciocinando dentro da Teoria $\Longrightarrow$ Conclusão Específica
  Axiomas e Regras de Inferência (Lógica Matemática):
  - **regras de Inferência: pai(A,B) $\wedge$ pai(B,C) $\Longrightarrow$ neto(C,A)**
  - **axiomas: pai(antônio, josé) $\wedge$ pai(josé, pedro) $\wedge$ pai(josé, joão)**
  - **dedução: infere-se que netos de antônio são {pedro, joão}**

- **Processo Indutivo (inductive inference):**
  Conhecimento Específico $\Longrightarrow$ Conclusão Geral
  Massa de Dados and Resultados Associados (treinamento):
  - **amostra (conhecimento específico): $\{(x_i, y_i)$, para $1 \leq i \leq N_a\}$**
  - **indução: infere-se um padrão que permite prever o valor aproximado de y dado x**

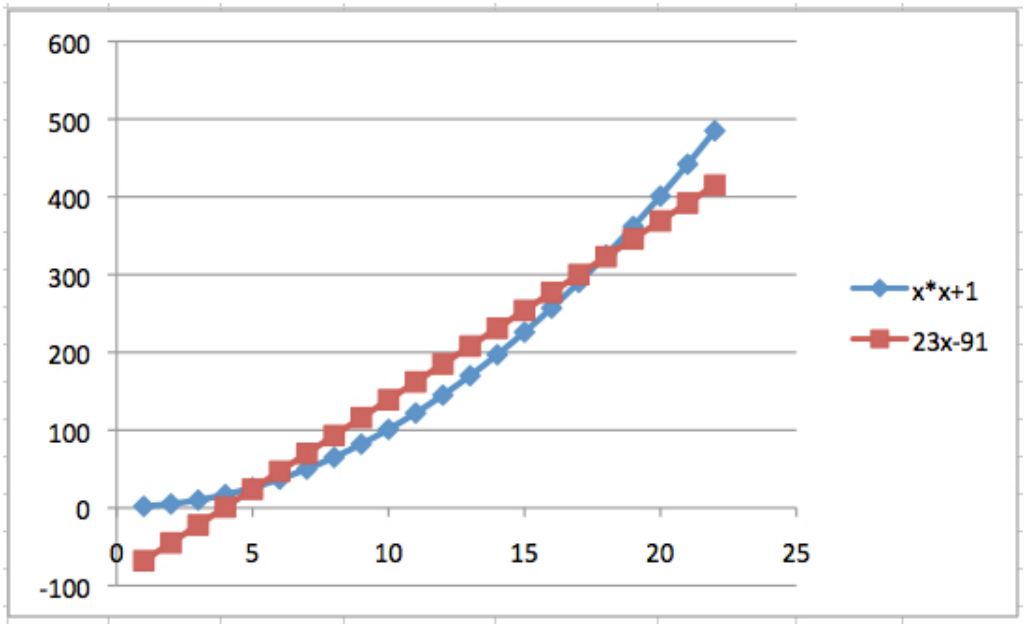
Exemplo Elementar de Indução

- Premissa maior (conhecimento específico): tabela que relaciona valores de x a de y, $1 \leq x \leq 22$
- Premissa menor: há uma reta $y = mx + b$ que generaliza esse mapeamento
- A partir dessas premissas, conclui-se que o melhor modelo, usando o método do Mínimo dos Quadrados, é a reta $y = 23x - 91$
- Ou seja, a partir da premissa maior, a tabela (x, y), e por intermédio da premissa menor de que o mapeamento de x a y é linear, induz-se a conclusão de que esse mapeamento é modelado pela reta $y = 23x - 91$, para todo x
- Observe que a correção dessa conclusão depende da validade das premissas nas quais o raciocínio foi baseado

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

Indução e Aprendizado de Máquina

- Em termos do Aprendizado de Máquina, o processo de determinação de m e b de $y = mx + b$ é chamado de treinamento
- Baseando-se na tabela dada, o algoritmo de treinamento induz a função $y = 23x - 91$, que mapeia novos x aos y correspondentes



x	y	z	|z-y|
	x*x+1	23x-91	Erro (%)
1	2	-68,0	3.500,0
2	5	-45,0	1.000,0
3	10	-22,0	320,0
4	17	1,0	94,1
5	26	24,0	7,7
6	37	47,0	27,0
7	50	70,0	40,0
8	65	93,0	43,1
9	82	116,0	41,5
10	101	139,0	37,6
11	122	162,0	32,8
12	145	185,0	27,6
13	170	208,0	22,4
14	197	231,0	17,3
15	226	254,0	12,4
16	257	277,0	7,8
17	290	300,0	3,4
18	325	323,0	0,6
19	362	346,0	4,4
20	401	369,0	8,0
21	442	392,0	11,3
22	485	415,0	14,4

Silogismo do Treinamento

- 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 features x_i a labels y_i
- A premissa menor, aceita como verdadeira, embora sem provas, é a da existência de uma função geral $y = f(x)$ que mapeia features a labels.
- 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 label associado a cada feature, dentro ou fora da amostra.
- De acordo com Aristóteles, trata-se de uma inferência por indução

REVISÃO DE MATEMÁTICA

Vetores e Matrizes

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

- **vetor: 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**
  Exemplo: $x_i \in \mathbb{N}$, $1 \leq i \leq n$, denota o i-ésimo elemento do vetor $x \in \mathbb{N}^n$

- **matriz: arranjo bidimensional de escalares, identificado por uma letra maiúscula, sendo cada elemento identificado pelo nome da matriz com dois índices**
  Exemplo: $A_{i,j} \in \mathbb{R}$, para $1 \leq i \leq m$ e $1 \leq j \leq n$ denota o elemento da linha i e coluna j da matriz $A \in \mathbb{R}^{mXn}$

Tensores

- **tensor: um arranjo de escalares com mais de duas dimensões, sendo seus elementos indexados conforme cada dimensão**
  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** $T \in \mathbb{N}^{mXnXp}$

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

Mean Squared Error

- **MSE (Mean Squared Error): 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**

- **MSE é uma medida da qualidade da estimativa, quanto menor melhor**

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 $j : 1..n$, é uma função diferenciável.**

  Então

  u **é uma função de** $t_1, t_2, \cdots, t_m$

  e

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

Gradiente ∇

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

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

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
- 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
  - função densidade de probabilidade

Função Massa de Probabilidade $P(x)$

- **Função $P(x)$ de massa de probabilidade descreve a distribuição de variáveis aleatórias discretas**

- **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. $\sum_{x \in X} P(x) = 1$

Função Densidade de Probabilidade $p(x)$

- Função $p(x)$ de densidade de probabilidade descreve a distribuição de variáveis aleatórias contínuas

- 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 de massa $P(x)$ de x estar no intervalo $[a, b]$: $\int_a^b p(x)dx$

Conceito de Convolução

- **Em Matemática, convolução é uma operação que combina duas funções, permitindo medir a soma do produto de uma função pela outra deslocada e invertida.**

- **A convolução será discreta ou contínua conforme assim seja o domínio das duas funções envolvidas**

- **Representa-se por $(f * g)(t)$ a convolução das funções f e g**

Convolução Discreta

- **Sejam** $f, g \in [0, n-1] \to \mathbb{R}$
- $h(t) = (f * g)(t)$ **representa a convolução de** f **e** g
- $h(t) = \sum_{i=0}^{t} f(i)g(t-i)$, **para** $t \in \mathbb{Z}$ **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**

Convolução Contínua

- **Sejam** $f, g \in \mathbb{R} \rightarrow \mathbb{R}$ **são funções integráveis**
- $h(x) = (f * g)(x)$ **representa a convolução de** f **e** g
- $h(x) = \int_{-\infty}^{\infty} f(u)g(x-u)du$, **para** $x, u \in \mathbb{Z}$

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

- Suponha que o sensor não seja 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

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

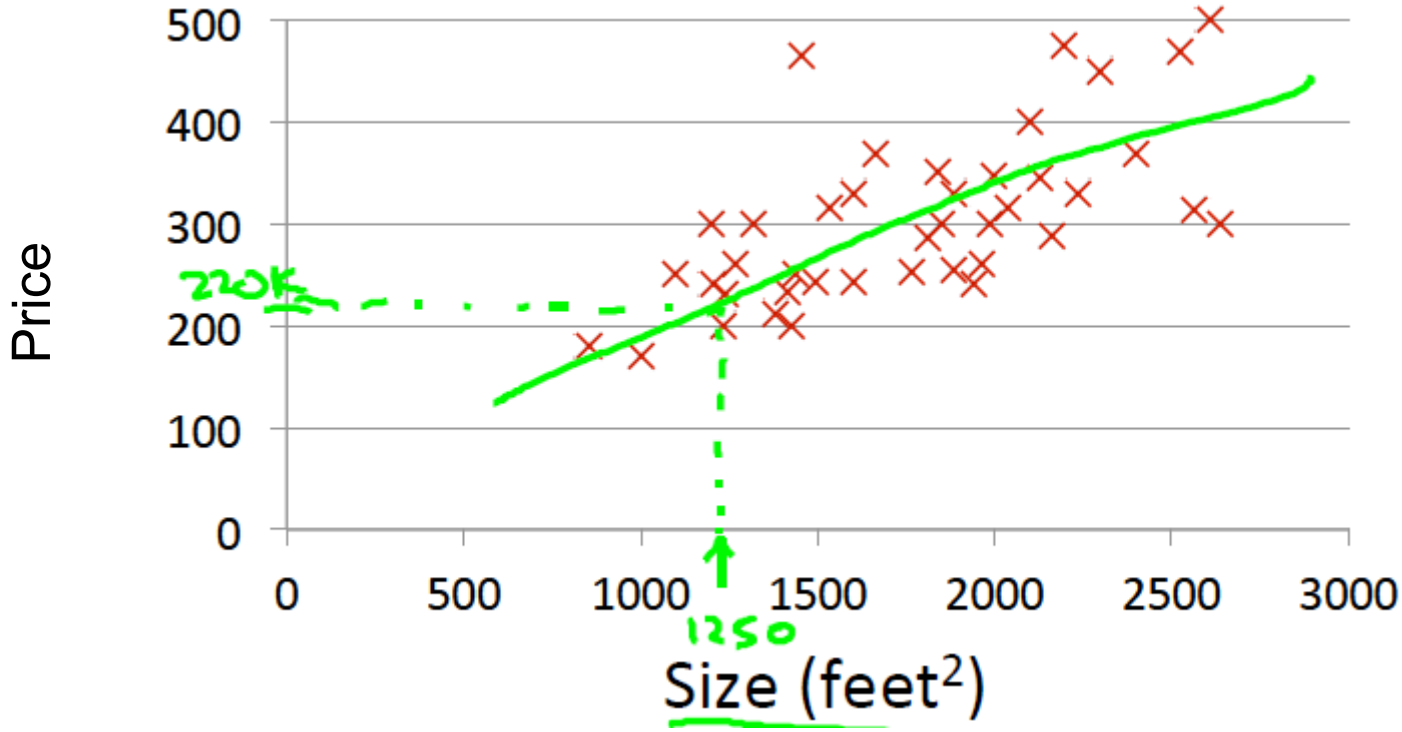
- Assim, a posição estimada da espaçonave:

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

REGRESSÃO LINEAR

Exemplo de Treinamento: Uso de Regressão

- **Amostra para treinamento:** $\{(Size_i, Price_i),$ **para** $1 \leq i \leq N_a\}$
- **Resultado do treinamento:** $price = h(size)$, **onde** $h(x) = mx + b$
- **Graficamente:**



$$y = mx + b$$

SOLUÇÃO ALGÉBRICA DA REGRESSÃO

Mínimo dos Quadrados (Gauss-1795, Legendre-1805)

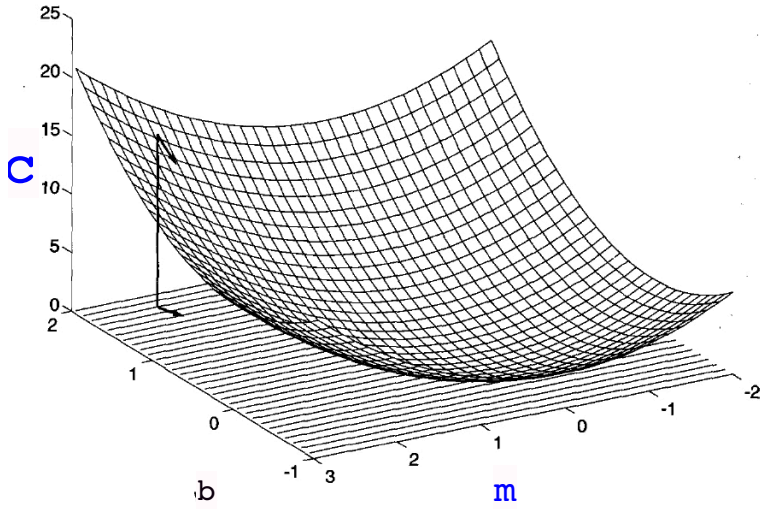
- **Amostra para treinamento:** $\{(x_i, y_i)$ **para** $1 \leq i \leq N_a\}$
- **Defina a forma da função a ser ajustada à amostra, e.g.:**

$$h(x) = mx + b$$

- **Indicador** C **de erro de ajustamento (MSE) em relação à amostra:**

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

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



- **Os pesos** m **e** b **devem ser calculados para definir custo** C **mínimo**
- **Esses pesos podem ser determinados analiticamente ou iterativamente**

Determinação Analítica do C Mínimo

- **Erro C de ajustamento em relação à amostra:**

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

- **Pesos m e b calculados para definir mínimo de C:**

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

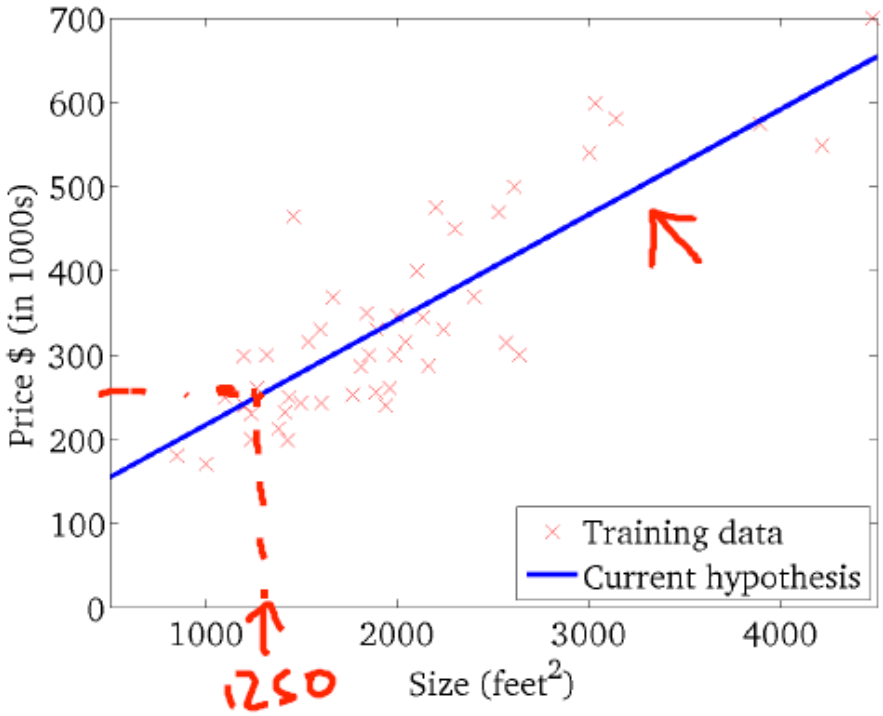
$$\Longrightarrow \quad \boxed{m = \frac{(\sum_{i=1}^{N_a} x_i)(\sum_{i=1}^{N_a} y_i) - N_a(\sum_{i=1}^{N_a} x_i y_i)}{(\sum_{i=1}^{N_a} x_i)^2 - N_a(\sum_{i=1}^{N_a} x_i^2)}}$$

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

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

Solução do Problema Imobiliário

- **Dada a amostra: $\{(\text{Size}_i, \text{Price}_i),$ para $1 \leq i \leq N_a\}$ calculam-se os valores de m e b pelo método do Mínimo dos Quadrados, e tem-se:**



$$y = 0.08x + 170$$

Solução Analítica X Solução Iterativa

- O método do Mínimo dos Quadrados é muito conveniente para obter a regressão linear $y = mx + b$, quando temos apenas uma variável independente
- Quando tem-se $y = \sum_{i=1}^{n} m_i x_i + b$, onde n é muito grande, pode ser impraticável a obtenção das fórmulas-fechadas de m_i, e o custo das derivadas da função C pode ser superior a $O(n^3)$
- Nesses casos, uma solução iterativa, como o Gradient Descent, pode ser computacionalmente muito mais simples
- Na descrição do Gradient Descent apresentada a seguir, usa-se apenas uma única variável independente de forma a facilitar o entendimento
- A generalização para n variáveis independentes será vista à frente

SOLUÇÃO ITERATIVA DA REGRESSÃO

O Método Gradient Descent (Cauchy-1847)

- **Gradient Descent** foi inventado por Augustin Cauchy em 1847 com o objetivo de definir um método para achar valores de $x_1, \ldots, x_n$ que satisfazem $u = f(x_1, \ldots, x_n) = 0$

- 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

Gradient Descent com uma Variável Independente

- **Amostra para treinamento:** $\{(x_i, y_i)$ **para** $1 \leq i \leq N_a\}$
- **Hipótese inicial para** m **e** b **aleatórios:** $h(x) = mx + b$
- **Formulação do custo** C **e de suas derivadas:**

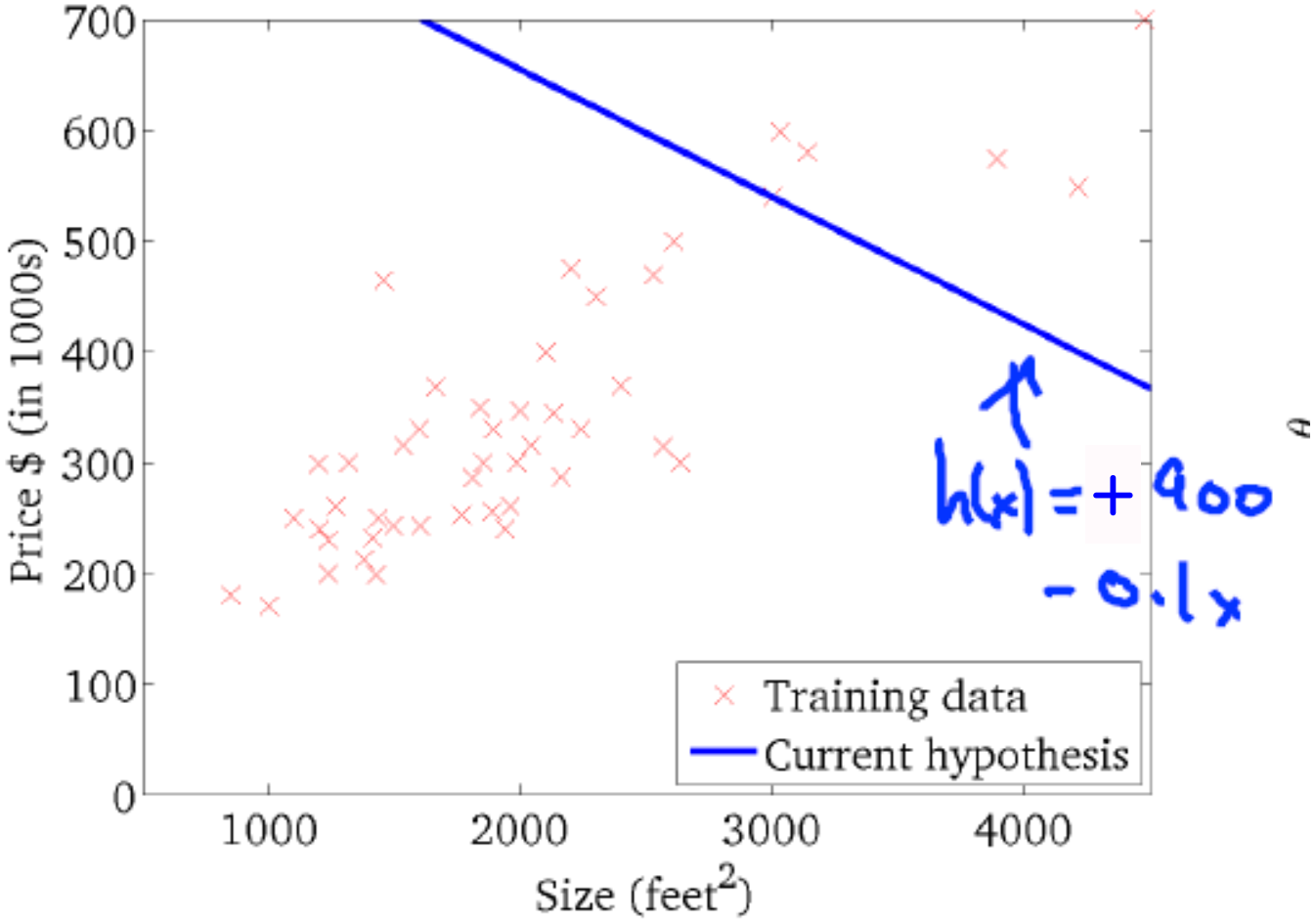
$$C(m,b) = \frac{1}{2N_a}\sum_{i=1}^{N_a}(h(x_i) - y_i)^2$$

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

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

- **Repita os passos abaixo enquanto o custo** C **estiver reduzindo-se:**
  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**

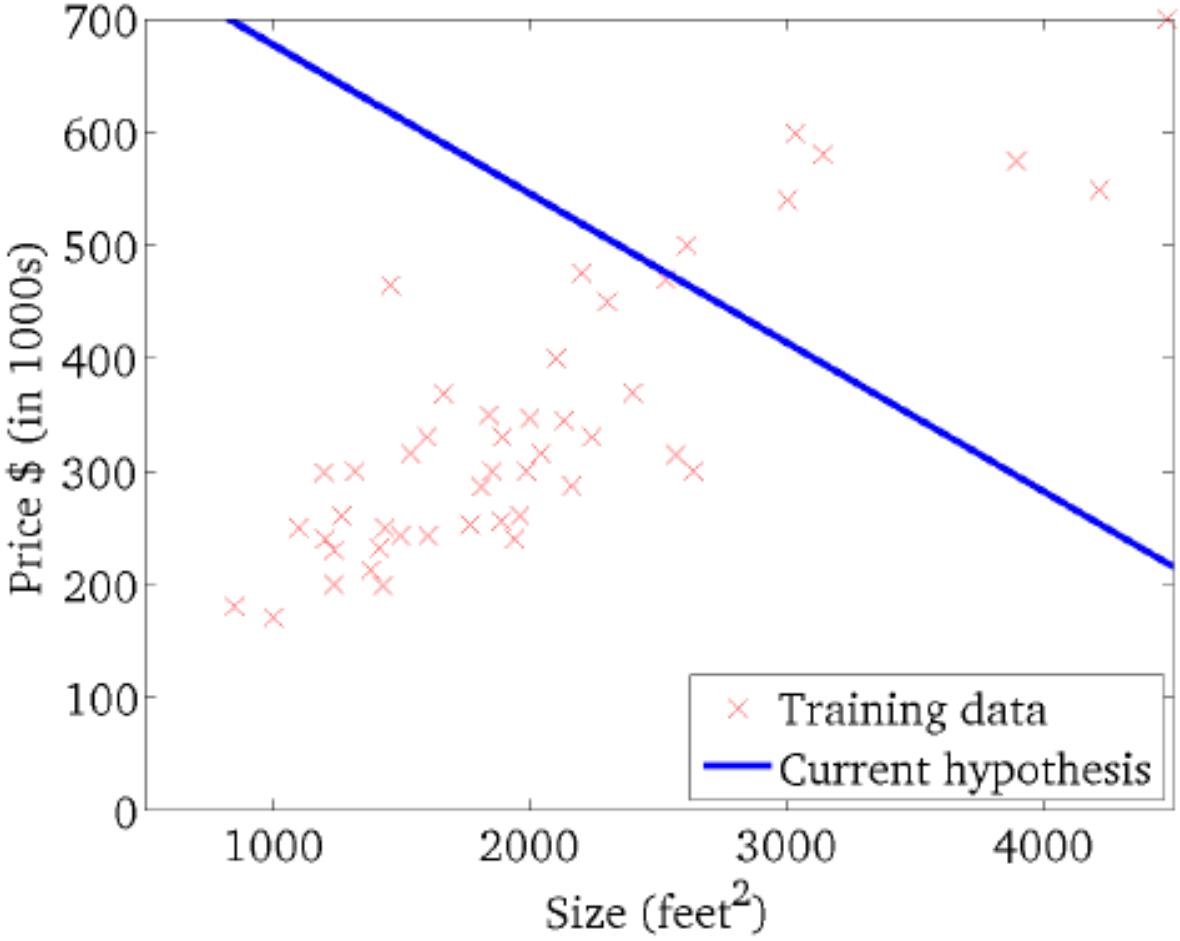
Solução por Métodos Iterativos-1



$$h(x) = m_1 x + b_1$$

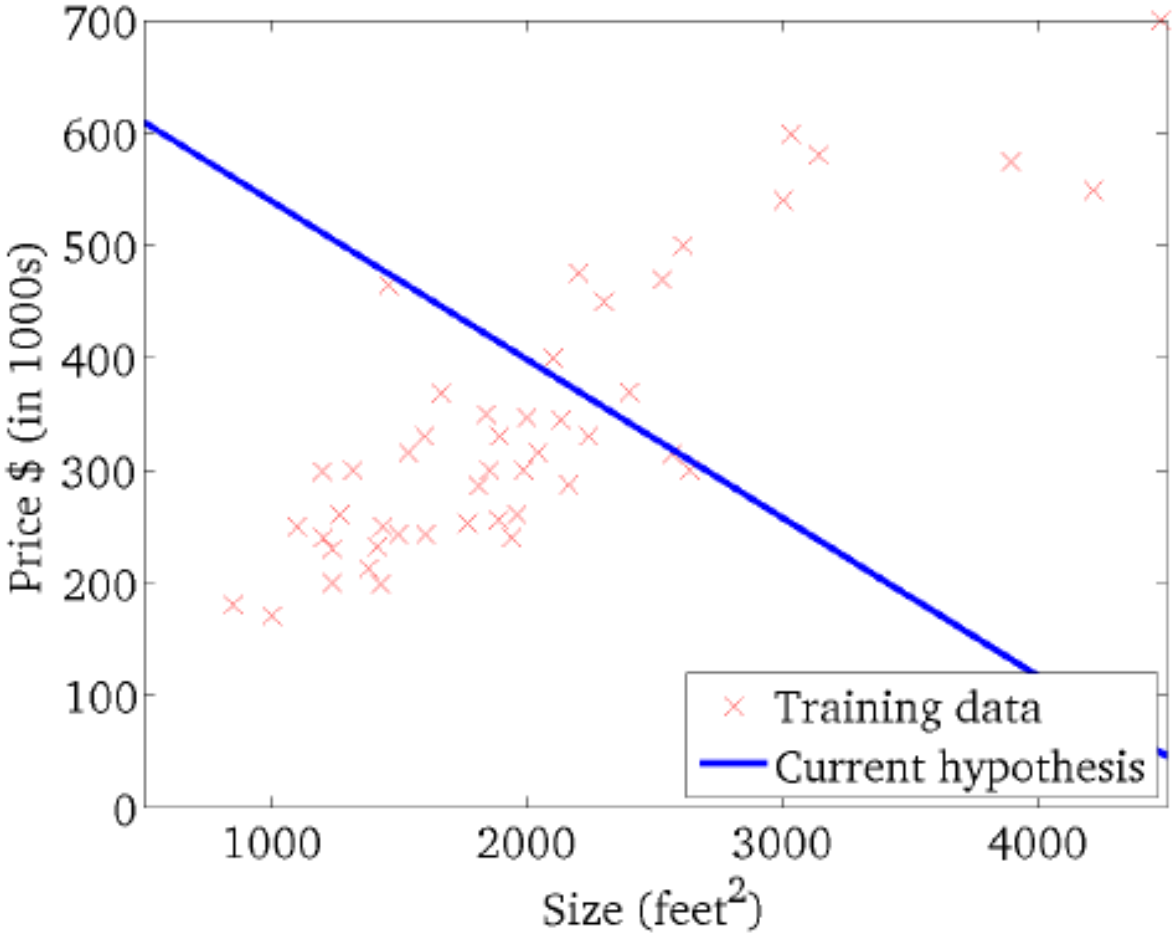
, sendo $m_1 = -0.1$ e $m_2 = 900$

Solução por Métodos Iterativos-2



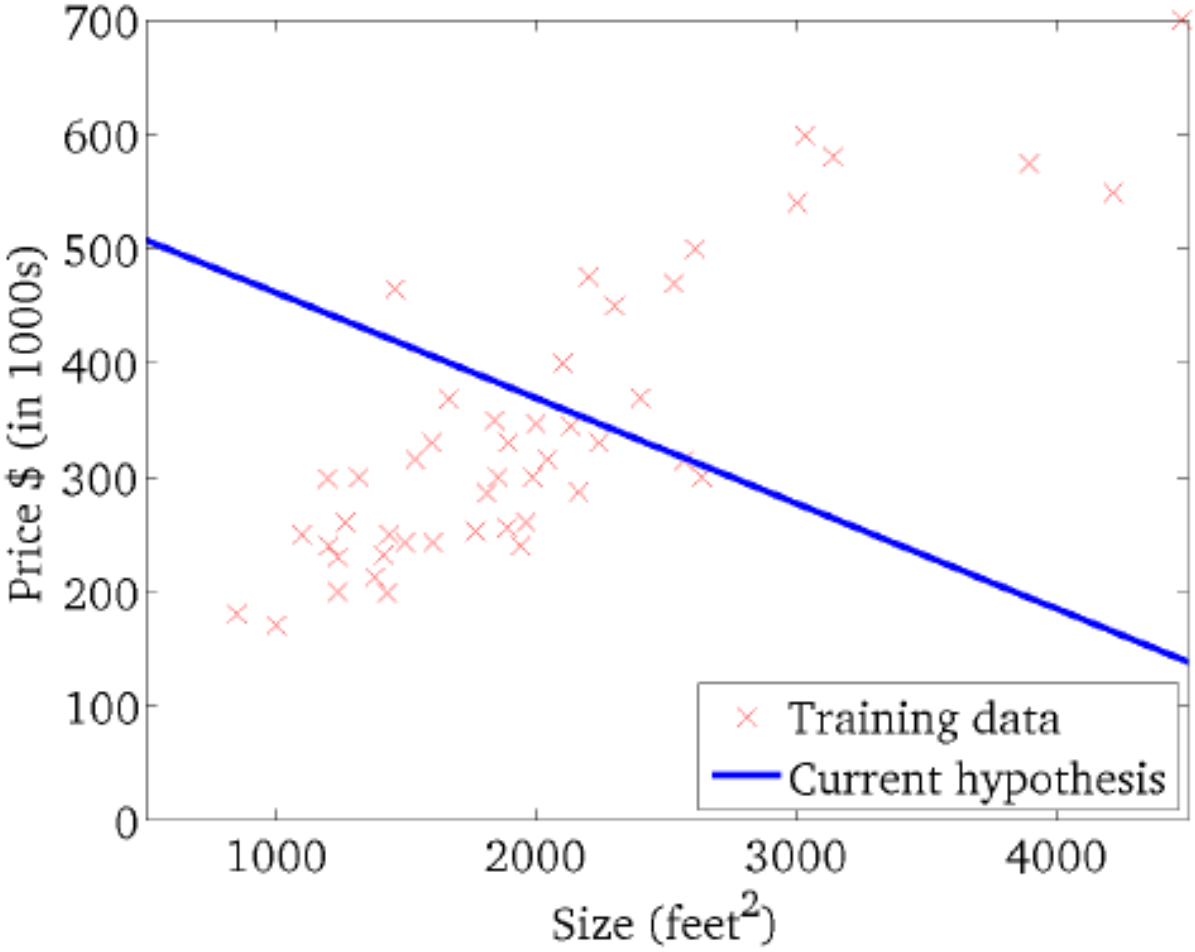
$$h(x) = m_2 x + b_2$$

Solução por Métodos Iterativos-3



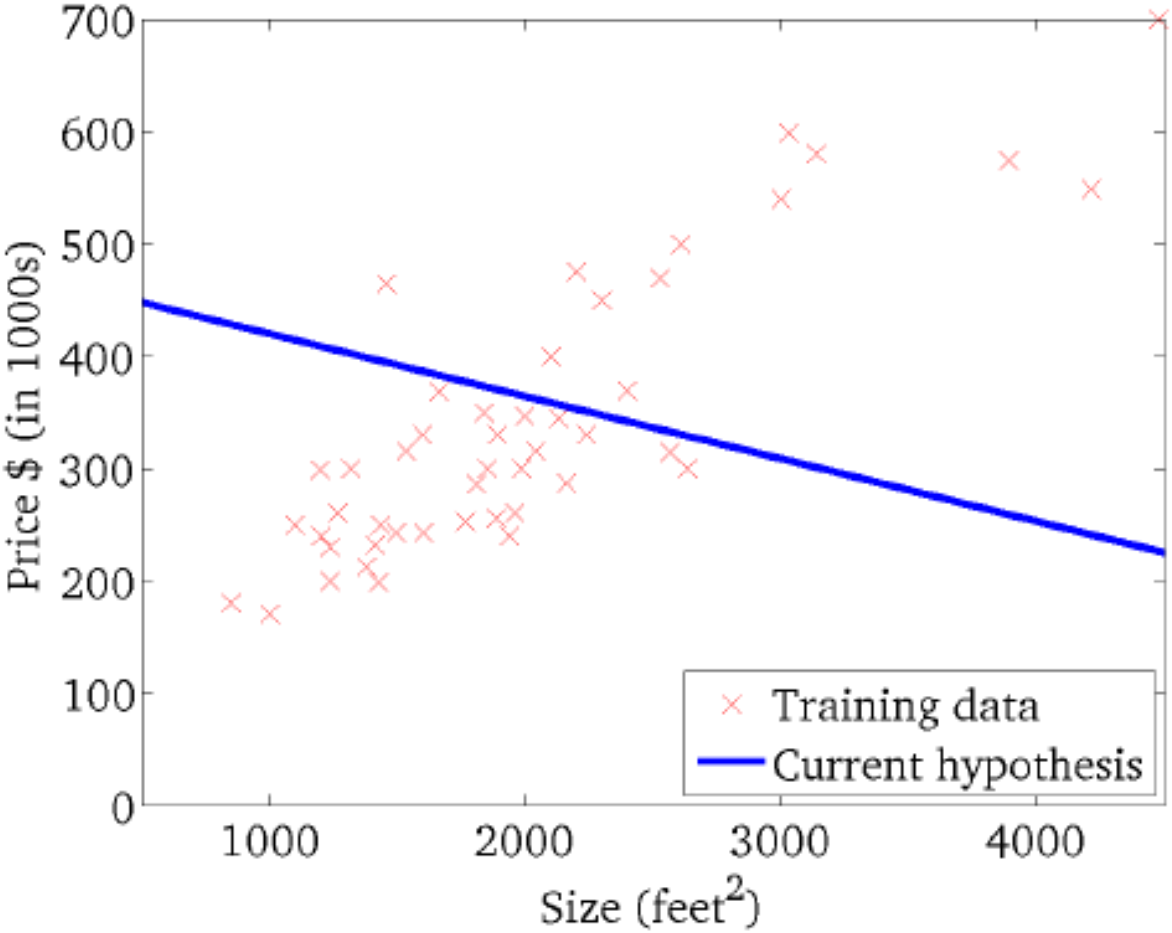
$$h(x) = m_3 x + b_3$$

Solução por Métodos Iterativos-4



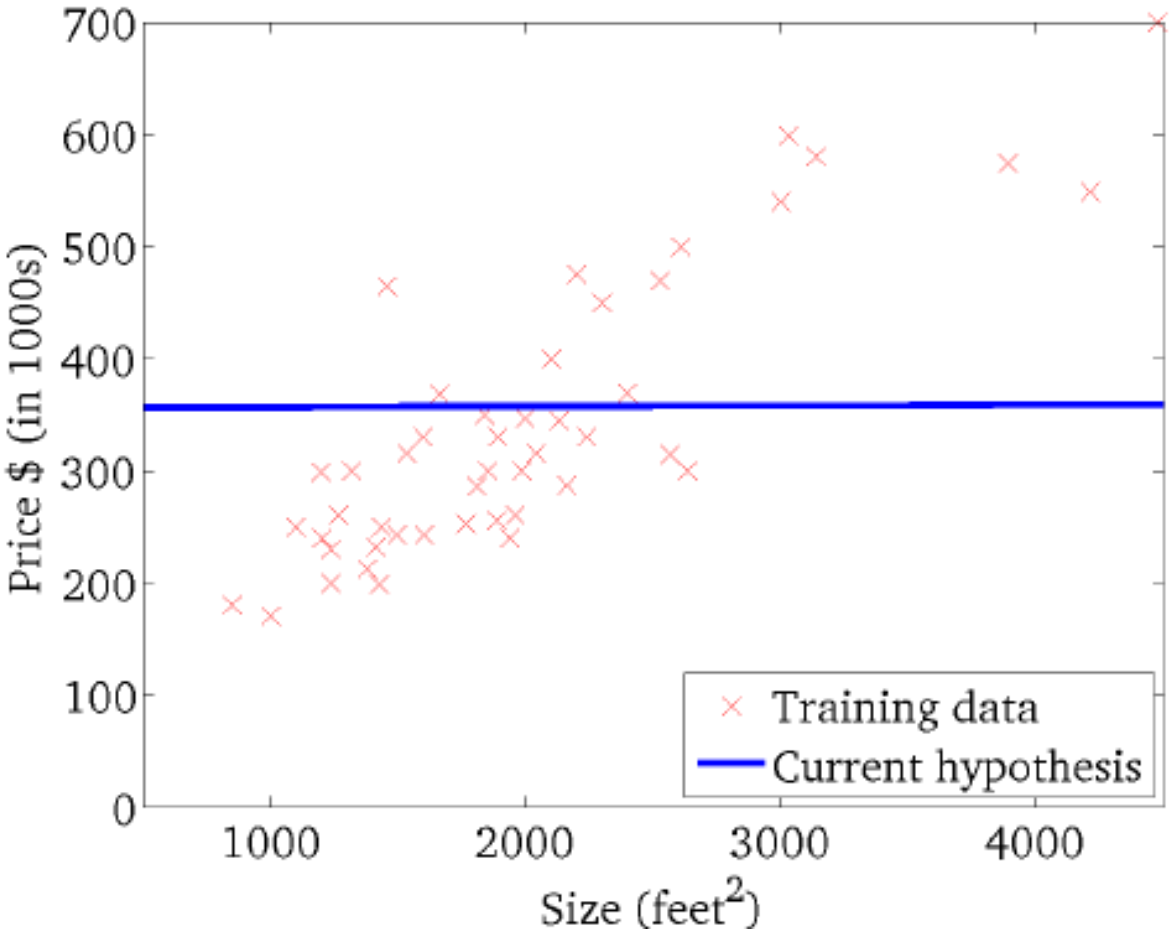
$$h(x) = m_4 x + b_4$$

Solução por Métodos Iterativos-5



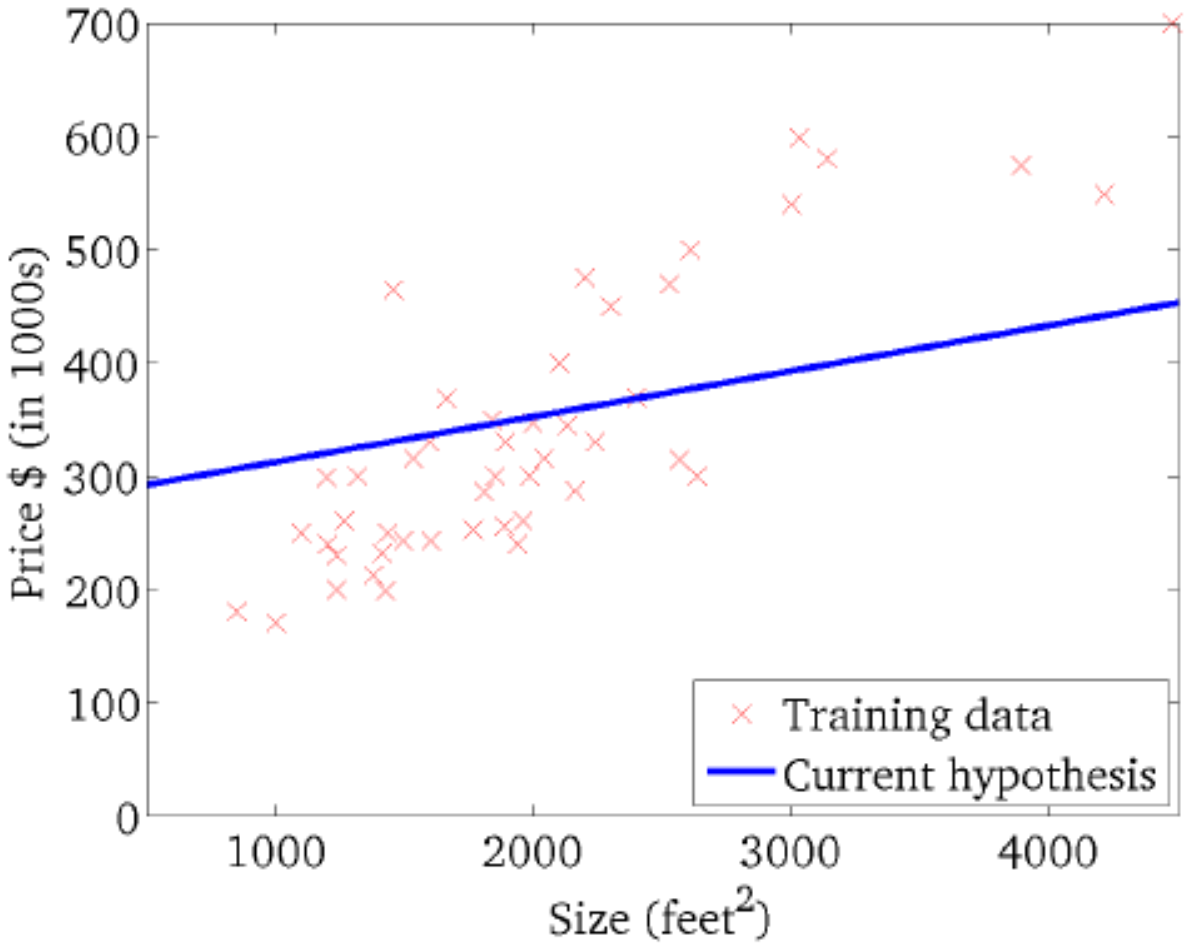
$$h(x) = m_5 x + b_5$$

Solução por Métodos Iterativos-6



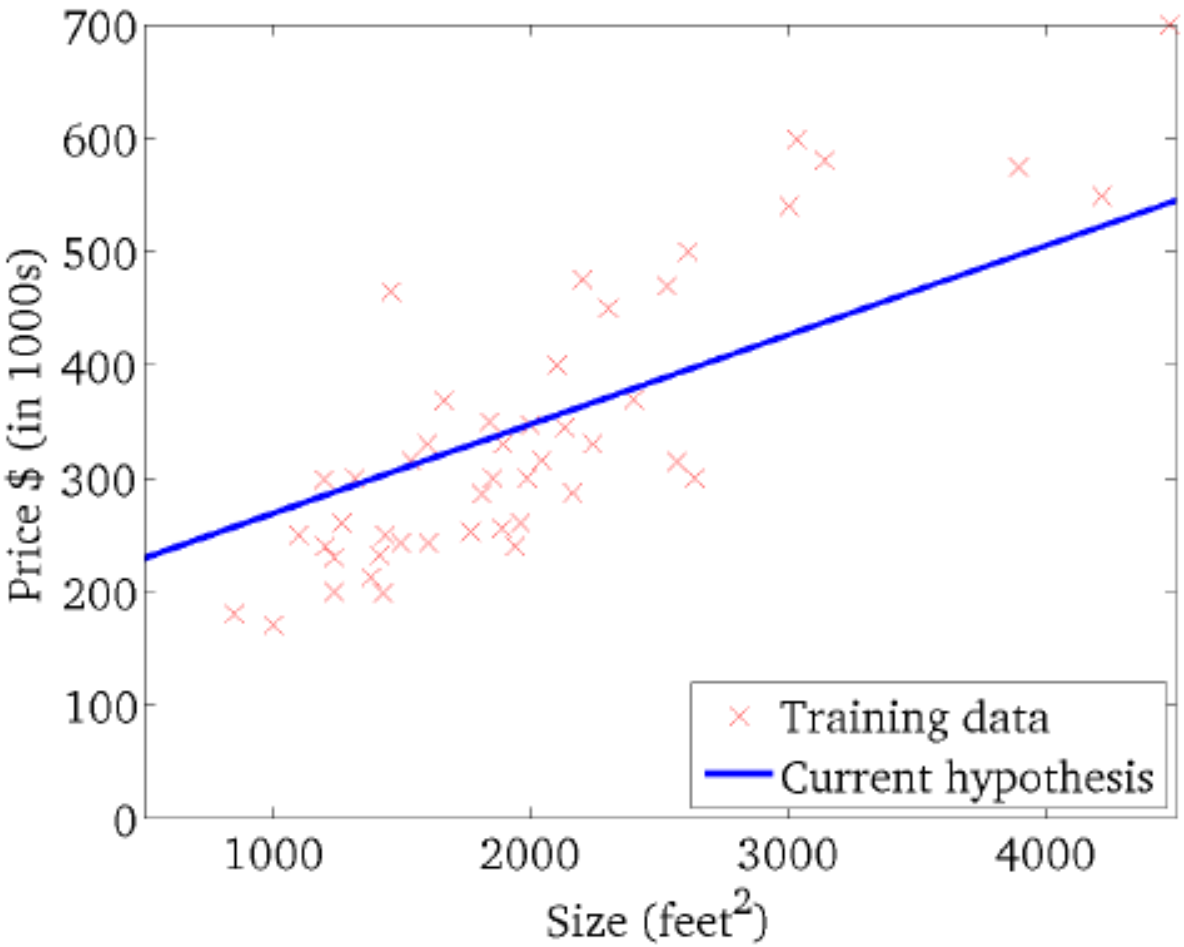
$$h(x) = m_6 x + b_6$$

Solução por Métodos Iterativos-7



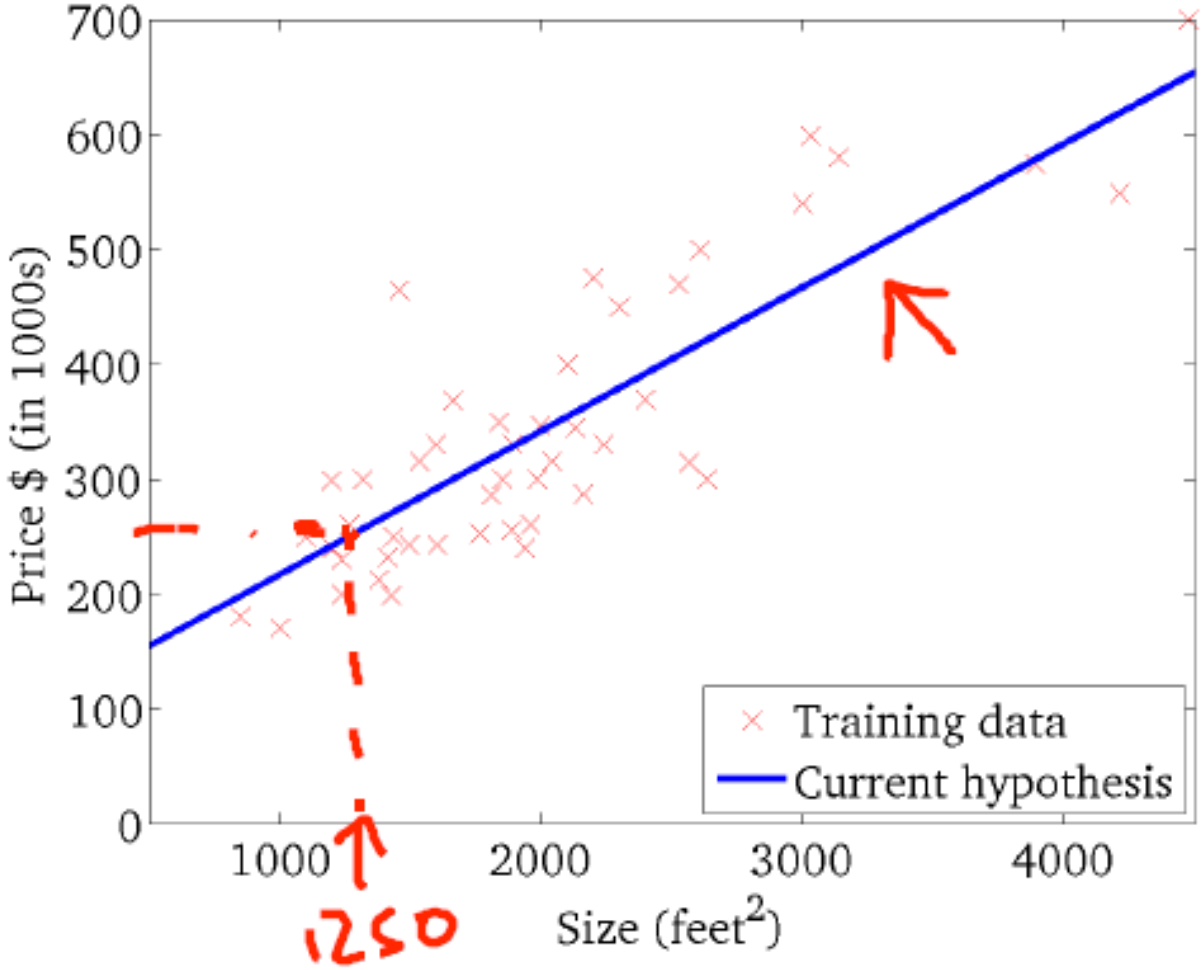
$$h(x) = m_7 x + b_7$$

Solução por Métodos Iterativos-8



$$h(x) = m_8 x + b_8$$

Solução por Métodos Iterativos-fim



$h(x) = mx + b$, **com** $m = 0.08$ **e** $b = 170$

Abordagens da Descida pelo Gradiente

- **Há duas variantes importantes do método iterativo para determinar os parâmetros m e b que minimizam a função de custo C:**
  1. **Modo Batch:**
     usa-se toda a amostra em cada ajuste de m e b
  2. **Modo Estocástico:**
     particiona-se a amostra em subamostras, denominadas mini-batch, e fazem-se ajustes de m e b para cada subamostra em separado

Formulação do Modo Batch

- **Amostra batch para treinamento:**
  $A = \{(x_i, y_i),$ **para** $1 \leq i \leq N_a\}$
- **Função de custo para toda a amostra** A**:**
  $C(m, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} (h(x_i) - y_i)^2,$
  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 $\frac{\partial C}{\partial m} = \frac{1}{N_a} \sum_{i=1}^{N_a} (mx_i + b - y_i)x_i$
  $\frac{\partial C}{\partial b} = \frac{1}{N_a} \sum_{i=1}^{N_a} (mx_i + b - y_i)$

Algoritmo de Descida no Modo Batch

- 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:
  1. 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
  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 valor anterior
- Método muito caro, pois o custo total é reavaliado a cada ajuste de m e b sempre considerando toda a amostra A

Formulação no Modo Estocástico

- **Amostra de treinamento:** $A = \{(x_i, y_i),$ **para** $1 \leq i \leq N_a\}$
- **Função de custo para toda a amostra** A**:**
  $C(m,b) = \frac{1}{2N_a}\sum_{i=1}^{N_a}(h(x_i) - y_i)^2$, **onde** $h(x_i) = mx_i + b$
- **Subamostras mini-batch** A_K **de tamanho** s**, e.g.,** $50 \leq s \leq 256$**:**
  $A_K = \{(x_i, y_i),$ **para** $K \leq i \leq (K+s)\}$,
  onde $K \in \{1, s+1, 2s+1, \cdots, (N_a/s - 1)s + 1\}$
- **Função de custo para** A_K**:** $C_K(m,b) = \frac{1}{2s}\sum_{i=K}^{K+s}(h(x_i) - y_i)^2$
- **Gradiente da função de custo de cada subamostra** K**:**

$$\nabla C_K(m,b) = \left(\frac{\partial C_K}{\partial m}, \frac{\partial C_K}{\partial b}\right)$$

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

$$\frac{\partial C_K}{\partial b} = \frac{1}{s}\sum_{i=K}^{K+s}(mx_i + b - y_i)$$

Algoritmo de Descida 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, até C convergir:
  1. 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 $b := b - \alpha\frac{\partial C_K}{\partial b}$
  2. recalcule $C(m,b)$ e compare com valor anterior
- 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

Gradient Descent com n Variáveis Independentes

- **Amostra para treinamento:** $\{(x^{(i)}, y^{(i)})$ **para** $1 \leq i \leq N_a\}$
- **Sejam** $x = (x_1, \cdots, x_n)$ **e** $m = (m_1, \cdots, m_n)$**, para** $n \geq 1$
- **Hipótese para** m **e** b **iniciados aleatoriamente:** $h(x) = \sum_{j=1}^{n} m_j x_j + b$
- **Indicador** C **de erro da hipótese** h **em relação à amostra:**

$$C = \frac{1}{2N_a} \sum_{i=1}^{N_a} \left[h(x^{(i)}) - y^{(i)} \right]^2$$

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

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

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

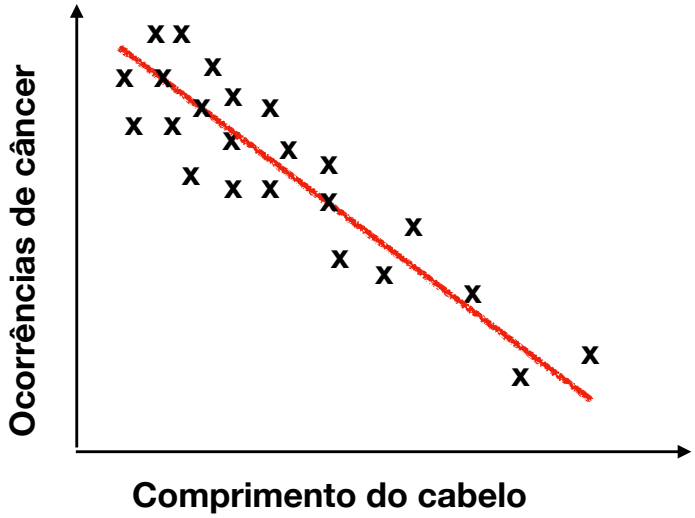
$m_k := m_k - \alpha \frac{\partial C}{\partial m_k}$ **e** $b := b - \alpha \frac{\partial J}{\partial b}$

onde α **é a taxa de aprendizagem**

- **Repetem-se as afinações de** m **e** b **até** C **parar de diminuir**

Alerta: Regressão X Causa e Efeito

- Regressão não estabelece nem identifica relação de causa e efeito
- Se essa relação existir, regressão apenas permite determinar o efeito dado a causa
- Regressão pode levar a conclusão errônea caso a relação não exista.



- Comprimento do cabelo X ocorrência de câncer de próstata?

Regressão Logística

- Regressão logística ocorre 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 $y = h(x)$:
  1. faz-se uma regressão linear clássica $y' = h'(x)$
  2. 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
  3. discretiza-se y'' para obter o resultado final

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

APRENDIZAGEM DE MÁQUINA

Filosofia do Aprendizado

- Machine learning é o processo de descobrir uma função-objetivo f que mapeia as features x (entradas) nos respectivos labels y (resultados), i.e., $y = f(x)$
- O processo de aprendizagem 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 aprendizagem é dito parametrizado, e sua função é inferir os parâmetros ou coeficientes que compõem a função f
- Quando não se faz suposição quanto à forma da função f, o algoritmo de aprendizagem é dito não-parametrizado
- Nesse caso, pode ser necessário testar diferentes alternativas para se escolher a melhor forma da função e os valores de seus parâmetros

Algoritmos de Aprendizado Parametrizados...

- Como foi dito, a forma da função $y = f(x)$ é arbitrada
- 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

...Algoritmos de Aprendizado Parametrizados

- 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

- As desvantagens dos algoritmos paramétricos são:
  - riscos: a prefixação da forma da função limita a qualidade do aprendizado, sendo possível underfitting
  - escalabilidade: algoritmos limitados a problemas mais simples

Algoritmos de 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
- Sem prefixação da forma da função, o algoritmo é livre para aprender qual é a melhor forma funcional a partir da amostra de treinamento
- Métodos não-paramétricos devem ser usados quando se têm grandes volumes de dados amostrais e pouco conhecimento de seu padrão
- Essa abordagem elimina os riscos de prefixação da forma da função
- Literatura cita modelos que são baseados em bilhões de parâmetros, o que impossibilita o uso de algoritmos paramétricos

...Algoritmos de Aprendizado Não-Parametrizados

- As vantagens dos algoritmos não-paramétricos são:
  - flexibilidade: capaz de fazer ajustes de muitas formas funcionais
  - aplicabilidade: não têm limitação quanto à forma da função a ser aprendida
  - performance: apresentam grande capacidade de generalização

- As desvantagens dos algoritmos não-paramétricos são:
  - demanda de dados: necessário grandes amostras de treinamento para induzir a função desejada
  - custo: treinamento mais trabalhoso, pois pode requerer determinação de muitos parâmetros de treinamento
  - overfitting: observa-se maior risco de overfitting

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)
- O ponto mais importante do aprendizado é que a função f seja capaz de mapear corretamente entradas que não estejam na amostra
- Ou seja o aprendizado visa generalizar o mapeamento descrito pela amostra para quaisquer dados dentro do domínio do problema
- Generalização é indispensável, pois tipicamente as amostras de treinamento são incompletas e imprecisas
- Generalização está fortemente vinculada à distribuição dos dados da amostra de treinamento
- Se essa distribuição dos dados mudar, deve proceder com um retreinamento para ajustar f à nova distribuição

...Generalização da Amostra

- Se as novas entradas apresentadas à função f inferida pelo algoritmo de aprendizagem não seguirem a distribuição dos dados da amostra de treinamento, o modelo pode não funcionar adequadamente

- Isso implica na necessidade de retreinamento com a ampliação da distribuição de dados para incorporar a das novas entradas

- O mecanismo de inferência usado pelo mecanismo de generalização é o de indução, no sentido de 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

Resumo dos Passos de um Aprendizado

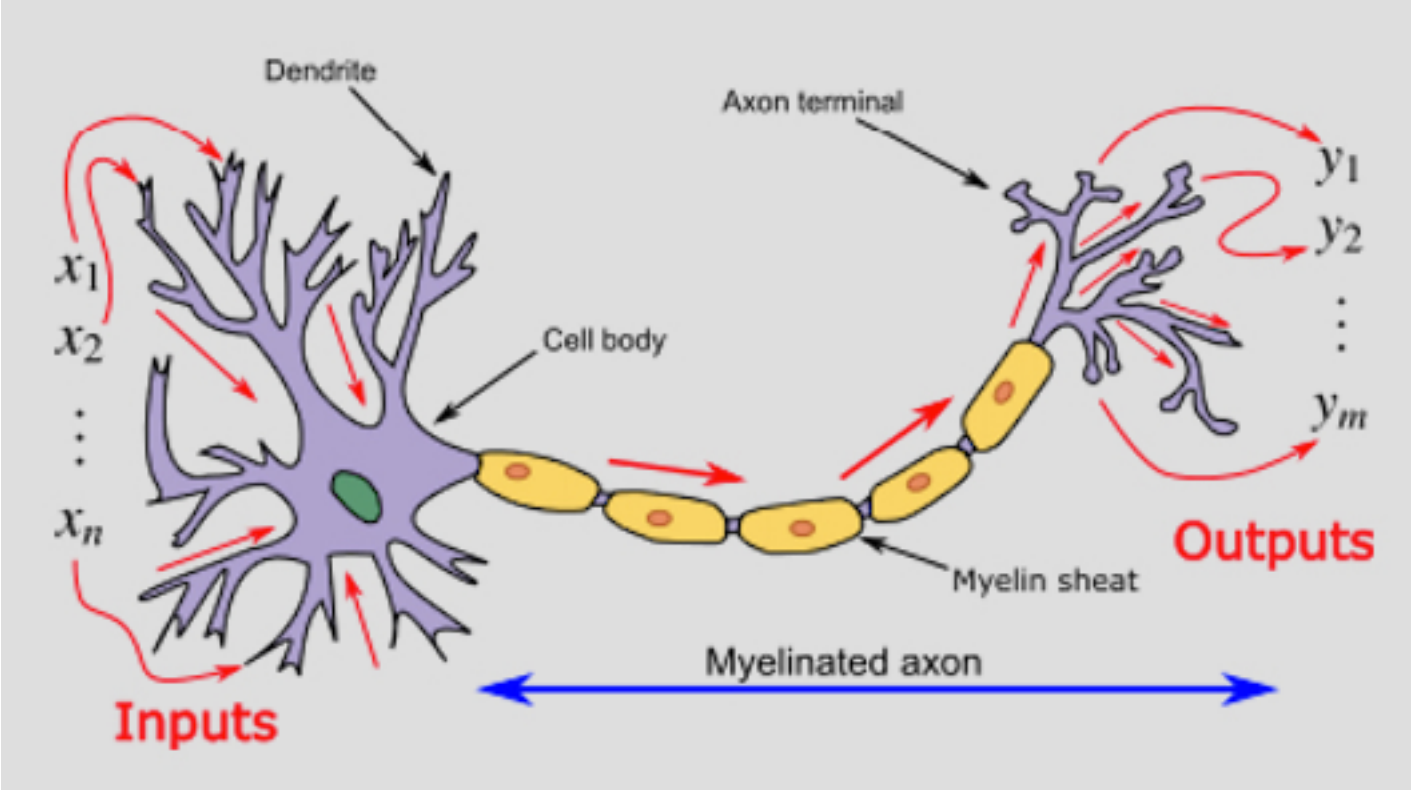
1. Identifique uma amostra de treinamento A:
   $\{(x^{(i)}, y^{(i)}), i : 1..N_a\}$, onde cada feature $x = (x_1, \cdots, x_m)$
2. Arbitre uma função $y = g(w, x)$, que mapeie as features x a labels y, com custo médio dado por
   $C(w) = \frac{1}{2N_a} \sum_{i=1}^{N_a} \left[g(w, x^{(i)}) - y^{(i)} \right]^2$
   sendo w um vetor de constantes cujos valores devem ser aprendidos pelo algoritmo de treinamento
3. Aplique o algoritmo do Gradient Descent para identificar (ou aprender) o valor do vetor $\overline{w}$ das constantes em w que minimiza o custo médio $C(w)$
4. A generalização do mapeamento descrito por A é então dado por $y = h(x)$, onde a função $h(x) = g(\overline{w}, x)$

Redes Neurais X Aprendizado de Máquina

1. A definição da forma mais adequada da função $g(w, x)$ citada acima é essencial para se ter um bom aprendizado
2. A forma escolhida para essa função tem efeito direto na qualidade da generalização da amostra de treinamento
3. Em princípio, a definição da forma de $g(w, x)$ é arbitrária e empírica, sendo feita por tentativa e erro
4. Por outro lado, um método sistemático de definição da forma geral de g, ainda que empírico, são as redes neurais.
5. Assim, em vez de arbitrar a expressão de uma função de generalização com coeficentes a ser aprendidos, arbitra-se a estrutura de uma rede neural com pesos e biases a ser determinados

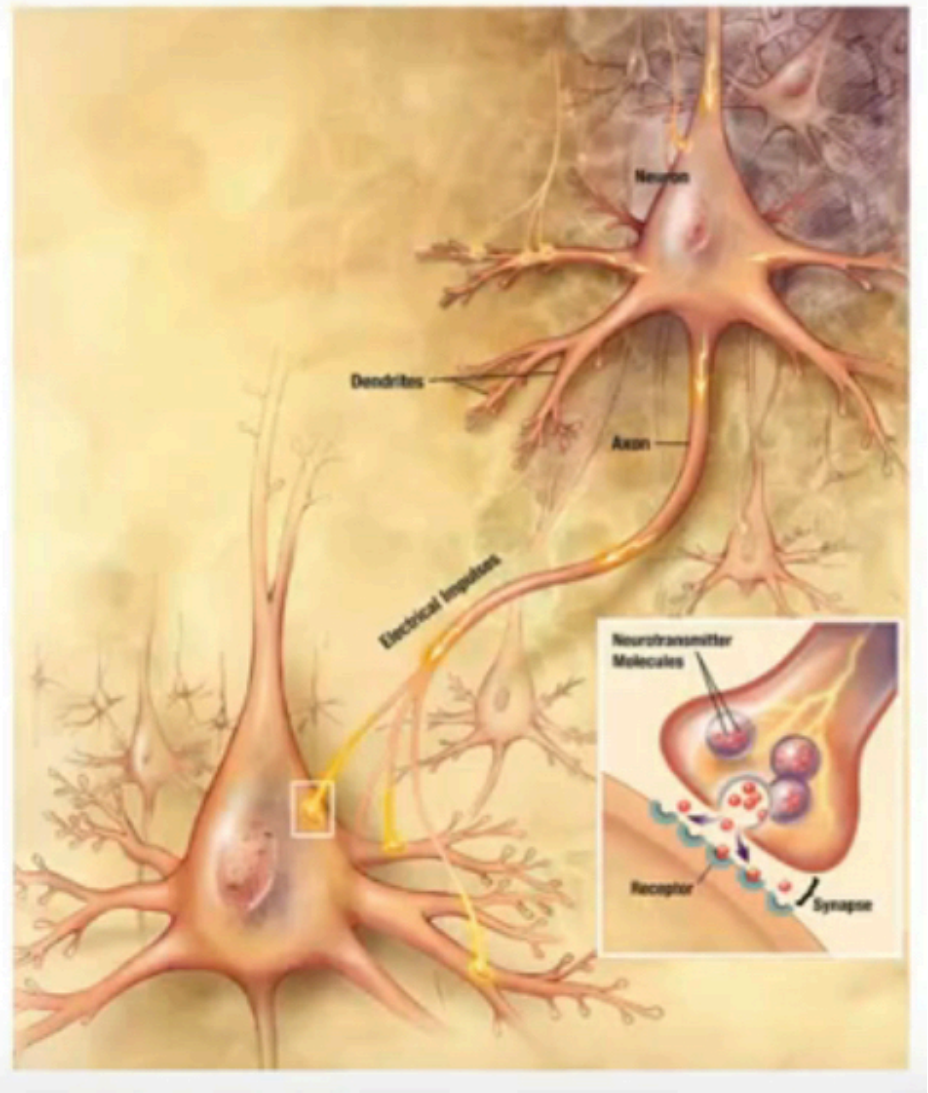
REDES NEURAIS DENSAS

Neurônio Humano

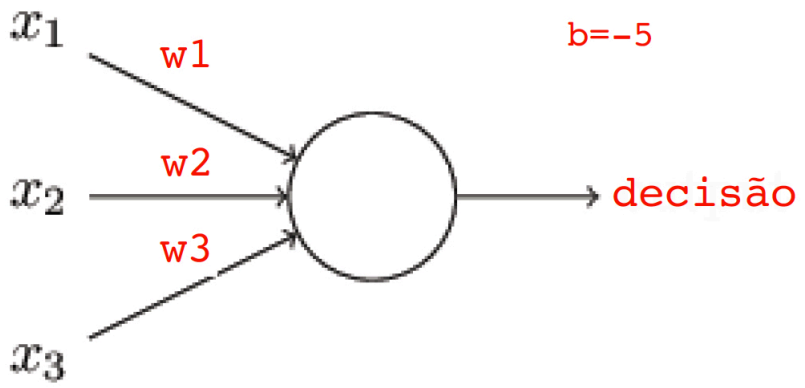


- x_i, $i : 1..n$, são terminais que recebem impulsos elétricos
- y_i, $i : 1..m$ são terminais que transmitem impulsos elétricos

Rede de Neurônios Humanos



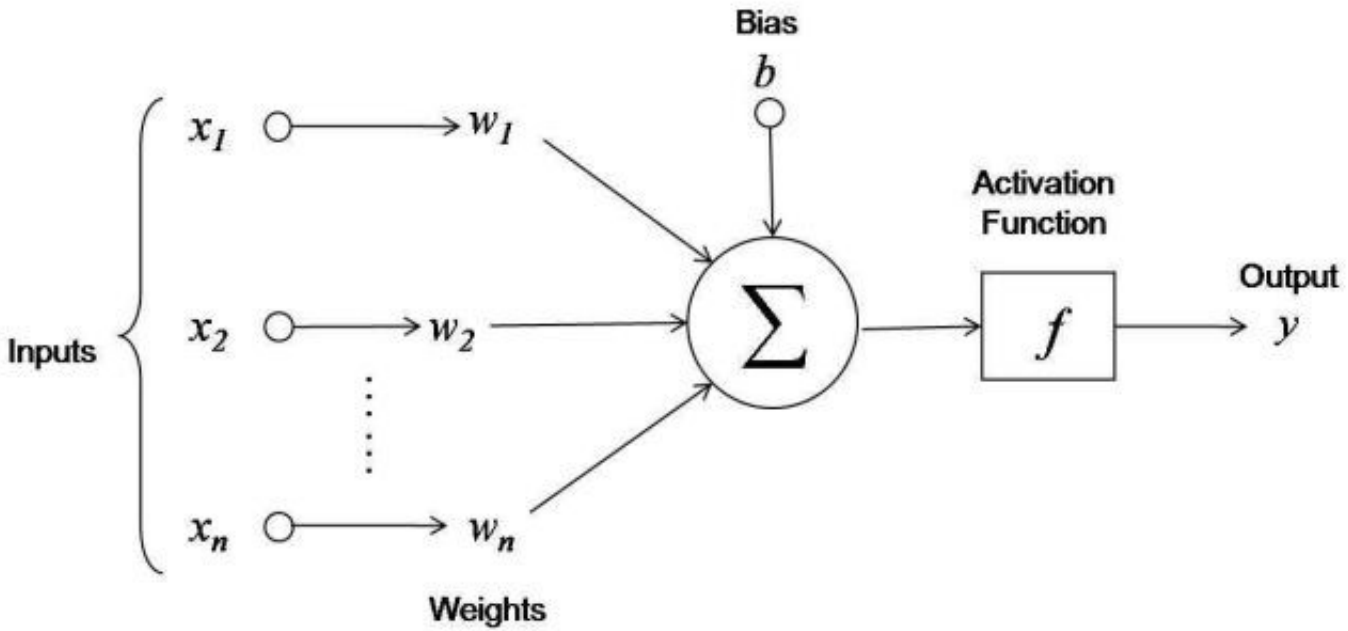
Modelo Biológico do Neurônio



- **Problema: decisão de ir ao Mineirão ver um jogo:**
  - x_1, x_2 e x_3 são evidências ou condições que levam à decisão
  - x_1 indica se ou não é um jogo de meu time (peso $w_1 = 4$)
  - x_2 indica se minha mulher vai comigo ou não (peso $w_2 = 1$)
  - x_3 indica se tenho ou não carona assegurada (peso $w_3 = 2$)
  - b é o chamado $bias$ ($b = -5$, onde **5** é o **threshold**)
- **decisão** $= f(\overbrace{x_1w_1 + x_2w_2 + x_3w_3 + b}^{a}) = \begin{cases} 0 & \text{se } a \leq 0 \Longrightarrow \neg \text{ synapse} \\ 1 & \text{se } a > 0 \Longrightarrow \quad \text{synapse} \end{cases}$
- **Novos pesos** $(4, 3, 1) \Longrightarrow$ **a mulher passar a mandar**

Neurônios Perceptrons

Perceptron: um Neurônio de n Entradas



- **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

  $x_i = 1$: **com estímulo na entrada** x_i

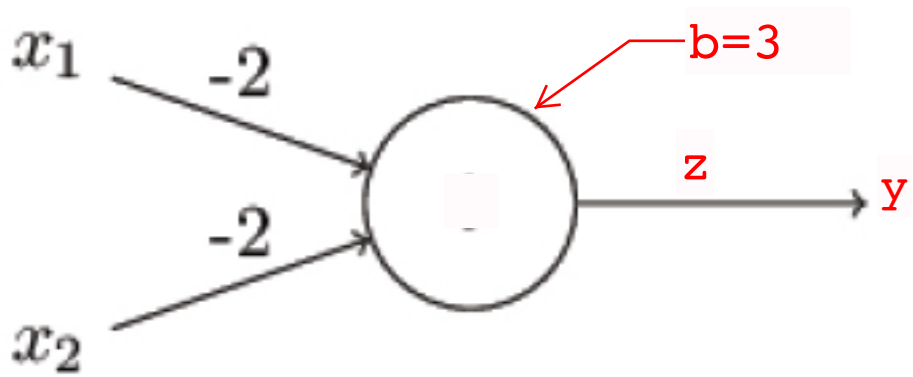
  $$y = f(\Sigma) = \begin{cases} 0 \text{ se } \Sigma \leq 0 \Longrightarrow \text{não gera synapse} \\ 1 \text{ se } \Sigma > 0 \Longrightarrow \text{gera synapse} \end{cases}$$

Modus Operandi de um Perceptron

- 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
- Saída y com valor 1 indica uma decisão que deve ser propagada para outros perceptrons (synapse)
- O peso w_i de cada evidência define seu poder de influência no processo de ponderação para a decisão final
- O threshold define o valor mínimo que influências acumuladas das evidências devem atingir para permitir uma decisão positiva
- O bias é uma medida da propensão de o perceptron produzir a saída 1 ($b = -$threshold)
- Modelo de decisão de um perceptron depende de seus pesos e biases

Perceptron que Computa NAND

- **Considere o perceptron, onde $z = x_1 w_1 + x_2 w_2 + 3$ e $y = f(z)$**



Q = A NAND B

Truth Table

Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

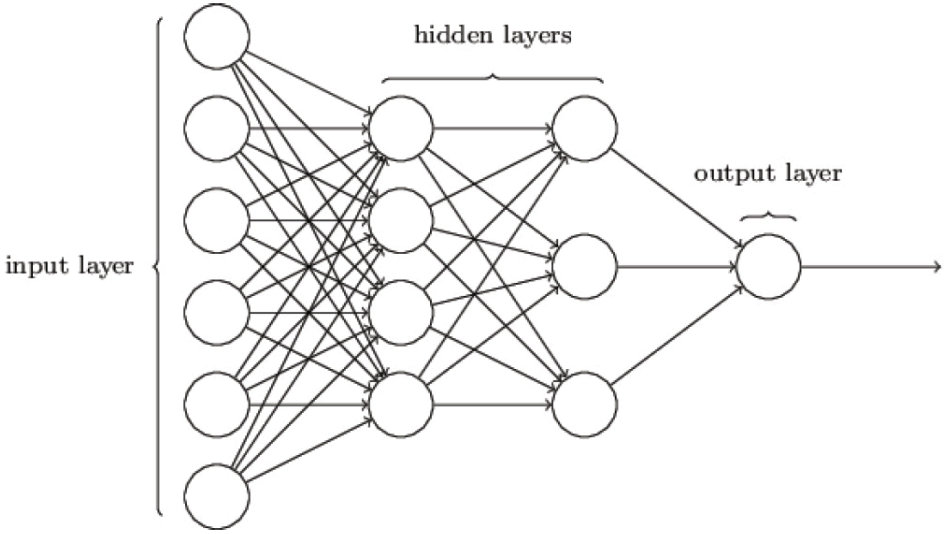
- **Saídas:**

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

- **Como NANDs, rede de perceptrons pode computar qualquer função lógica! $\Longrightarrow$ perceptron é um elemento universal para computação**

Redes de Perceptrons

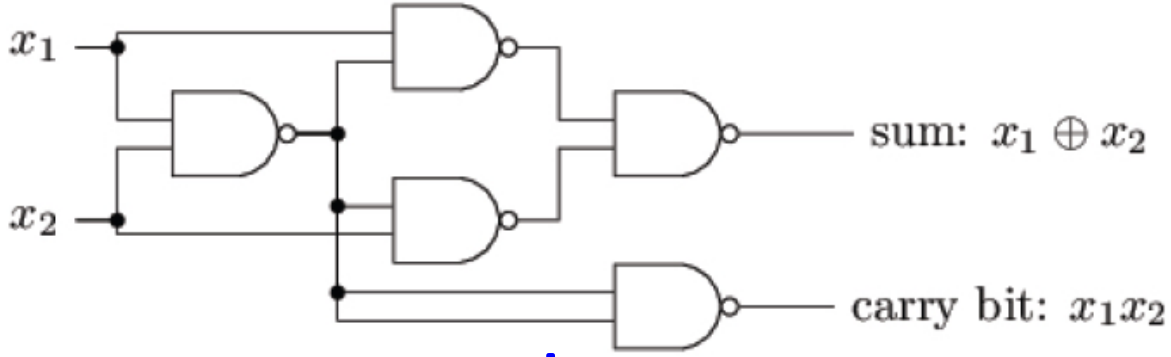
- Perceptrons podem ser organizados em redes de várias camadas



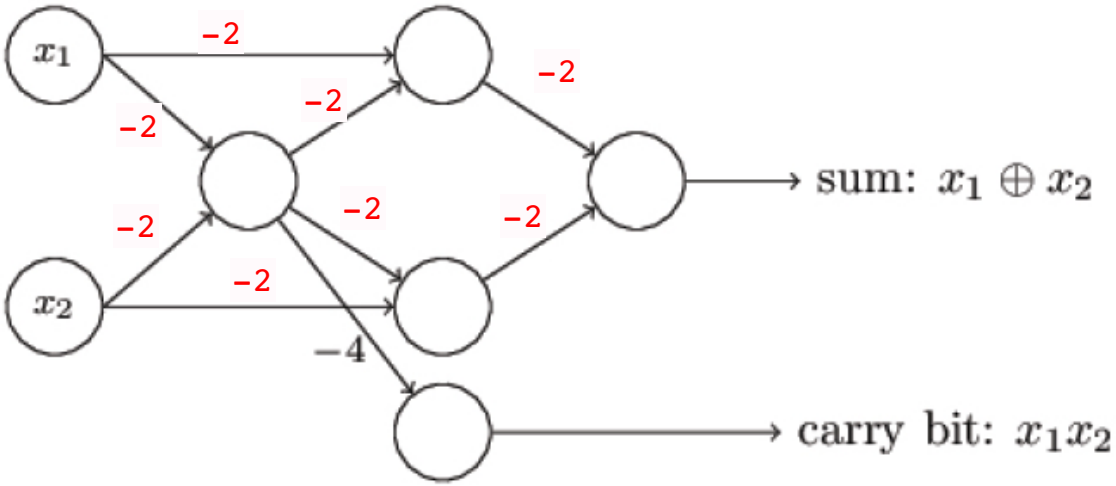
- Camada de entrada: perceptrons ponderam as evidências de entrada
- Camadas ocultas: perceptrons ponderam os resultados da camada anterior em um nível mais complexo e mais abstrato
- Camada de saída: perceptron faz a ponderação para a decisão final
- Quando todos os neurônios de uma camada estão conectados a todos da camada seguinte diz-se que a rede é densa

Rede Perceptron de um Somador

- **Circuito com portas NAND que adiciona os bits x_1 e x_2**



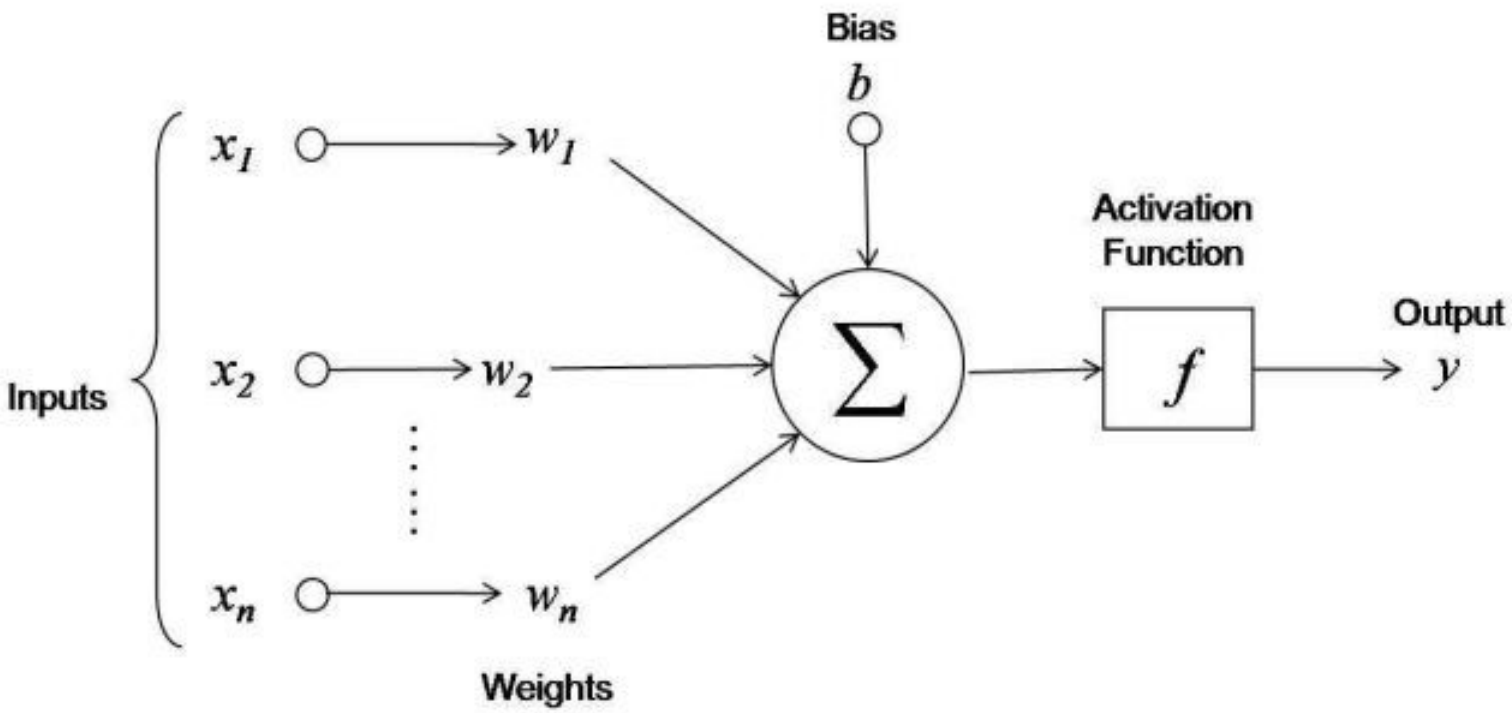
- **Rede de perceptrons correspondente:**



- **Com bias = 3**

Neurônios Sigmoids

Neurônio Sigmoid de n Entradas



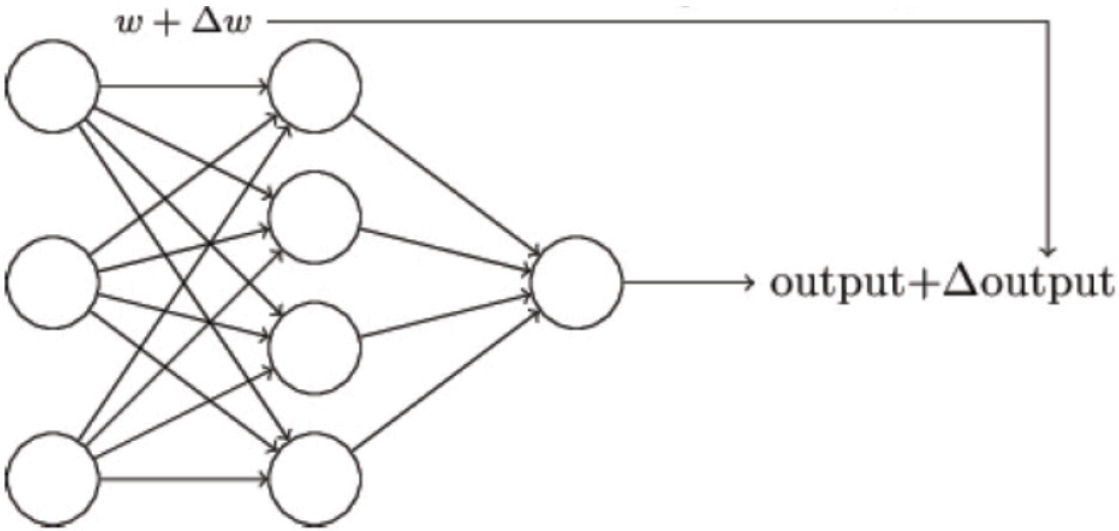
onde:

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

Afinação de Pesos e Biases

- Aprendizagem de máquina consiste 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:

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

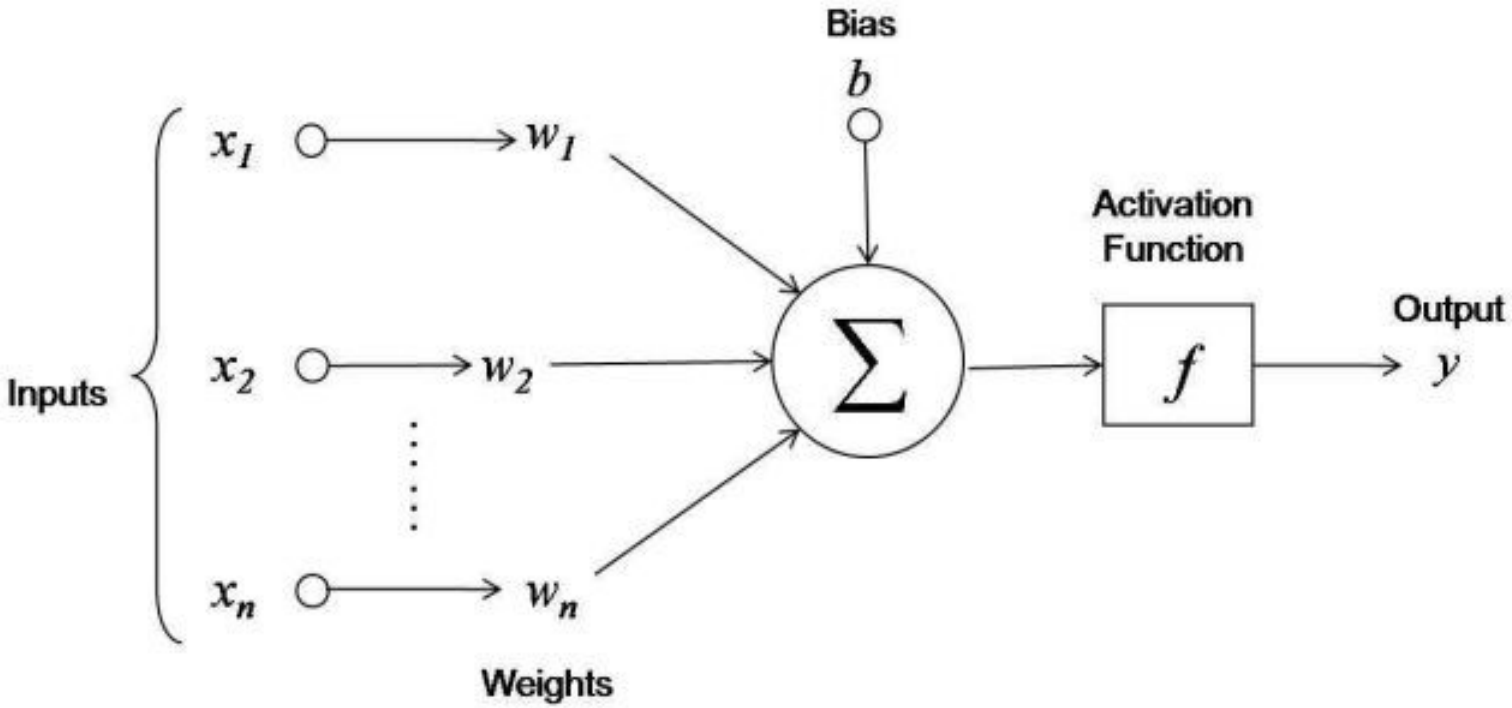


- Todos os pesos e biases devem ser afinados iterativamente

Perceptron X Neurônio Sigmoid

- 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 reais no intervalo $[0.0, 1.0]$
- Assim, aprendizagem de máquina funciona melhor com a função σ (sigmoid), pois, pequenas mudanças nos pesos e biases geram pequenas alterações no resultado final $y \in [0.0, 1.0]$
- No fim, para gerar ou não a synapse, pode-se interpretar $y \leq 0.5$ como false e $y > 0.5$ como true
- O mesmo se aplica a outras funções de ativação, como ReLU ou $\tanh$

Treinamento de um Neurônio Sigmoid



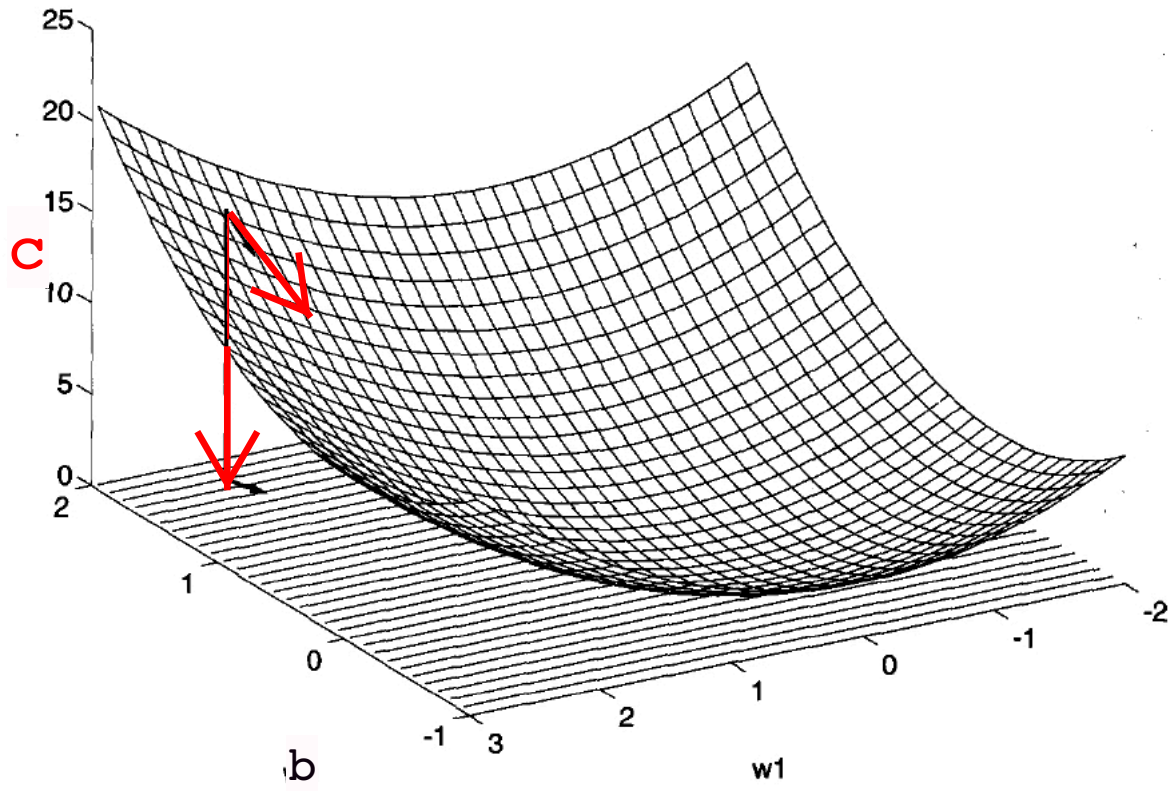
- **Erro C do treinamento com amostra $(x^{(i)}, y^{(i)})$, $i : 1..N_a$, sendo $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ e f a função sigmoid σ:**

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

- **Determinam-se w_k, $k : 1..n$, e b pelo Gradient Descent**

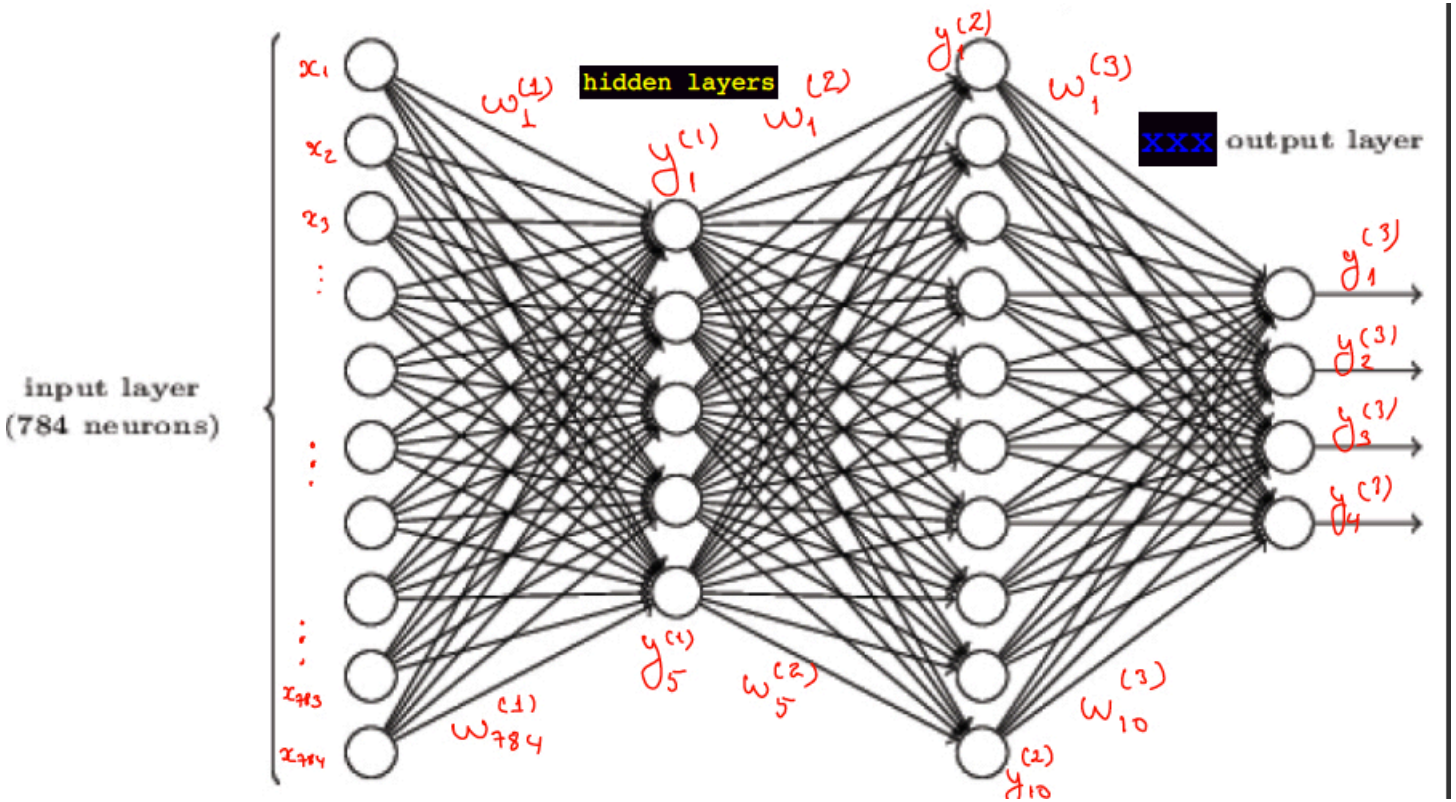
Descida pelo Gradiente (visão simplificada)

- Seta indica direção mais íngreme para o ponto de erro mínimo
- Reta vertical dá a medida do erro no ponto considerado
- $C(w_1, b) = \frac{1}{2N_a} \sum_{i=1}^{N_a} [(w_1 x_i + b) - y_i]^2$
- Na Figura, supõe-se:
  Amostra: (x_i, y_i)
  $y = w_1 x + b$
- Deseja-se achar b e w_1 que minimizem a função C



FUNÇÃO MODELADA PELA REDE

Exemplo de uma Rede Neural mais Complexa



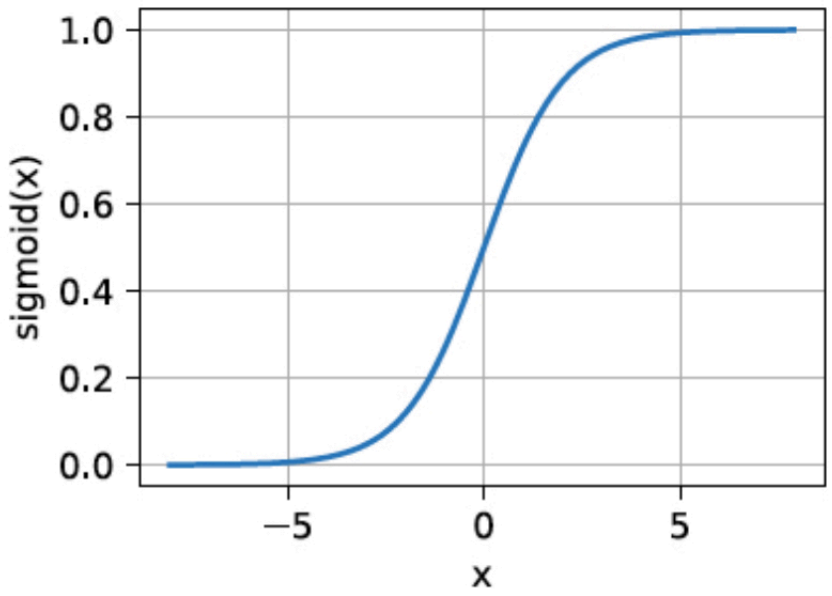
- f é a função de ativação
- $y_k^{(1)} = f(\sum_{i=1}^{784} w_i^{(1)} x_i + b^{(1)})$ para $k : 1..5$
- $y_k^{(2)} = f(\sum_{i=1}^{5} w_i^{(2)} y_i^{(1)} + b^{(2)})$ para $k : 1..10$
- $y_k^{(3)} = f(\sum_{i=1}^{10} w_i^{(3)} y_i^{(2)} + b^{(3)})$ para $k : 1..4$

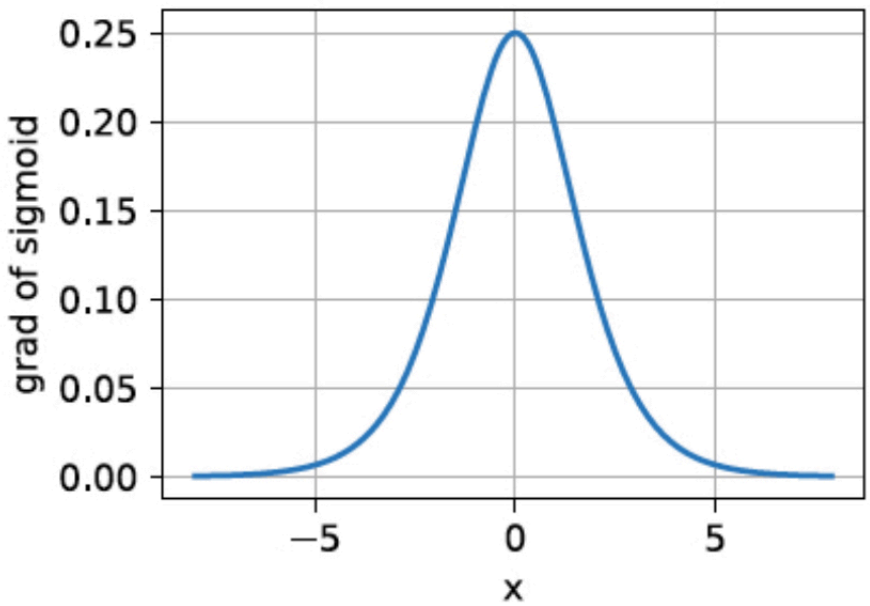
Funções de Ativação Populares

- **Função Sigmoid (σ):**
  - confina valores no intervalo $[0.0, 1.0]$
- **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 $[-1.0, 1.0]$

Função de Ativação Sigmoid

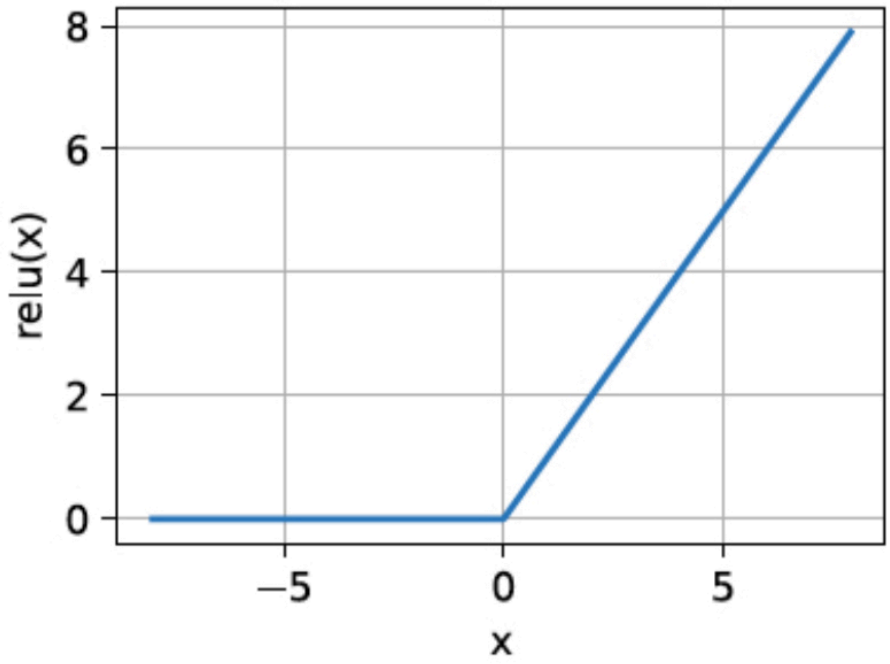
- $\sigma(t) = \frac{1}{1+e^{-t}}$
- $\sigma'(t) = \sigma(t)(1 - \sigma(t))$
- **Sigmoid confina valores no intervalo $[0.0, 1.0]$ das probabilidades**
- **Não necessariamente forma uma distribuição de probalidade**

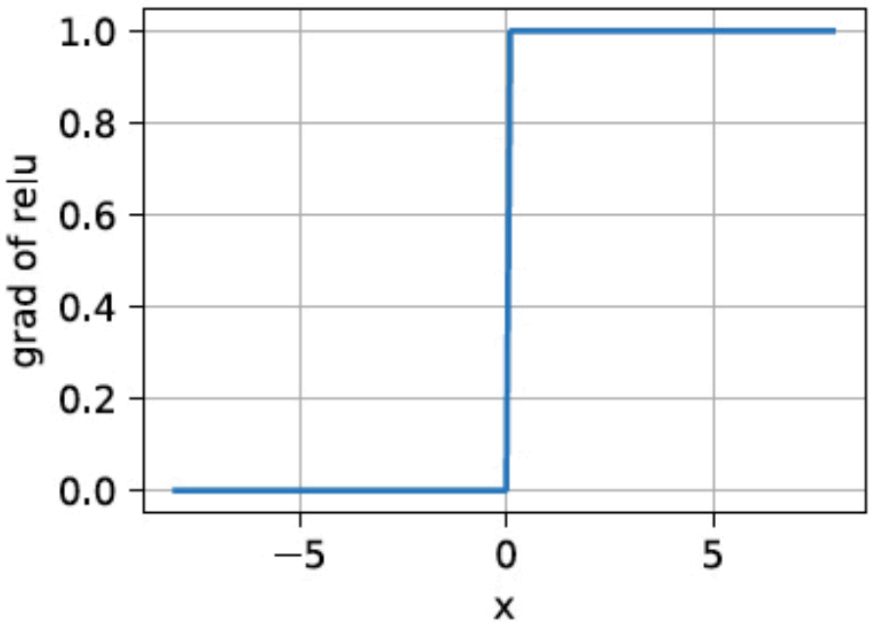




Função de Ativação ReLU

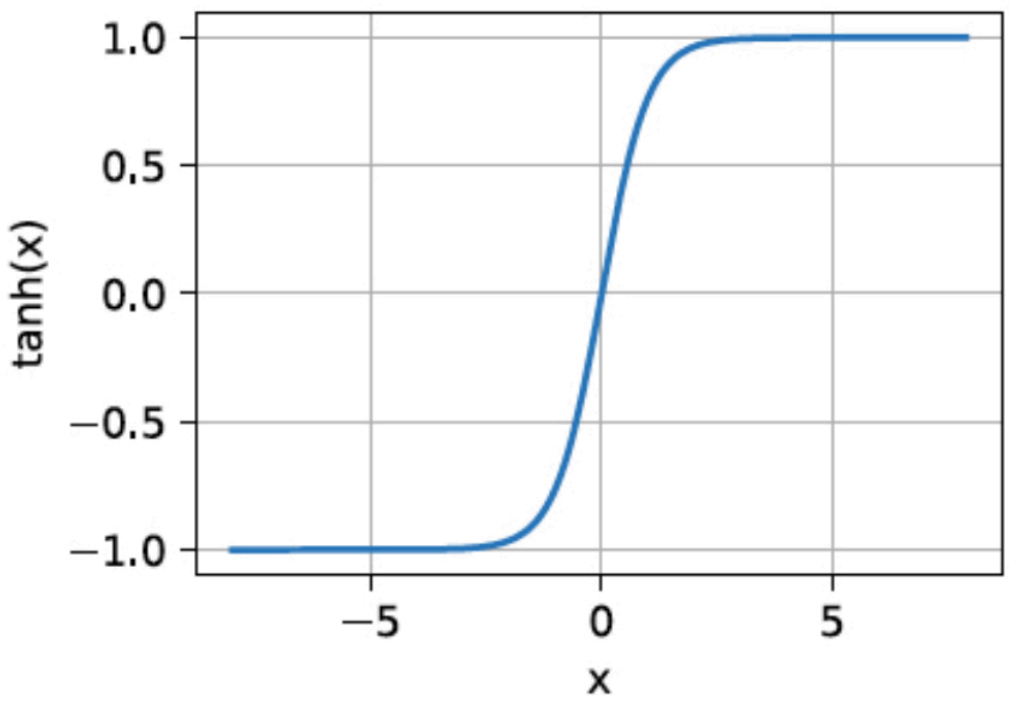
- $\sigma(t) = \max(0, t)$
- $\sigma'(t) = \begin{cases} 1 \text{ se } t > 0 \\ 0 \text{ se } t = 0 \text{ por default} \\ 0 \text{ se } t < 0 \end{cases}$

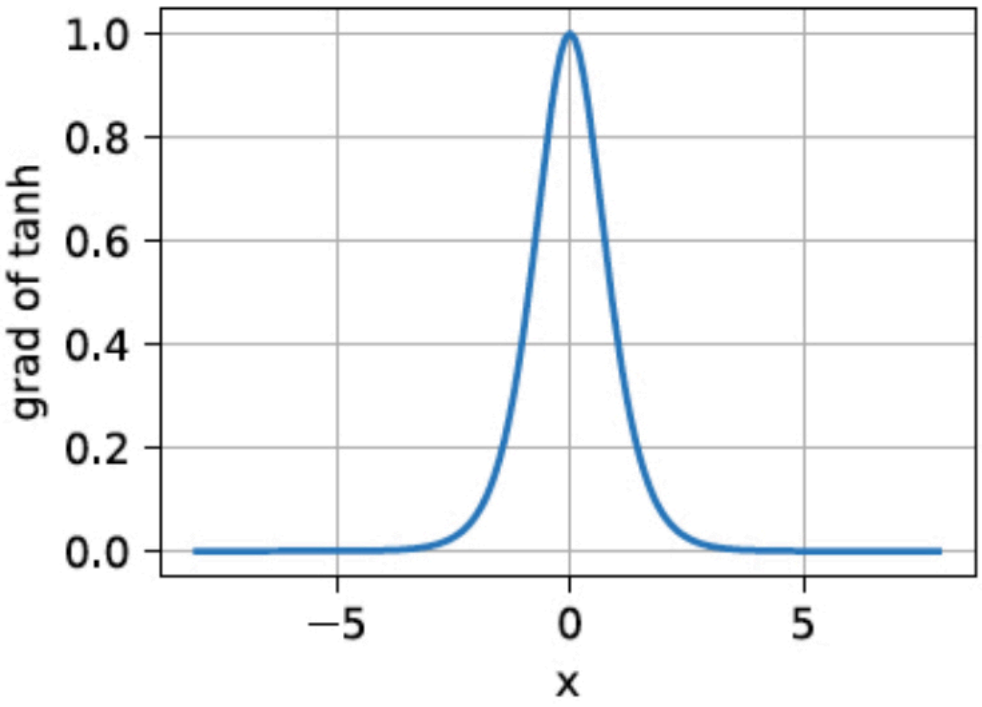




Função de Ativação $\tanh$

- $\sigma(t) = \frac{1-e^{-2t}}{1+e^{-2t}}$
- $\sigma'(t) = 1 - \tanh^2(t)$



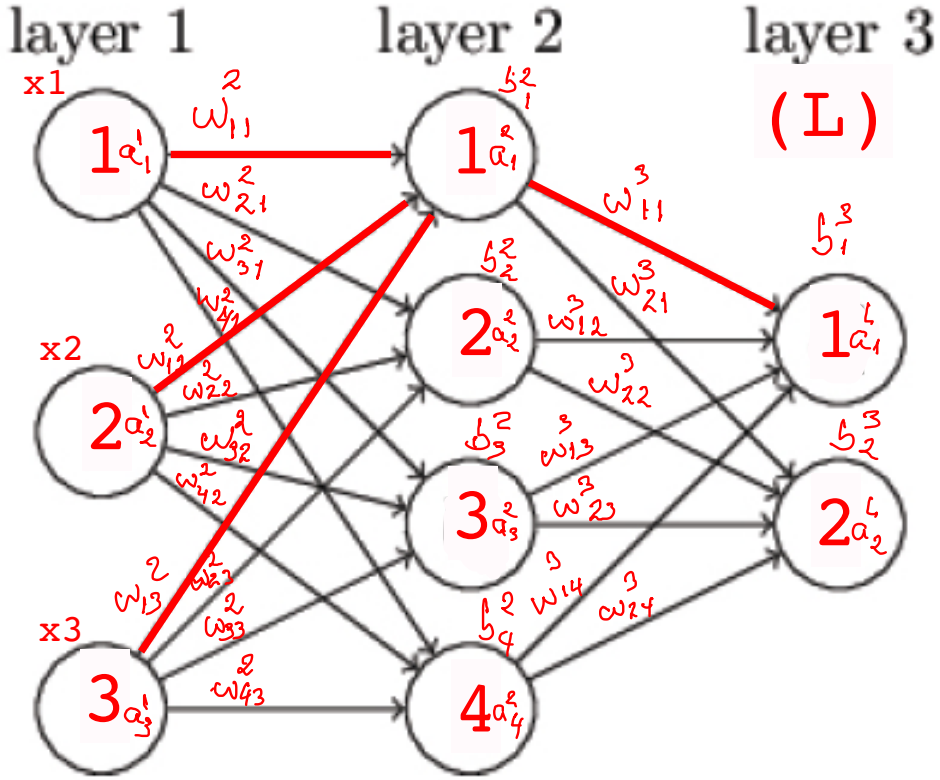


Função de Ativação softmax

- Seja $y = (y_1, \cdots, y_n)$, $y \in \mathbb{R}^n$, o vetor com os resultados da camada de saída da rede de neurônios
- $\mathrm{s}(y) = \left(\frac{e^{y_1}}{\sum_{i=1}^{n} e^{y_i}} \, , \, \cdots \, , \, \frac{e^{y_n}}{\sum_{i=1}^{n} e^{y_i}} \right)$ é a função softmax
- A função softmax $s(y)$:
  - confina os elementos de y no intervalo $[0.0, 1.0]$
  - garante que a soma dos valores em $s(y)$ seja 1
  - adequada a redes neurais de classificação
- O papel do neperiano e é assegurar que as parcelas das somas acima sejam sempre positivas, e assim garantir que a soma dos valores em $s(y)$ seja sempre 1
- Função softmax sempre gera uma distribuição de probabilidade

Forma da Função Modelada...

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



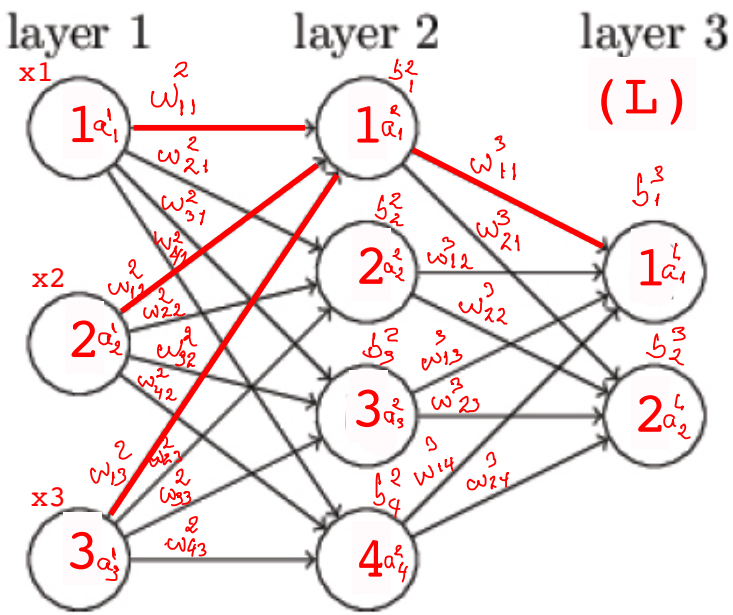
...Forma da Função Modelada...

- $a_1^3 = f(b_1^3 + \sum_{k=1}^{4} w_{1k}^3 a_k^2)$

- $a_1^3 = f(b_1^3 + w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + w_{13}^3 a_3^2 + w_{14}^3 a_4^2)$

- $$\begin{aligned} a_1^3 = f(b_1^3 &+ w_{11}^3 f(b_1^2 + w_{11}^2 a_1^1 + w_{12}^2 a_2^1 + w_{13}^2 a_3^1) \\ &+ w_{12}^3 f(b_2^2 + w_{21}^2 a_1^1 + w_{22}^2 a_2^1 + w_{23}^2 a_3^1) \\ &+ w_{13}^3 f(b_2^3 + w_{31}^2 a_1^1 + w_{32}^2 a_2^1 + w_{33}^2 a_3^1) \\ &+ w_{14}^3 f(b_2^4 + w_{41}^2 a_1^1 + w_{42}^2 a_2^1 + w_{43}^2 a_3^1)) \end{aligned}$$

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

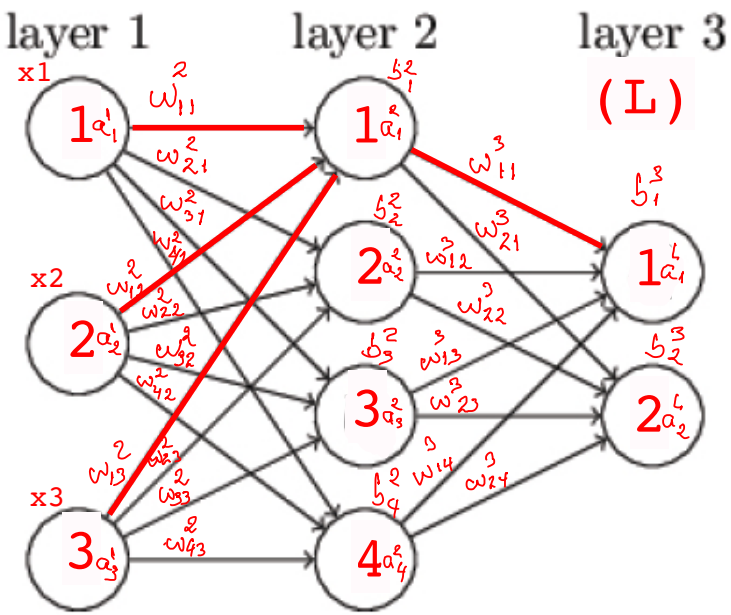


...Forma da Função Modelada...

- $a_2^3 = f(b_2^3 + \sum_{k=1}^{4} w_{2k}^3 a_k^2)$

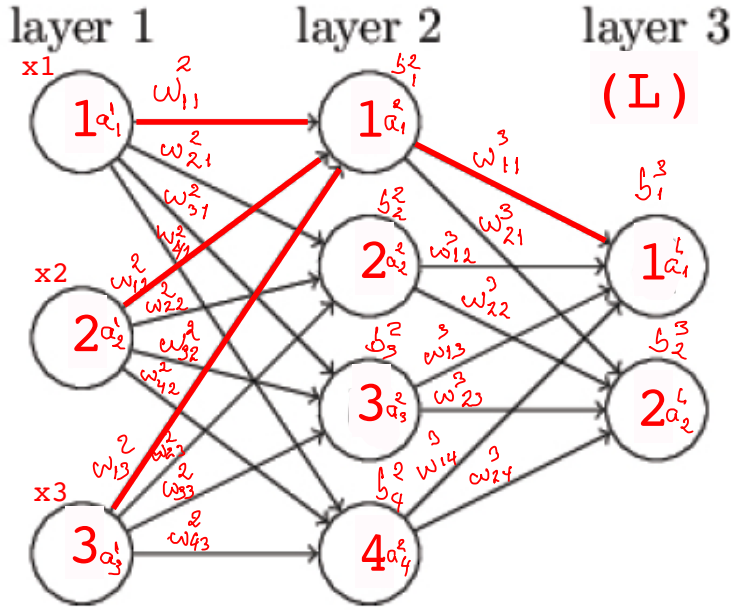
- $a_2^3 = f(b_2^3 + w_{21}^3 a_1^2 + w_{22}^3 a_2^2 + w_{23}^3 a_3^2 + w_{24}^3 a_4^2)$

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



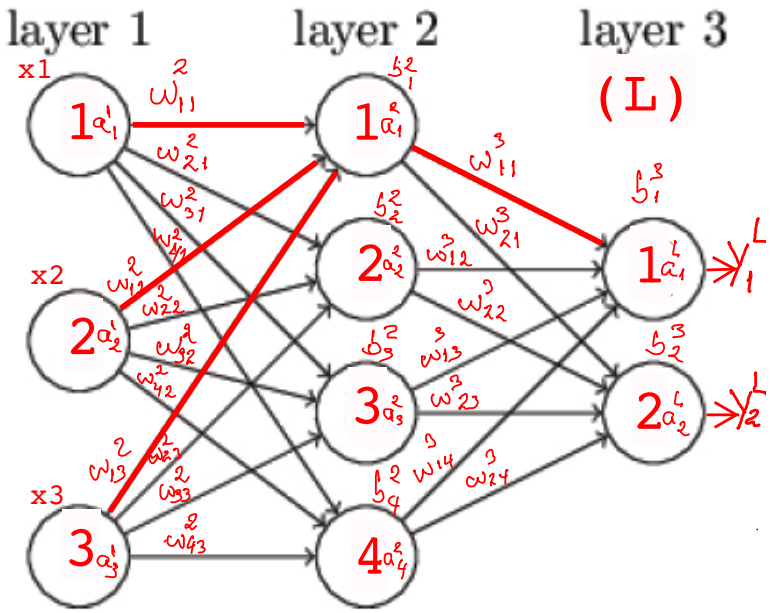
Resumo: Forma da Função Modelada

- $a_i^1 = x_i$ para $i : 1..3$
- $a_1^2 = f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3)$
- $a_2^2 = f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3)$
- $a_3^2 = f(b_3^2 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3)$
- $a_4^2 = f(b_4^2 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3)$
- $$\begin{aligned} a_1^3 = f(b_1^3 &+ w_{11}^3 f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3) \\ &+ w_{12}^3 f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3) \\ &+ w_{13}^3 f(b_2^3 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3) \\ &+ w_{14}^3 f(b_2^4 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3)) \end{aligned}$$
- $$\begin{aligned} a_2^3 = f(b_2^3 &+ w_{21}^3 f(b_1^2 + w_{11}^2 x_1 + w_{12}^2 x_2 + w_{13}^2 x_3) \\ &+ w_{22}^3 f(b_2^2 + w_{21}^2 x_1 + w_{22}^2 x_2 + w_{23}^2 x_3) \\ &+ w_{23}^3 f(b_2^3 + w_{31}^2 x_1 + w_{32}^2 x_2 + w_{33}^2 x_3) \\ &+ w_{24}^3 f(b_2^4 + w_{41}^2 x_1 + w_{42}^2 x_2 + w_{43}^2 x_3)) \end{aligned}$$



Função Modelada Linear e Não-Linear

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



- **A definição de** a_2^L **tem dois componentes: um linear em relação às features** x_1, x_2 **e** x_3 **e aos pesos das conexões e outro, que pode ser não-linear, dependendo da função de ativação** f**, e.g., a funcão** σ

- **Assim, uma rede neural tipicamente é uma representação geral de funções que têm um componente linear e outro não-linear**

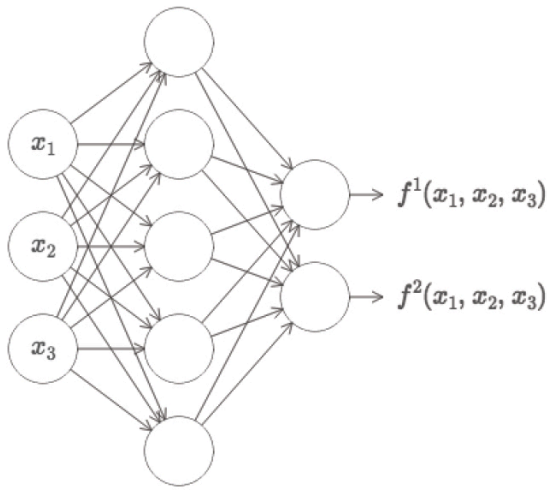
DEEP LEARNING

Camadas da Rede Neural

- Uma camada de entrada com tantos neurônios quanto o tamanho da tupla de entrada da amostra
- Uma camada de saída com tantos neurônios quanto o tamanho da tupla de resultados esperados
- Zero ou mais camadas ocultas com um número variado de neurônios determinado experimentalmente.
- Há quem recomende que esse número de neurônios (ocultos):
  - deve estar entre 2/3 do número de neurônios nas camadas de entrada e de saída e o dobro do tamanho da camada de entrada
  - deve ter pelo menos o mesmo número de neurônios da camada de entrada

Camadas Ocultas da Rede Neural

- As camadas ocultas determinam o poder de aprendizado da rede:
  - 0 camadas: capaz de representar somente funções lineares
  - 1 camada: capaz de representar qualquer função de um espaço finito a outro também finito
  - 2 ou mais camadas: capaz de representar qualquer função contínua com grande precisão



Hierarquia de Conceitos

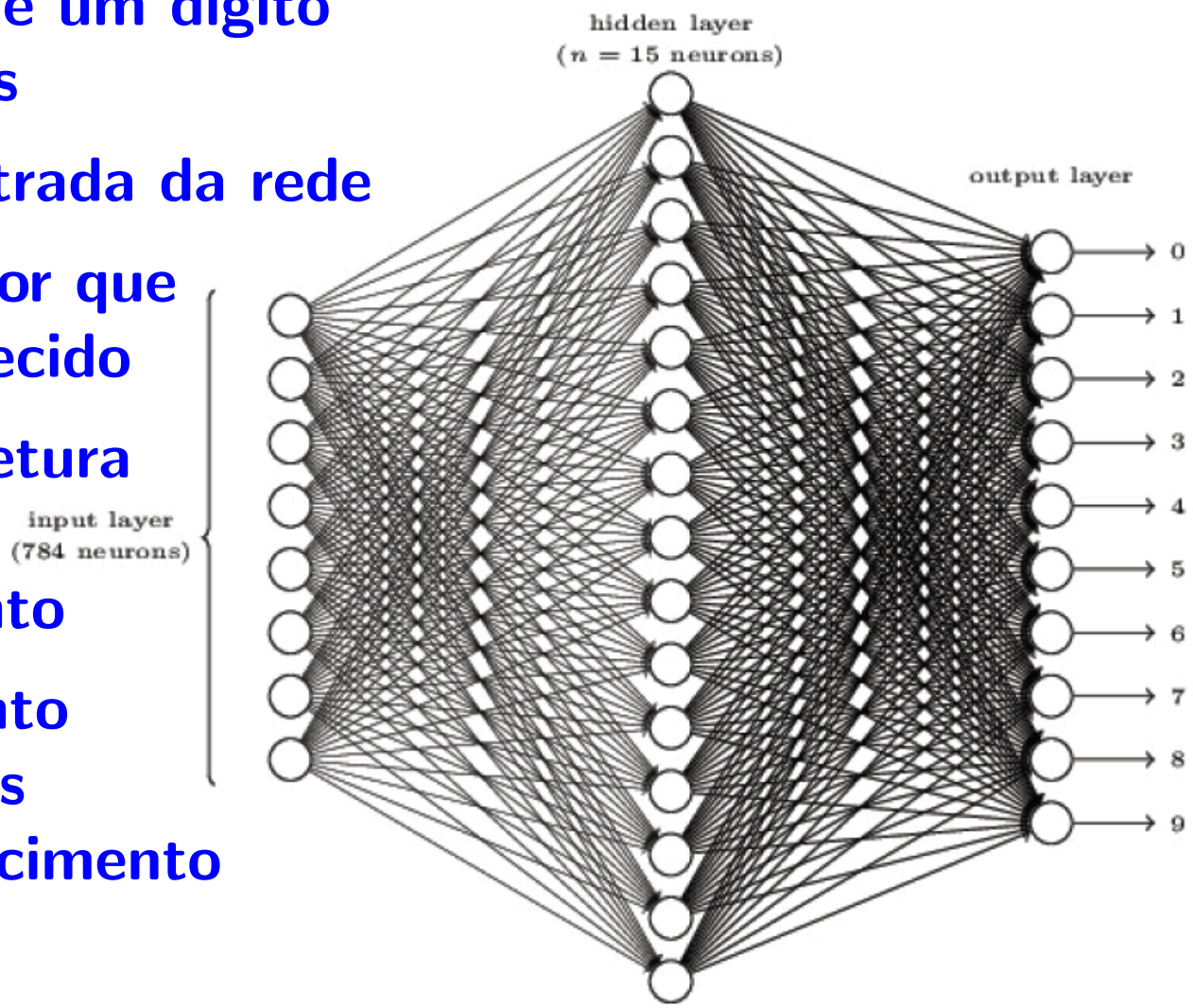
- O desafio da IA é resolver problemas cujas soluções são muito difíceis de serem descritas formalmente
- A ideia é resolver problemas compondo soluções de problemas mais simples, resolvidos, um a um, experimentalmente
- Soluções são coletadas a partir de experiências, dispensando a necessidade de especificação formal do conhecimento necessário
- A hierarquia de conceitos permite que o computador aprenda conceitos mais complexos a partir de conceitos mais simples
- Uma rede neural com apenas uma camada oculta é dita shallow
- Uma rede neural com duas ou mais camadas ocultas é dita deep

Deep Learning

- Deep Learning é uma aprendizagem com redes neurais com pelo menos quatro camadas (entrada, duas ocultas, saída)
- Nesse tipo de aprendizagem de máquina, o computador é treinado a realizar tarefas típicas de humanos, como reconhecimento de fala ou de imagens
- Deep learning submete ao computador 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) produza para novos dados respostas coerentes com as da amostra
- 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

Uma Rede para Reconhecimento de Dígitos

- Suponha que a imagem de um dígito tenha 28X28 = 784 pixels
- Esses pixels formam a entrada da rede
- A saída da rede é um vetor que identifica o dígito reconhecido
- O usuário define a arquitetura da rede neural e fornece as amostras de treinamento
- O algoritmo de treinamento descobre os pesos e biases que asseguram o reconhecimento dos dígitos



TREINAMENTO DE REDES NEURAIS DENSAS

Notação...

- N_r: número de neurônios da rede neural
- L: número de camadas da rede neural
- n: vetor contendo o número de neurônios em cada camada
- w: matrix tridimensional de pesos w^l_{jk}
- w^l_{jk}: peso da conexão do neurônio k da camada $l-1$ ao j da camada $l \in [2, L]$
- b: matrix bidimensional de biases b^l_j, que dá a medida da propensão de o neurônio produzir saída ativa, e.g., 1
- b^l_j: valor do bias do neurônio j da camada $l \in [2, L]$
- threshold: valor limite acima do qual uma synapse é produzida
- (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

...Notação...

- N_a: tamanho da amostra a ser usada no treinamento
- A: amostra de treinamento
- $(x^{(i)}, y^{(i)})$: i-ésimo item da amostra de treinamento, $1 \leq i \leq N_a$
- $x^{(i)}$: i-ésima feature ou atributo de entrada $(x_1^{(i)}, x_2^{(i)}, \cdots, x_{n_1}^{(i)})$ da amostra de treinamento
- F: vetor de tamanho n_1 com fatores para normalizar as entradas da rede neural, i.e., $a_j^1(i) = x_j^{(i)}/F_j$, $\forall i\!:\!1..N_a$ e $j\!:\!1..n_1$
- $y^{(i)}$: resultado observado para a entrada ou feature $x^{(i)}$
- C: função de custo que dá a grandeza do erro de treinamento
- α: learning rate usada no algoritmo de ajuste de pesos e biases

...Notação...

- 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 da amostra A
- 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

...Notação...

- $\partial C/\partial w$: 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
- $\partial C/\partial w^l_{jk}$: elemento da matriz $\partial C/\partial w$ contendo o valor médio das derivadas em relação ao peso w^l_{jk} 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
- $\partial C^{(i)}/\partial w^l_{jk}$: elemento da matriz $\partial C^{(i)}/\partial w$ contendo o valor da derivada em relação ao peso w^l_{jk} 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

...Notação

- $\partial C/\partial b$: 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
- $\partial C^{(i)}/\partial b_j^l$: 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 da função logística $\sigma'(t) = \sigma(t)(1 - \sigma(t))$

O Treinamento da Rede Neural

T1. Defina arquitetura da rede de neurônios (L, n, w, b)

T2. Prepare a amostra $A = \{(x, y)\}$ para treinamento

T3. Deduza a função de custo $C(x, y, w, b)$

T4. Selecione método de minimização da função de erro C da Rede

T5. Deduza as derivadas parciais de C em relação a w e b

T6. Ajuste a rede (L, n, w, b) para garantir custo C mínimo

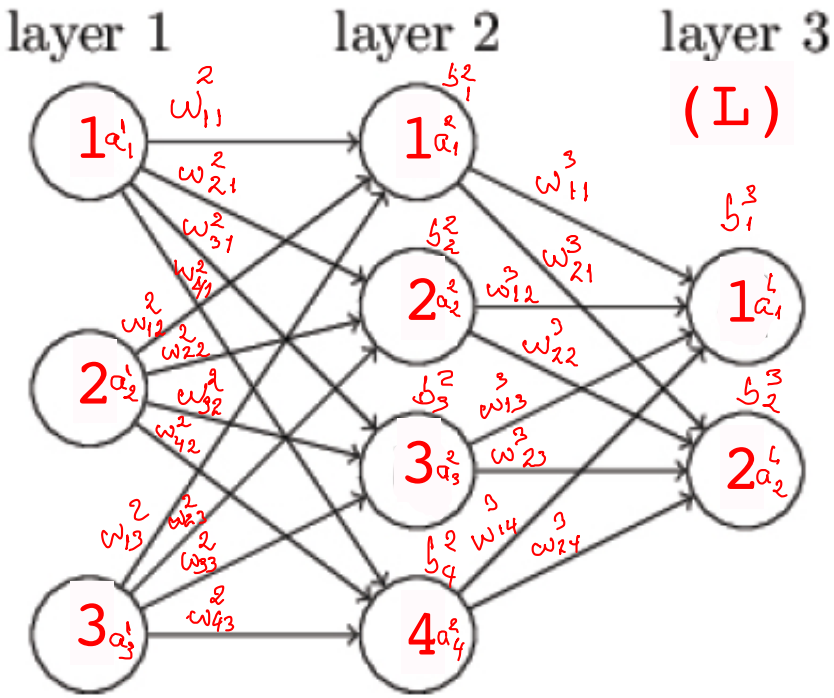
RESULTADO:

A rede (L, n, w, b) obtida é a rede treinada com a amostra A, tal que, se iniciada com uma nova entrada produz uma saída compatível com o padrão dos resultados identificado na amostra A

T1: Definição da Arquitetura da Rede

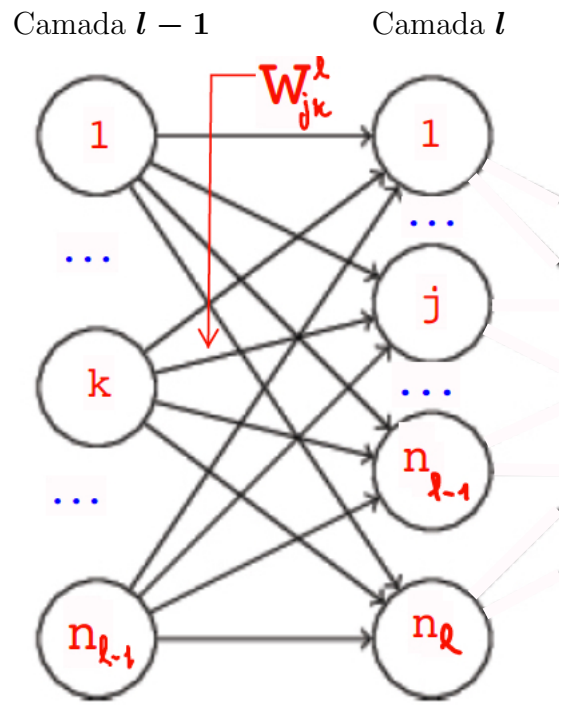
Arquitetura de uma Rede (L, n, w, b, α)...

- L: número de camadas (≥ 2)
- n: vetor com número de neurônios de cada camada, e.g., $n = (3, 4, 2)$
- Camada de entrada: a_j^1, $j \in [1, n_1]$
- Camada de saída: a_j^L, $j \in [1, n_L]$
- a_j^l: ativação na saída do neurônio j da camada $l \in [1, L]$
- α: taxa de aprendizado (learning rate)
- b: array contendo todos os biases b_j^l
- b_j^l: valor do bias do neurônio $j \in [1, n_l]$ da camada $l \in [2, L]$



...Arquitetura de uma Rede (L, n, w, b, α)...

- w^l_{jk}: 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
- O peso de cada uma dessas conexões é w^l_{jk}
- w: array contendo todos os pesos w^l_{jk}



...Arquitetura de uma Rede (L, n, w, b, α)...

- A: amostra de treinamento

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

  $A = \{(x, y)\}$, onde x e y são matrizes de entrada e resultados

- i: índice de uma entrada da amostra A
- x: matriz N_a X n_1 contendo todas as entradas (features) da amostra
- y: matriz N_a X n_L contendo as saídas previstas para as entradas correspondentes em x
- $a_j^l(i)$: ativação do neurônio j da camada l com i-ésima entrada da amostra A
- σ: função logística (sigmoid)

  $$\sigma(t) = \frac{1}{1+e^{-t}} \Longrightarrow \sigma'(t) = \sigma(t)(1 - \sigma(t))$$

...Arquitetura de uma Rede (L, n, w, b, α)

Inicialização dos parâmetros de (L, n, w, b):

- **Defina número de camadas L**
- **Initialize vetor n com o número de neurônios por camada**
- **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^l_j: **valor do bias do neurônio j da camada $l \in [2, L]$**
- **Inicialize a learning rate α com um valor definido experimentalmente**

T2: Preparação da Amostra de Treinamento

Cálculo do Fator de Normalização da $A = \{(x, y)\}$

1. Dada a amostra de treinamento A

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

2. Determine fator de normalização F, para que as escalas dos valores das features x sejam de mesma ordem de grandeza, preferencialmente no intervalo $[0, 1]$ em $\mathbb{R}$

3. Por exemplo, supondo

$$n_1 = 2,\ x_1^{(i)} \in [50, 10.000],\ x_2^{(i)} \in [1, 10]\ ,\ i : 1..N_a,$$

recomenda-se definir F da seguinte maneira:

para $j : 1..n_1$

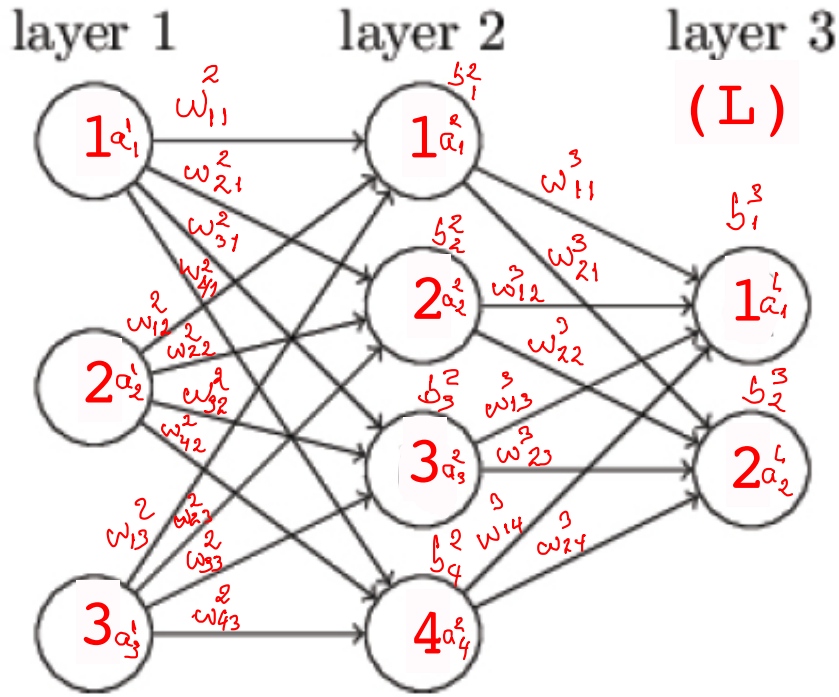
faça $F_j := \max(x_j^{(i)})$, para $i : 1..N_a$

4. Caso a normalização das features não seja necessária,

faça $F_j := 1$, para $j : 1..n_1$

T3: Dedução da Função de Custo C

Função de Custo C de (L, n, w, b)



- **Entrada da rede para cada item i:**
  $a_j^1(i) = x_j^{(i)}/F_j,\ j : 1..n_1$
- **Ativação neurônio $j \in [1, n_l]$ da camada $l \in [2, L]$ para item i:**
  $a_j^l(i) = \sigma(z_j^l(i))$, **onde**
  $z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$
- **Erro do neurônio j da camada L:**
  $C_j^{(i)}(x, y, w, b) = \frac{1}{2}\left(a_j^L(i) - y_j^{(i)}\right)^2,\ j \in [1, n_L]$
- **Erro de todos neurônios da camada L para item i:**
  $C^{(i)}(x, y, w, b) = \sum_{j=1}^{n_L} C_j^{(i)}(x, y, w, b)$
- **Erro toda amostra A:** $C(x, y, w, b) = \frac{1}{N_a}\sum_{i=1}^{N_a} C^{(i)}(x, y, w, b)$

T4: Métodos de Minimização da Função de Erro

Método Analítico

- A minimização da função de erro
  $C(x,y,w,b) = \frac{1}{N_a}\left(\sum_{i=1}^{N_a}\sum_{j=1}^{n_L} C_j^{(i)}(x,y,w,b)\right)$
- Pode ser obtida analiticamente, resolvendo as equações:

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

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

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

Método Iterativo

- 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.,

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

- Assim, de acordo com o algoritmo do Gradient Descent, 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

$$b^l_j \quad := b^l_j - \alpha\frac{\partial C(x,y,w,b)}{\partial b^l_j}$$

onde α é o learning rate

Formulação do Problema...

- O custo C de treino de toda a amostra A:

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

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

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

$$a_j^l(i) = \sigma(z_j^l(i)), \text{ onde } \sigma(t) = \frac{1}{1+e^{-t}}$$

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

$$a_j^1(i) = x_j^{(i)}/F_j \text{ para } j \in [1, n_1]$$

...Formulação do Problema

- **Considerando** $C \equiv C(x, y, w, b)$

$$C^{(i)} \equiv C^{(i)}(x, y, w, b)$$
$$C_j^{(i)} \equiv C_j^{(i)}(x, y, w, b)$$

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

  onde $C^{(i)} = \sum_{j=1}^{n_L} C_j^{(i)}$

$$C_j^{(i)} = \frac{1}{2}\left(a_j^L(i) - y_j^{(i)}\right)^2$$

- **Deseja-se calcular**

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

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

T5: Cálculo das Derivadas $\frac{\partial C^{(i)}}{\partial w_{jk}^{l}}$ e $\frac{\partial C^{(i)}}{\partial b_{j}^{l}}$

Método Backpropagation

Gradiente Local

- Backpropagation calcula as derivadas parciais da função que defina a saída de uma rede neural em relação a seus pesos e biases
- Essas derivadas permitem identificar como alterações nos pesos e biases modificam o custo na saída de rede para cada item da amostra
- Backpropagation calcula os componentes do gradiente local:
  $\delta_j^l(i) \equiv \partial C^{(i)}/\partial z_j^l(i)$, $l\!:\!1..L$ 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

Derivadas do Custo para um Item da Amostra

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

- **As derivadas do custo para o i-ésimo item da amostra são:**

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

$$\frac{\partial C^{(i)}}{\partial b_j^l} = \underbrace{\frac{\partial C^{(i)}}{\partial z_j^l(i)}}_{\delta_j^l(i)} \underbrace{\frac{\partial z_j^l(i)}{\partial b_j^l}}_{1} = \delta_j^l(i), \text{ para } l : 2..L \text{ e } j : 1..n_l$$

Derivadas do Custo para Toda a Amostra

- **As derivadas do custo total relativas a pesos e biases são as médias das respectivas derivadas obtidas com os exemplos individuais de treinamento:**

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

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

onde

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

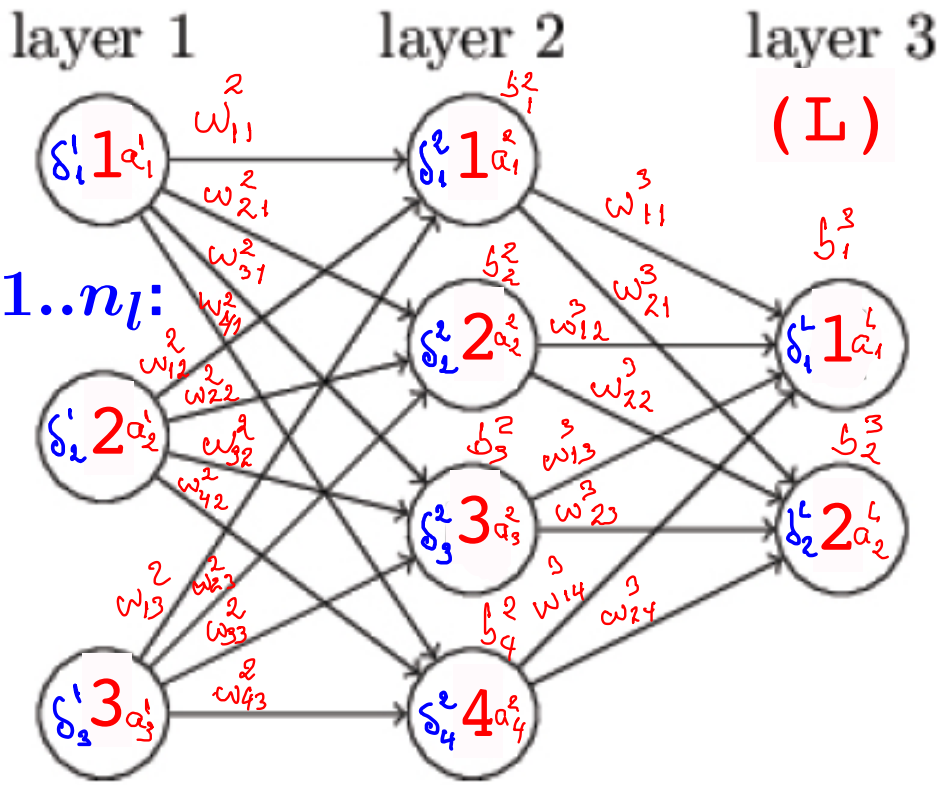
Formulação do Forward Propagation

- **Considere item i da amostra A de treinamento:**
  $A = \left\{ ((x_1^{(i)}, \cdots, x_{n_1}^{(i)}), (y_1^{(i)}, \cdots, y_{n_L}^{(i)})), \text{for } i \in [1, N_a] \right\}$

- **Ativação da entrada:**
  $a_j^1(i) = x_j^{(i)} / F_j,\ j : 1..n_1$

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

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



Formulação de $\delta_j^L(i)$ da Camada de Saída

- **Tem-se que**

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

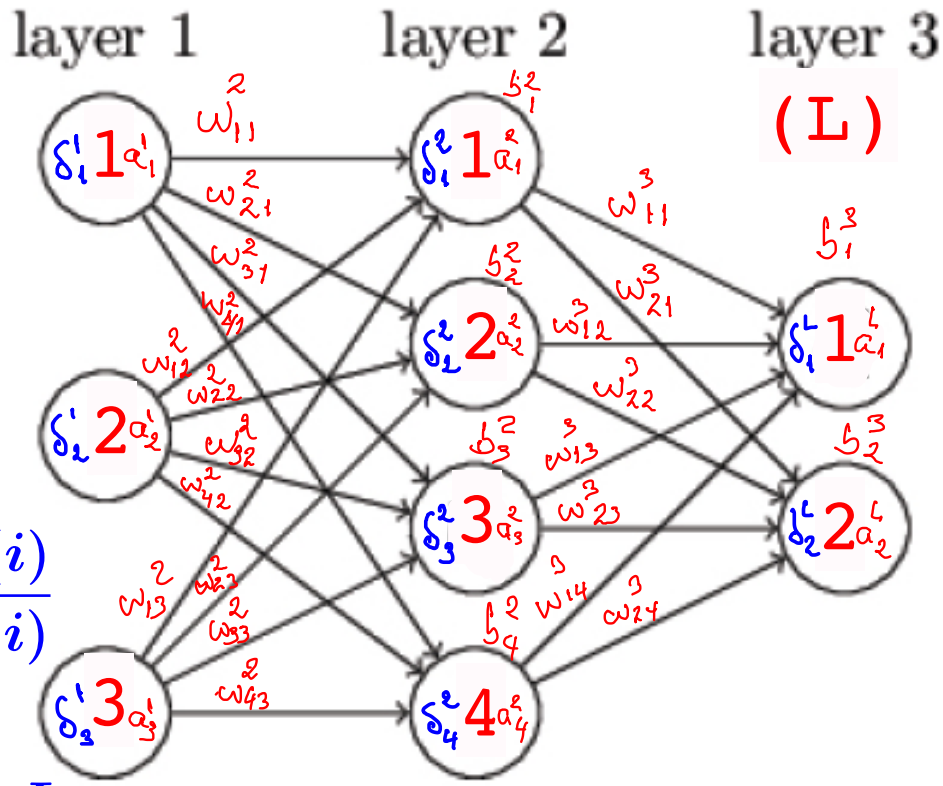
$$C_k^{(i)} = \frac{1}{2}\left(a_k^L(i) - y_k^{(i)}\right)^2$$

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

- **Então o erro na saída j é:**

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

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



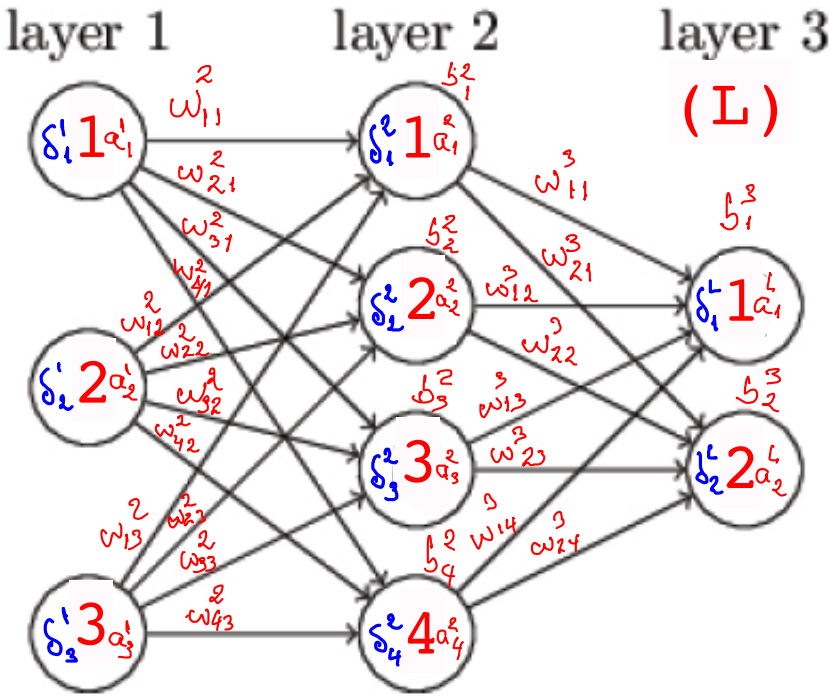
$$\Longrightarrow \boxed{\delta_j^L(i) = (a_j^L(i) - y_j^{(i)})\ \sigma'(z_j^L(i)), \text{ onde } \sigma'(t) = \sigma(t)(1-\sigma(t))}$$

Formulação de $\delta_j^l(i)$ das Demais Camadas

Tem-se que:

- $C^{(i)} = \sum_{k=1}^{n_L} C_k^{(i)}$
- **para** $l : 2..L$ **e** $j : 1..n_l$**:**
  $z_j^l(i) = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$
- **para** $l : 1..L-1$ **e** $k : 1..n_l$**:**
  $z_k^{l+1}(i) = \sum_{j=1}^{n_l} w_{kj}^{l+1} \sigma(z_j^l(i)) + b_k^{l+1}$

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



Então:

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

$$\Longrightarrow \boxed{\delta_j^l(i) = \sum_{k=1}^{n_L} w_{kj}^{l+1} \delta_k^{l+1}(i) \sigma'(z_j^l(i)), \text{ onde } \sigma'(t) = \sigma(t)(1-\sigma(t))}$$

Taxas de Alteração de Pesos e Biases

- **Supondo $\sigma(t) = t$ para simplificar, tem-se:**

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

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

$$a_j^l(i) = \sigma(z_j^l(i)), \text{ para } l : 2..L \text{ e } j : 1..n_l,$$

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:**

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

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

Resumo: Propagations forward and backward

1. **Considere item i da amostra A:**

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

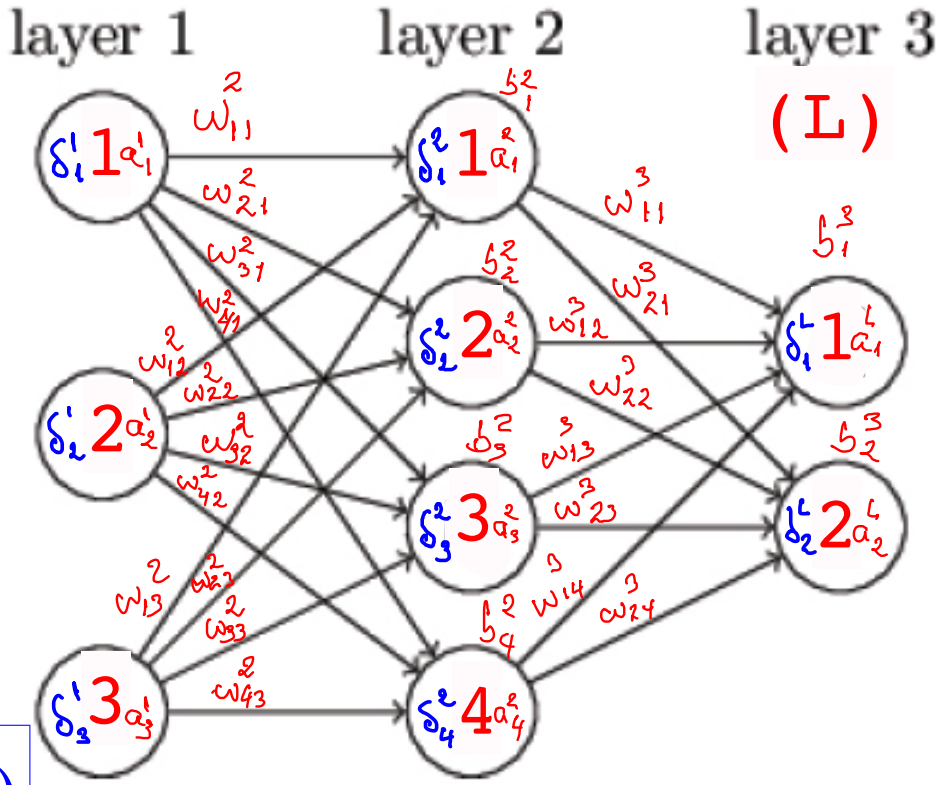
- $a_j^1(i) = x_j^{(i)}/F_j,\ j : 1..n_1$

  $\delta_j^L(i) = a_j^L(i) - y_j^{(i)},\ j : 1..n_L$
- **para** $l : 2..L$ **e** $j : 1..n_l$

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

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

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



2. $\boxed{\dfrac{\partial C^{(i)}}{\partial b_j^l} = \delta_j^l(i)}$ **e** $\boxed{\dfrac{\partial C^{(i)}}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l(i)}$

Resumo: Cálculo de $\delta_j^l(i)$

- **Custo do treinamento do item i:**
  $C^{(i)} = \sum_{k=1}^{n_L} C_k^{(i)}$
- **para $j:1..n_L$, tem-se:**

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

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

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

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

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

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

- **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 $j:1..n_l$:**

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

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

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

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

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

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

T6: Determinação da Rede de Custo Mínimo

Algoritmo do Treinamento da 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 custo $C(x, y, w, b)$
5. Repita 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 $C(x, y, w, b)$

ANÁLISE DA COMPLEXIDADE DO BACKPROPAGATION

Convenções para Análise da Complexidade

Para simplificar a avaliação da complexidade do algoritmo de treinamento, sem perda de precisão, convencionam-se:

- L, que é o número de camadas da rede neural, seja tratado como uma constante de pequeno valor comparado com N_r e N_a
- Supõe-se que todas as camadas da rede tenham o mesmo número m de neurônios, i.e., $m = N_r/L$
- $\sigma = \frac{1}{1-e^t}$ substituído por $\sigma(t) = t$, pois ambas em custo $O(1)$
- A seguir, determinam-se as complexidades das principais operações do algoritmo de treiamento da rede neural

Definição da Rede $R = (L, n, w, b)$ a Ser Treinada

1. Arbitre um valor para L:
   $L :=$ valor arbitrário $\Longrightarrow$ custo $= 1$

2. para $l : 1..L$ faça $n_l :=$ valor escolhido $\Longrightarrow$ custo $= L$

3. para $l : 2..L$, $j : 1..\overbrace{n_l}^{N_r/L}$ e $k : 1..\overbrace{n_l}^{N_r/L}$ faça $w^l_{jk} :=$ valor escolhido
   $\Longrightarrow$ custo $= (L-1)n_l^2 = (L-1)(\frac{N_r}{L})^2 = \frac{L-1}{L^2}N_r^2$

4. para $l : 2..L$, $j : 1..n_l$ faça $b^l_j :=$ valor escolhido
   $\Longrightarrow$ custo $= (L-1)n_l = (L-1)\frac{N_r}{L} = \frac{L-1}{L}N_r$

CUSTO TOTAL: $T_i(N_r) = 1 + L + \frac{L-1}{L^2}N_r^2 + \frac{L-1}{L}N_r$

$$\boxed{T_i(N_r) = O(N_r^2)}$$

Cálculo do Fator de Normalização das Features

1. para $j : 1..n_1$
   faça $F_j := \max(x_j^{(i)})$, para $i : 1..N_a$ $\Bigg\} \Longrightarrow$ custo $= \frac{N_r}{L} N_a$

CUSTO: $\boxed{T_f(Na, N_r) = O(N_a N_r)}$

Cálculo da Matriz a

1. para $i : 1..N_a$ e $j : 1..n_1$
   faça $a_j^1(i) := x_j^{(i)}/F_j$ $\Longrightarrow$ custo $= N_a \frac{N_r}{L}$

2. para $i : 1..N_a$, $l : 2..L$ e $j : 1..n_l$
   faça $a_j^l(i) := \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1}(i) + b_j^l$ $\Longrightarrow$ custo $= N_a(L-1)\frac{N_r}{L}\frac{N_r}{L}$

CUSTO: $T_a(N_a, N_r) = N_a \frac{N_r}{L} + N_a(L-1)\frac{N_r}{L}\frac{N_r}{L} = N_a \frac{N_r}{L} + \frac{L-1}{L^2} N_a N_r^2$

$$\boxed{T_a(N_a, N_r) = O(N_a N_r^2)}$$

Cálculo da Matriz δ

Suponha que as matrizes a e w já tenham sido calculadas:

1. para $i : 1..N_a$ e $j : 1..n_L$
   faça $\delta_j^L(i) := \underbrace{a_j^L(i)}_{O(1)} - y_j^{(i)}$ $\Longrightarrow$ custo $= N_a \frac{N_r}{L}$

2. para $i : 1..N_a, l : L-1..1$ e $j : 1..n_l$
   faça $\delta_j^l(i) := \sum_{k=1}^{n_{l+1}} w_{kj}^{l+1} \delta_k^{l+1}(i)$ $\Longrightarrow$ custo $= N_a(L-1)\frac{N_r}{L}\frac{N_r}{L}$

CUSTO: $\boxed{T_\delta(N_A, N_r) = O(N_a N_r^2)}$

Cálculo da Função de Custo C

Suponha que a matriz a já tenha sido calculada:

1. $C(x,y,w,b) := \frac{1}{N_a}(\sum_{i=1}^{N_a}\sum_{j=1}^{n_L}\frac{1}{2}(\overbrace{a_j^L(i)}^{O(1)} - y_j^{(i)})^2$ $\Bigg\}$ **custo** $= N_a\frac{N_r}{L}k_1$

CUSTO:

- **Suponha que a computação de C sempre ocorra após a computação da matrix a, de complexidade $T_a(N_a, N_r)$, ter sido concluída**

- **Assim, a complexidade do cálculo do custo C é:**

$$\boxed{T_c(N_a, N_r) = O(N_a N_r)}$$

Cálculo da Matriz $\partial C/\partial b$

Suponha que a matriz δ já tenha sido calculada:

1. para $l : L..2$ e $j : 1..n_l$
   faça $\left.\dfrac{\partial C}{\partial b_j^l} := \dfrac{1}{N_a}\sum_{i=1}^{N_a}\dfrac{\partial C^{(i)}}{\partial b_j^l}\right\} \Longrightarrow$ custo $= (L-1)\,\frac{N_r}{L}\,N_a\,k_1$

   onde $\dfrac{\partial C^{(i)}}{\partial b_j^l} := \overbrace{\delta_j^l(i)}^{O(1)} \Longrightarrow$ custo $= O(1)$

CUSTO:

$$\Longrightarrow \quad T_{\partial b}(N_a, N_r) = k_1\left(\frac{L-1}{L}\right)N_a N_r$$

$$\Longrightarrow \quad \boxed{T_{\partial b}(N_a, N_r) = O(N_a N_r)}$$

Cálculo da matriz $\partial C/\partial w$

Suponha que as matrizes a e δ já tenham sido calculadas

1. para $l : L..2, j : 1..n_l$ e $k : 1..n_{l-1}$
   faça $\dfrac{\partial C}{\partial w^l_{jk}} := \dfrac{1}{N_a}\sum_{i=1}^{N_a} \dfrac{\partial C^{(i)}}{\partial w^l_{jk}}$ $\Longrightarrow$ custo $= (L-1)\frac{N_r}{L}\frac{N_r}{L}N_a k_1$

   onde $\dfrac{\partial C^{(i)}}{\partial w^l_{jk}} := a^{l-1}_k(i)\delta^l_j(i) \Longrightarrow$ custo $= O(1)$

CUSTO:

$$\Longrightarrow T_{\partial w}(N_a, N_r) = (L-1)(\tfrac{N_r}{L})(\tfrac{N_r}{L})(N_a)(k_1)$$

$$\Longrightarrow T_{\partial w}(N_a, N_r) = \tfrac{L-1}{L^2}(N_r^2)(N_a)[k_1]$$

$$\boxed{\Longrightarrow T_{\partial w}(N_a, N_r) = O(N_a N_r^2)}$$

Computação dos Ajustes nos Pesos e Biases

Suponha que as matrizes $\partial C/\partial w$ e $\partial C/\partial b$ já tenham sido calculadas

1. para $l:2..L, j:1..n_l, k:1..n_{l-1}$
   faça $w^l_{jk} := w^l_{jk} - \alpha * \underbrace{\partial C/\partial w^l_{jk}}_{O(1)}$ $\Longrightarrow$ custo $= (L-1)\frac{N_r}{L}\frac{N_r}{L}$

2. para $l:2..L$ e $j:1..n_l$
   faça $b^l_j := b^l_j - \alpha * \underbrace{\partial C/\partial b^l_j}_{O(1)}$ $\Longrightarrow$ custo $= (L-1)\frac{N_r}{L}$

CUSTO: $T_p(N_a, N_r) = (L-1)\frac{N_r}{L}\frac{N_r}{L} + (L-1)\frac{N_r}{L}$

$$\boxed{T_p(N_a, N_r) = O(N_r^2)}$$

Algoritmo do Treinamento da Rede Neural

1. Defina a rede neural $R = (L, n, w, b) \Longrightarrow T_i(N_r) = k_0 N_r^2$
2. Compute fator M de normalização $\Longrightarrow T_f(N_a, N_r) = k_1 N_a N_r$
3. Compute matrizes a e $\delta \Longrightarrow T_a(N_a, N_r) + T_\delta(N_a, N_r) = k_2 N_a N_r^2 + k_3 N_a N_r^2$
4. Compute custo $C(x, y, w, b) \Longrightarrow T_c(N_a, N_r) = k_4 N_a N_r$
5. Repita até $C(x, y, w, b)$ convergir: (suponha v repetições)
   - calcule a matriz $\partial C/\partial b \Longrightarrow T_{\partial b}(N_a, N_r) = k_5 N_a N_r$
   - calcule a matriz $\partial C/\partial w \Longrightarrow T_{\partial w}(N_a, N_r) = k_6 N_a N_r^2$
   - compute os ajustes dos pesos e biases $\Longrightarrow T_p(N_a, N_r) = k_7 N_r^2$
   - recompute a e $\delta \Longrightarrow T_a(N_a, N_r) + T_\delta(N_a, N_r) = k_2 N_a N_r^2 + k_3 N_a N_r^2$
   - recompute $C(x, y, w, b) \Longrightarrow T_c(N_a, N_r) = k_4 N_a N_r$

CUSTO: $k_0 N_r^2 + k_1 N_a N_r + (k_2 + k_3) N_a N_r^2 + k_4 N_a N_r + v((k_4 + k_5) N_a N_r + (k_2 + k_3 + k_6) N_a N_r^2 + k_7 N_r^2)$

$$\Longrightarrow \boxed{T_t(N_a, N_r) = O(v N_a N_r^2)}$$

Cálculo Simplificado das Derivadas $\frac{\partial C}{\partial w_{jk}^l}$ e $\frac{\partial C}{\partial b_j^l}$

Ajuste dos Pesos e Biases

- **As alterações iterativas de w^l_{jk} e b^l_j são definidas por:**

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

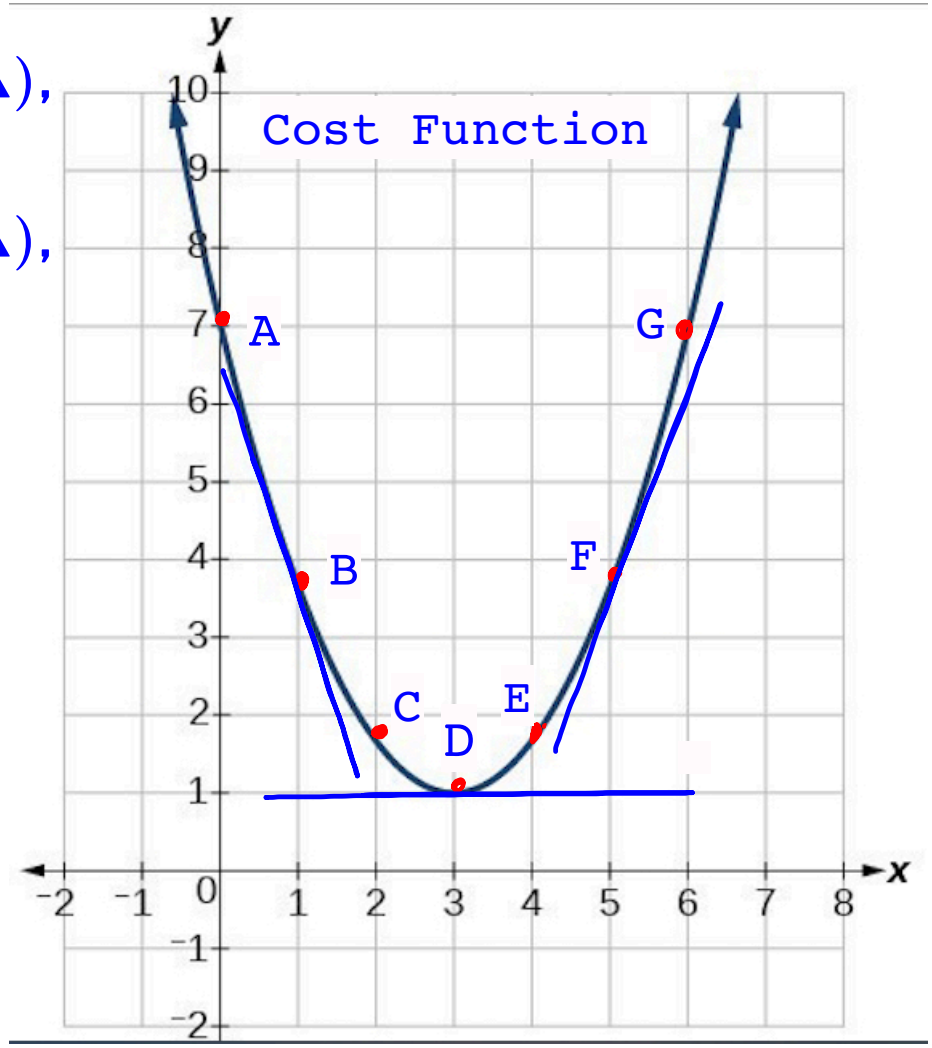
  $b^l_j = b^l_j - \alpha \frac{\partial C(x,y,w,b)}{\partial b^l_j}$, para $l: 2..L$ e $j: 1..n_l$

- **E requerem repetidos recálculos de derivadas parciais de**
  $C^{(i)}(x,y,w,b) = \frac{1}{n_L}\left(\sum_{j=1}^{n_L} C^{(i)}_j(x,y,w,b)\right)$

- **A seguir, a título de exercício, apresenta-se um método alternativo ao backpropagation**

Determinação do Sinal da Tangente

- Suponha apenas um peso x e $\Delta = 1$
- Se $(f(x-\Delta) < f(x)) \wedge (f(x) < f(x+\Delta))$, $\Longrightarrow f'(x) < 0$
- Se $(f(x-\Delta) > f(x)) \wedge (f(x) > f(x+\Delta))$, $\Longrightarrow f'(x) > 0$
- For example:
  - $x = 1$: $f(1-\Delta) > f(1) \wedge f(1) > f(1+\Delta)$ $\Longrightarrow f'(1) < 0$
  - $x = 2$: $f(2-\Delta) > f(2) \wedge f(2) > f(2+\Delta)$ $\Longrightarrow f'(2) < 0$
  - $x = 3$: $f(3-\Delta) > f(3) \wedge f(3) < f(3+\Delta)$ $\Longrightarrow f'(3) = 0$
  - $x = 4$: $f(4-\Delta) < f(4) \wedge f(4) < f(4+\Delta)$ $\Longrightarrow f'(4) > 0$
  - $x = 5$: $f(5-\Delta) < f(5) \wedge f(5) < f(5+\Delta)$ $\Longrightarrow f'(5) > 0$



Cálculo Simplificado das Derivadas Parciais

- Dado $J = C(x, y, w, b)$, faça $\Delta_w = 0.01$ and $\Delta_b = 0.01$
- Para cada $l : 2..L$, $j : 1..n_l$ e $k : 1..n_{l-1}$, definem-se

1. $$\partial C(x, y, w, b)/\partial w^l_{jk} = \begin{cases} 1 & \text{if } (w_1 > J \wedge J > w_2) \\ -1 & \text{if } (w_1 < J \wedge J < w_2) \\ 0 & \text{otherwise} \end{cases}$$

   onde $w_1 := C(x, y, [w|w^l_{jk} := w^l_{jk} - \Delta_w * w^l_{jk}], b)$

   $w_2 := C(x, y, [w|w^l_{jk} := w^l_{jk} + \Delta_w * w^l_{jk}], b)$

2. $$\partial C(x, y, w, b)/\partial b^l_j = \begin{cases} 1 & \text{if } (b_1 > J \wedge J > b_2) \\ -1 & \text{if } (b_1 < J \wedge J < b_2) \\ 0 & \text{otherwise} \end{cases}$$

   onde $b_1 := C(x, y, w, [b|b^l_j := b^l_j - \Delta_b * b^l_j])$

   $b_2 := C(x, y, w, [b|b^l_j := b^l_j + \Delta_b * b^l_j])$

Complexidade do Cálculo do Sinal de $\partial C/\partial w^l_{jk}$

- **Dado** $J = C(x, y, w, b)$, **faça** $\Delta_w = 0.01$ **and** $\Delta_b = 0.01$, **sendo** $T_c(N_a, N_r) = O\left(N_a N_r\right)$ **ou** $T_c(N_a, N_r) = k_1 N_a N_r$

- **Para cada** $l : 2..L$, $j : 1..n_l$ **e** $k : 1..n_{l-1}$, **define-se**

$$-\partial C(x, y, w, b)/\partial w^l_{jk} = \left\{\begin{array}{ll} 1 & \text{if } (w_1 > J \wedge J > w_2) \\ -1 & \text{if } (w_1 < J \wedge J < w_2) \\ 0 & \text{otherwise} \end{array}\right\} \Longrightarrow \quad 1 + 2k_2 N_r^2$$

$$\text{onde } w_1 := C(x, y, [w|w^l_{jk} := w^l_{jk} - \Delta_w * w^l_{jk}], b) \Longrightarrow \quad k_2 N_r^2$$

$$w_2 := C(x, y, [w|w^l_{jk} := w^l_{jk} + \Delta_w * w^l_{jk}], b) \Longrightarrow \quad k_2 N_r^2$$

$\Longrightarrow \; T_{\partial w}(N_a, N_r) =$ custo de $J + (L-1)\; n_l\; n_{l-1}(1 +$ custo de w_1 e $w_2)$

$\Longrightarrow \; T_{\partial w}(N_a, N_r) = k_1 N_a N_r + (L-1)\frac{N_r}{L}\frac{N_r}{L}(1 + 2k_2 N_r^2)$

$\Longrightarrow \; T_{\partial w}(N_a, N_r) = k_1 N_a N_r + \frac{L-1}{L^2} N_r^2 + 2k_2 \frac{L-1}{L^2} N_r^4$

$$\boxed{\Longrightarrow \; T_{\partial w}(N_a, N_r) = O(N_a N_r^4)}$$

Complexidade do Sinal de $\partial C/\partial b_j^l$

- Dado $J = C(x, y, w, b)$, faça $\Delta_w = 0.01$ and $\Delta_b = 0.01$ sendo $T_c(N_a, Nr) = O(1)$, pois J já foi calculado
- Para cada $l : 2..L$, $j : 1..n_l$ define-se

$$-\partial C(x, y, w, b)/\partial b_j^l = \left\{ \begin{array}{ll} 1 & \text{if } (b_1 > J \wedge J > b_2) \\ -1 & \text{if } (b_1 < J \wedge J < b_2) \\ 0 & \text{otherwise} \end{array} \right\} \Longrightarrow \quad 1 + 2k_3 N_r$$

$$\text{onde } b_1 := C(x, y, w, [b|b_j^l := b_j^l - \Delta_b * b_j^l]) \Longrightarrow \quad k_3 N_r$$

$$b_2 := C(x, y, w, [b|b_j^l := b_j^l + \Delta_b * b_j^l]) \Longrightarrow \quad k_3 N_r$$

$$\Longrightarrow \quad T_{\partial b}(N_a, N_r) = \text{custo de } J + (L-1)\ n_l\ (1 + \text{custo de } b_1 \text{ e } b_2)$$

$$\Longrightarrow \quad T_{\partial b}(N_a, N_r) = k_1 + (L-1)\frac{N_r}{L}(1 + 2k_3 N_r)$$

$$\Longrightarrow \quad T_{\partial b}(N_a, N_r) = k_1 + (L-1)\frac{N_r}{L} + 2k_3 \frac{L-1}{L^2} N_r^2$$

$$\boxed{\Longrightarrow \quad T_{\partial b}(N_a, N_r) = O(N_a N_r^2)}$$

Conclusão: Backpropagation ou Sinal de Derivada?

- **Complexidade das derivadas no backpropagation:**

$$T_{\partial b}(L, N_a) = O(N_a N_r)$$

$$T_{\partial w}(L, N_a) = O(N_a N_r^2)$$

- **Complexidade da derivadas no simplificado:**

$$T_{\partial b}(N_a, N_r) = O(N_a N_r^2)$$

$$T_{\partial w}(N_a, N_r) = O(N_a N_r^4)$$

Portanto, o método recomendado é o backpropagation!!!!

Intuição sobre os Custos

- No backpropagation, realiza-se uma forward propagation e uma backward propagation em cada etapa, na qual faz-se a afinação simultânea de todos os pesos

- No simplificado, em que se calcula apenas o sinal da derivadas, deve-se calcular o custo corrente na saída da rede neural duas vezes em cada etapa na qual faz-se a afinação individual de cada peso

- Como o cálculo do custo corrente demanda uma forward propagation, o método simplificado requer um número de propagações na rede igual ao dobro do número de pesos

- Assim, vê-se claramente que o backpropagation é muito mais eficiente que o método simplificado!!

CROSS-VALIDATION

Distribuição de Dados das Amostras...

- Uma distribuição de dados 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 deve ser distinta da amostra de treinamento, e ambas devem seguir a mesma distribuição de probabilidades

- Uma boa amostra de treinamento é aquela que é representativa de toda a população de dados

- Se a distribuição dos dados da população for dinâmica, novas amostras devem ser organizadas periodicamente, e o aprendizado submetido a retreinamento

...Distribuição de Dados das Amostras

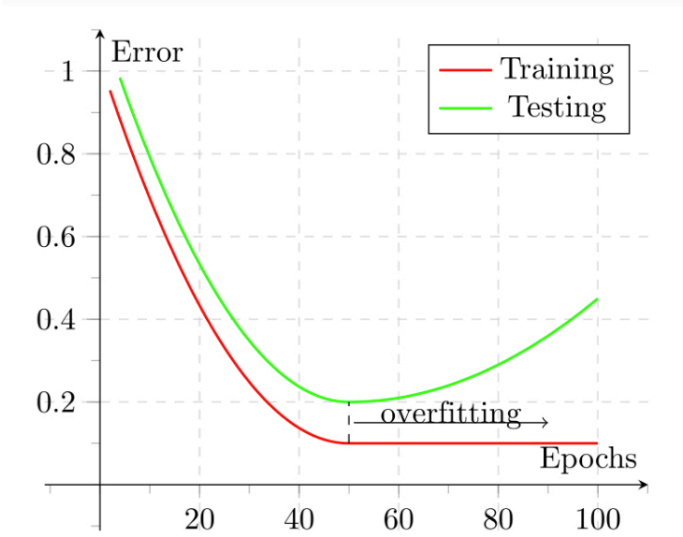
- Se o modelo usado ajusta-se igualmente bem com as amostras de treinamento e de validação, espera-se um overfitting mínimo

- Situações em que há um bom ajuste da amostra de treinamento e um mau ajuste da de validação aponta para overfitting

- A amostra de validação destina-se apenas para avaliar a capacidade de generalização do algoritmo usado

Avaliação do Aprendizado

- **Cross-Validation** é a técnica para testar o modelo de aprendizagem.
- Os procedimentos básicos são:
  1. retire da amostra dada uma subamostra para teste
  2. faça o devido treinamento com a amostra resultante
  3. avalie o treinamento usando a subamostra de teste
  4. repita esse processo com diferentes subamostras de teste

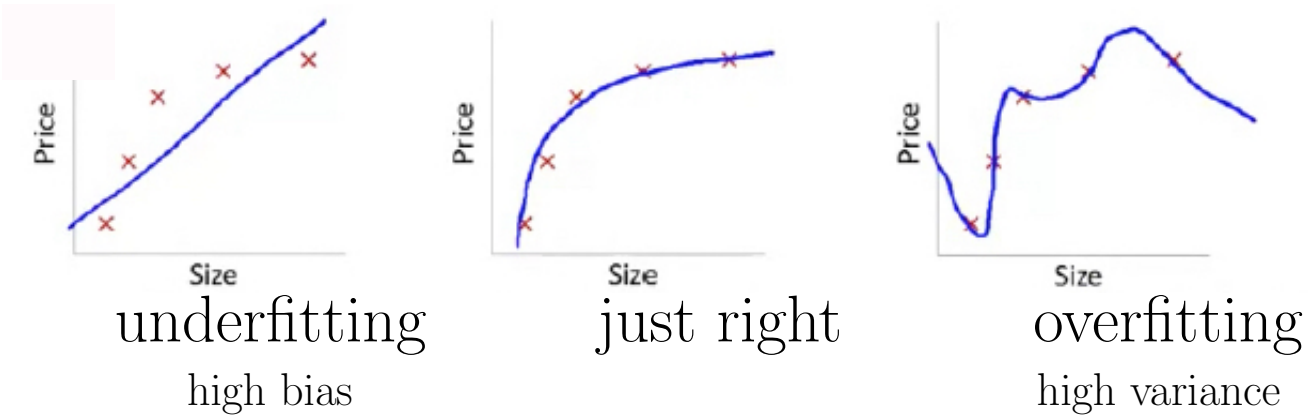


Tipos de Cross-Validation

- **Holdout:**
  remova uma subamostra de tamanho típico de 15% da amostra de treinamento e faça a cross-validation
- **K-Fold:**
  particione a amostra de treinamento em k subconjuntos de teste e aplique o holdout para cada um deles e considere o erro final apurado a média desses k testes
- **Stratified K-Fold:**
  remova subamostras de teste que tenham o mesmo perfil da amostra de treinamento e faça a cross-validation
- **Leave-P-Out:**
  retire subamostras de $p \geq 1$ pontos da amostra treino e faça o treino seguido de sua validação com essas subamostras.

Qualidade do Aprendizado

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



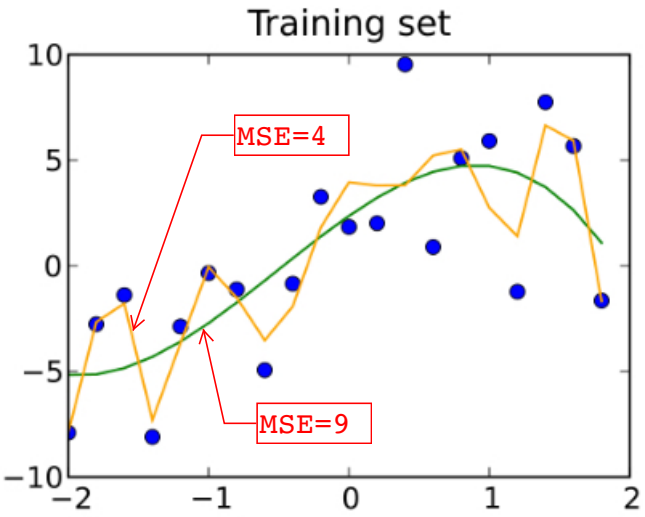
Underfitting

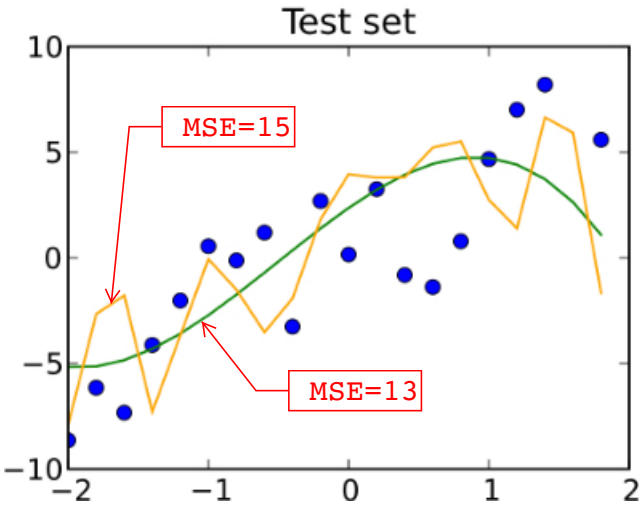
- A causa de baixa performance do algoritmo de machine learning é overfitting ou underfitting dos dados ao modelo
- Underfitting refere-se a um modelo que nem modela os dados da amostra de treinamento nem permite generalização para novos dados
- Underfitting é uma indicação clara que o algoritmo de aprendizagem não serve
- A solução é buscar algoritmos alternativos

Overfitting

- Como foi dito, a causa de baixa performance do algoritmo de machine learning é overfitting ou underfitting dos dados ao modelo
- 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 afetuam a distribuição dos dados
- O problema surge por que esses conceitos normalmente não aparecem nos novos dados e assim impactam negativamente o processo de generalização
- Overfitting são mais frequentes em modelos não-paramétricos e não-lineares flexíveis

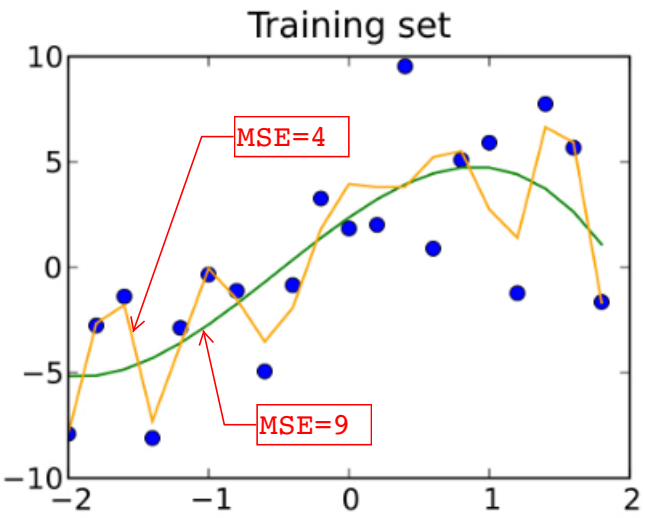
Exemplo de Overfittings...

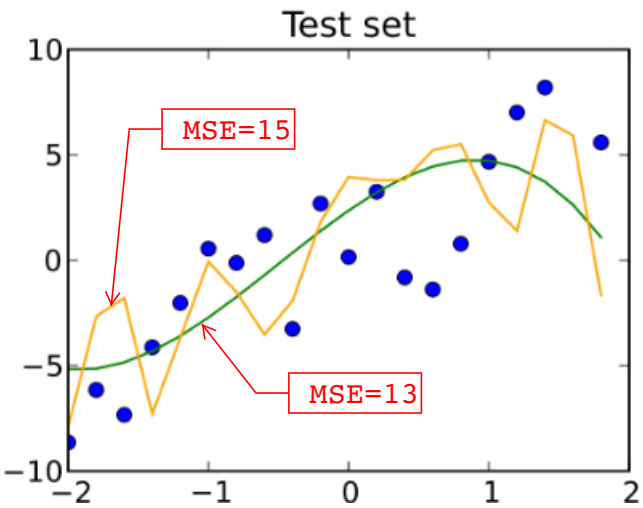




- Pontos azuis dessas figuras são obtidos de amostras distintas
- Dois aprendizados de modelos distintos estão representados pelas curvas laranja e verde em ambos os gráficos
- Suponha que o experimento mostrou que:
  - No treinamento, os MSEs são 4 (laranja) e 9 (verde)
  - Na validação, os MSEs são 15 (laranja) e 13 (verde)

...Exemplo de Overfittings





- **A curva laranja é um exemplo de overfitting severo, pois o seu MSE vai de 4 (treinamento) a 15 (validação)**
- **A curva verde tem menor grau de overfitting, pois apenas vai 9 (treinamento) para 13 (validação)**

Regularização

Redução de Overfitting

- Overfitting é o maior problema em rede neurais com um número elevado de pesos e biases

- 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
  - aplicar heurísticas para escolha de bons hiperparâmetros, como taxa de aprendizagem, etc

- Ou então aplicar Técnicas de Regularização

Técnicas de Regularização

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

- Regularização permite reduzir overfitting sem afetar o tamanho da rede ou da amostra

- As principais técnicas são:
  - Regularização L1
  - Regularização L2
  - Dropout

Regularização L2

- Adicionar uma parcela à função de custo C_0:

$$C = C_0 + \frac{\lambda}{2N_a} \sum_w w^2$$

- Pequeno $\lambda > 0 \Rightarrow$ o custo mínimo depende majoritariamente de C_0
- Grande $\lambda \Rightarrow$ 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
- Regularização é um compromisso entre o uso de pesos de pequenos valores, que geram melhores resultados, mas requerem elevado processamento, e a qualidade de minimização da função de custo

Regularização L1

- Adicionar uma parcela à função de custo C_0:

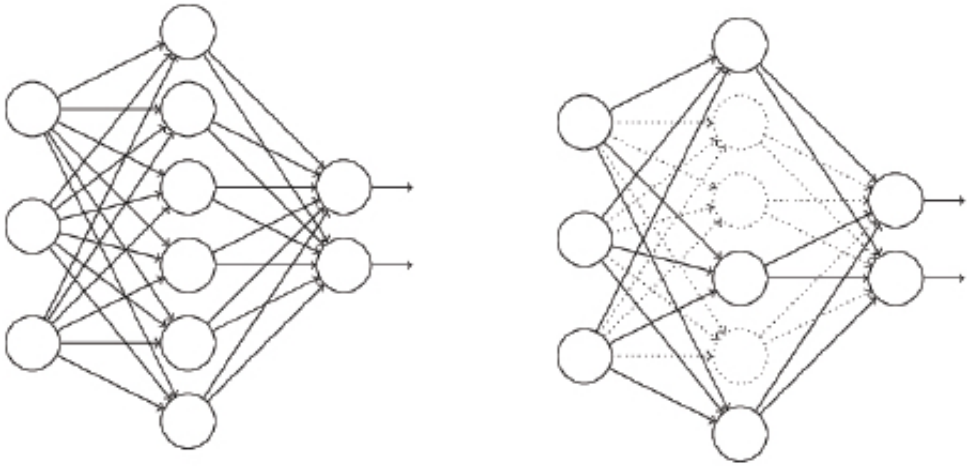
$$C = C_0 + \frac{\lambda}{N_a} \sum_w |w|$$

- λ tem o mesmo que papel de definir preferências que em L2

- Adicionalmente, observa-se que:
  - quando $\Delta w > 1$, tem-se que $|\Delta w| < \Delta w^2$
    $\Rightarrow$ afinação com L2 encolhem os pesos mais rapidamente que L1
  - quando $\Delta w < 1$, tem-se que $|\Delta w| > \Delta w^2$
    $\Rightarrow$ afinação com L1 encolhem os pesos mais rapidamente que L2

Dropout

- Apagar aleatória e temporariamente a metade dos neurônios de camadas ocultas
- Fazer as forward e backward propagations
- Atualizar os pesos e biases com os gradientes computados
- Restaurar os neurônios apagados e reavaliar o custo da rede
- Repetir o processo acima até obter pesos e bias satisfatórios



- Dropout assegura funcionamento mesmo com falta de dados!!!

REDES NEURAIS CONVOLUCIONAIS

Motivação

- As redes neurais apresentadas anteriormente são fortemente conectadas, isto, é são densas
- Em redes neurais densas, as saídas de todos os neurônios de uma camada dependem das saídas de TODOS os neurônios da camada anterior
- Nesse tipo de rede neural, o número de pesos e biases tendem a ser muito altos, o que aumenta substancialmente o custo de treinamento
- A redução do número desses pesos, agrupando neurônios, reduz custos de treinamento
- Certas redes neurais, como as para visão computacional ou de reconhecimento de imagens, permitem redução do número de parâmetros se perda de qualidade

Redes Convolucionais

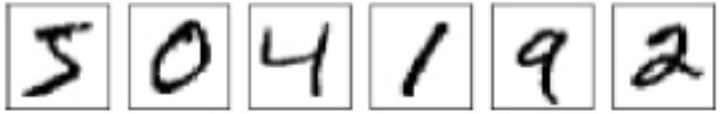
- Redes convolucionais são redes neurais nas quais camadas internas não são densas, i.e., seus neurônios não dependem de TODOS os neurônios da camada anterior

- Essas camadas internas são construídas por um método inspirado na operação matemática de convolução

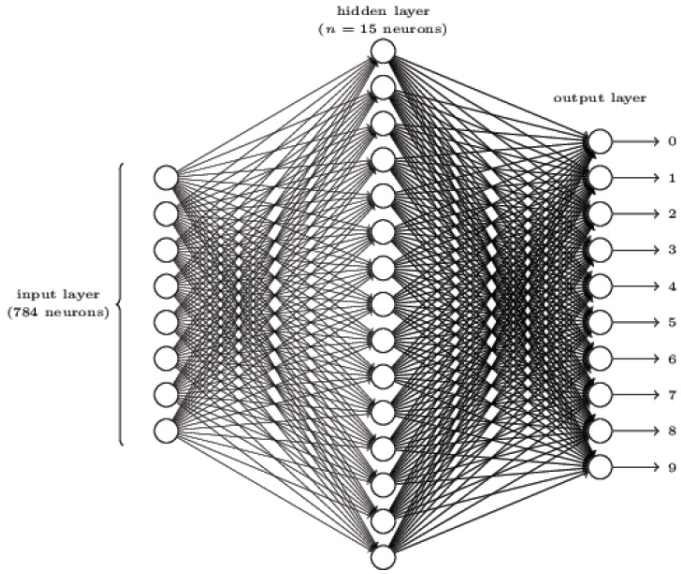
- Redes neurais convolucionais são baseadas nas seguintes ideias:
  - entrada na forma matricial em vez de vetorial
  - geração de camada interna via operação de filtragem, por meio de cálculo de features
  - geração de outras camadas internas via operação de pooling
  - camada de saída densa, no sentido em que as saídas de seus neurônios dependem de todos os neurônios da camada anterior

Camada de Entrada de uma Rede Neural Densa

- Um sistema de reconhecimento de imagens de dígitos manuscritos, onde cada dígito é apresentado como uma imagem de 28 X 28 pixels, e.g.,

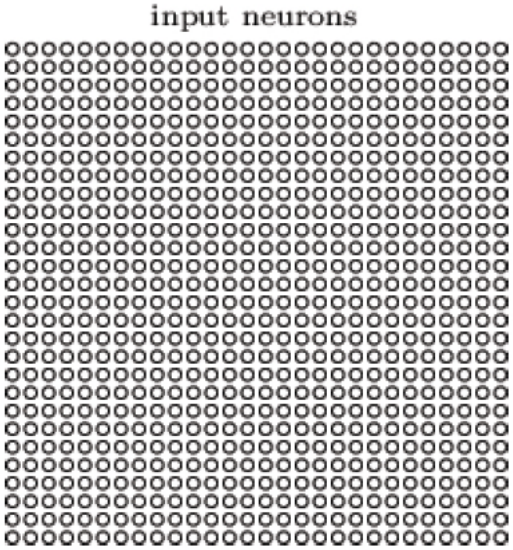


pode ser implementado por uma Rede Neural Densa, cuja entrada é um vetor de 28 X 28 pixels, representando a imagem de um dígito



Camada de Entrada da Rede Neural Convolutional

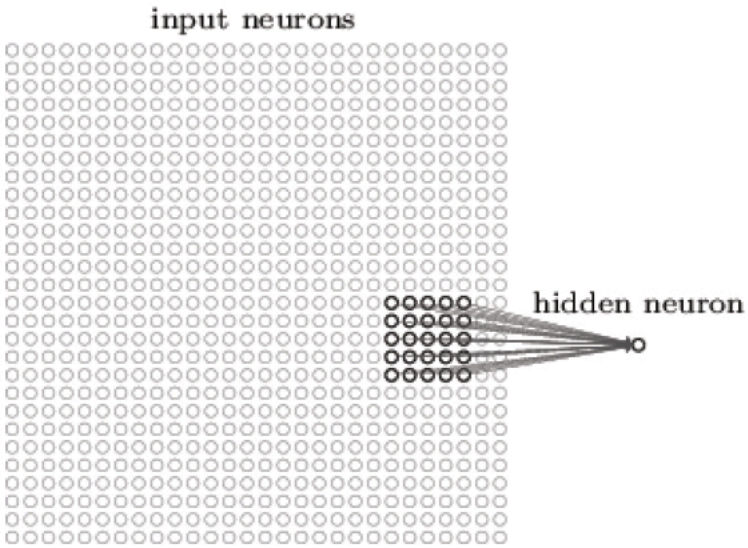
- Na rede convoluconal substitui-se o VETOR de neurônios de entrada, no exemplo citado de 28 X 28 elementos, por uma MATRIZ 28 X 28 elementos:



- Dessa forma, a posição dos pixels na imagem pode ser usada como informação relevante
- Assim, o uso da matrix em vez do vetor de entraa, permitem que a rede processe de forma eficiente informações espaciais

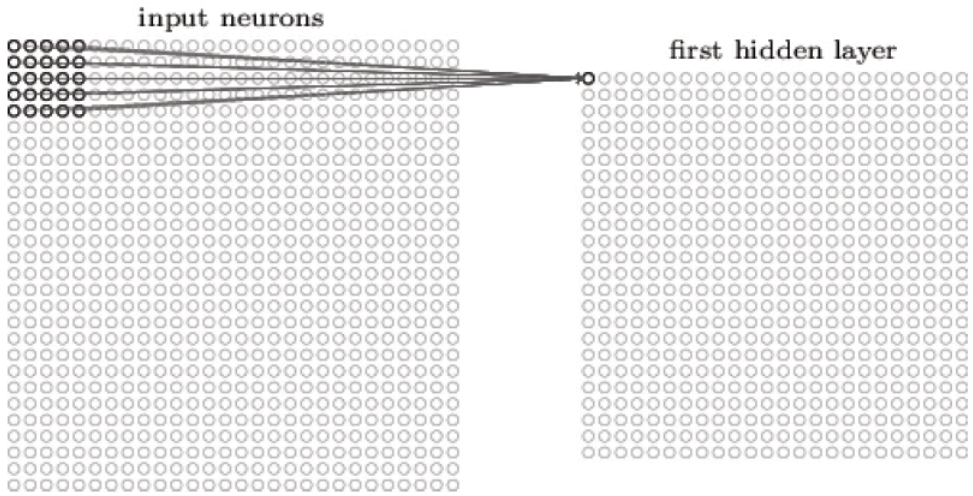
A Filtragem

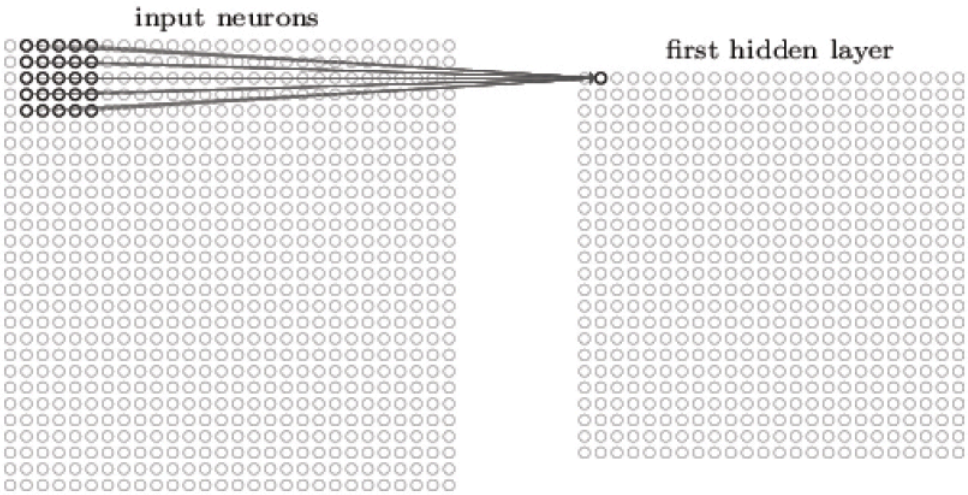
- Em Rede Neural Convolucional (CNN), uma função de convolução, usando filtros de diferentes tamanhos, é aplicada à imagem dada para gerar as entradas dos neurônios da primeira camada interna
- A matriz de filtro ou kernel é uma matriz de pesos da RN
- O tamanho da matriz de filtro pode variar de uma etapa a outra



Ideia Básica da Convolução...

- Kernels são uma matriz quadrada que define o foco de análise da matriz de entrada
- A janela definida pelo kernel ou filtro deve-se mover sobre a imagem para gerar a matriz convoluta





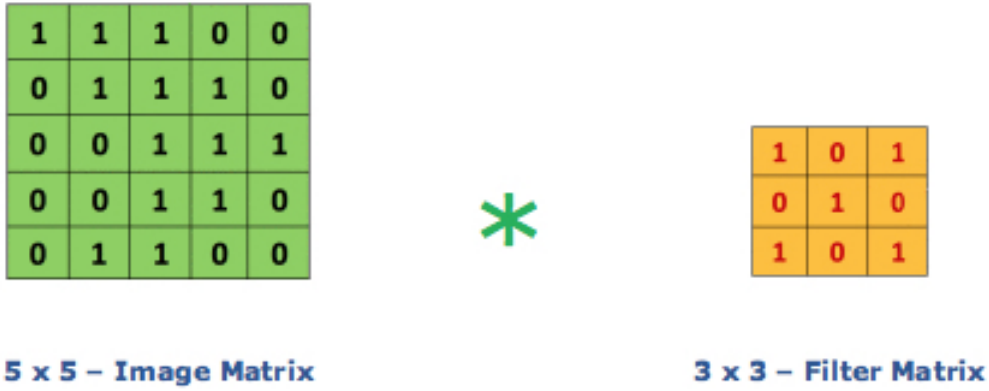
- Stride é a distância do passo do filtro (tipicamente stride = 1)
- A região da imagem selecionada pelo filtro denomina-se **local receptive field**

...Ideia Básica da Convolução

- Pode-se aumentar o tamanho do kernel quando deseja-se focar em partes maiores da imagem
- A matriz de saída, denominada convoluted feature, é menor que a de entrada
- Cada item da feature matriz corresponde a uma seção da imagem.
- O objetivo do processo de convolução é filtragem

Convolução de uma Imagem...

- **Suponha a uma matriz de entrada 5X5 e um filtro 3X3:**

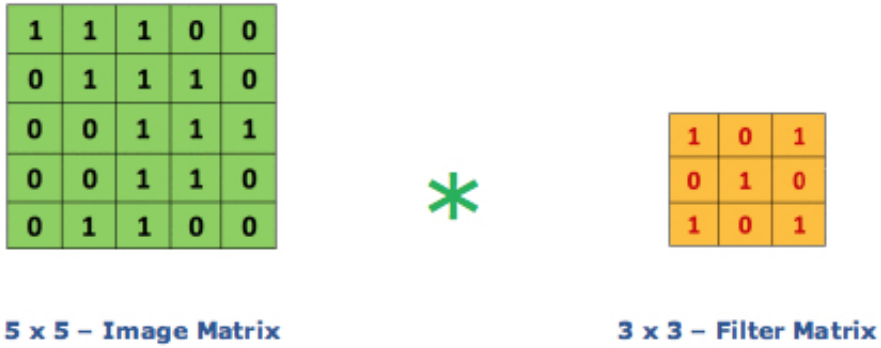


The image and kernel matrices

- **Sendo $*$ a operação de filtragem ou convolução**

...Convolução de uma Imagem...

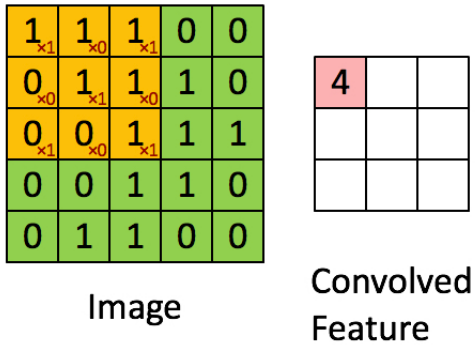
- **A aplicação inicial de filtragem**



The image and kernel matrices

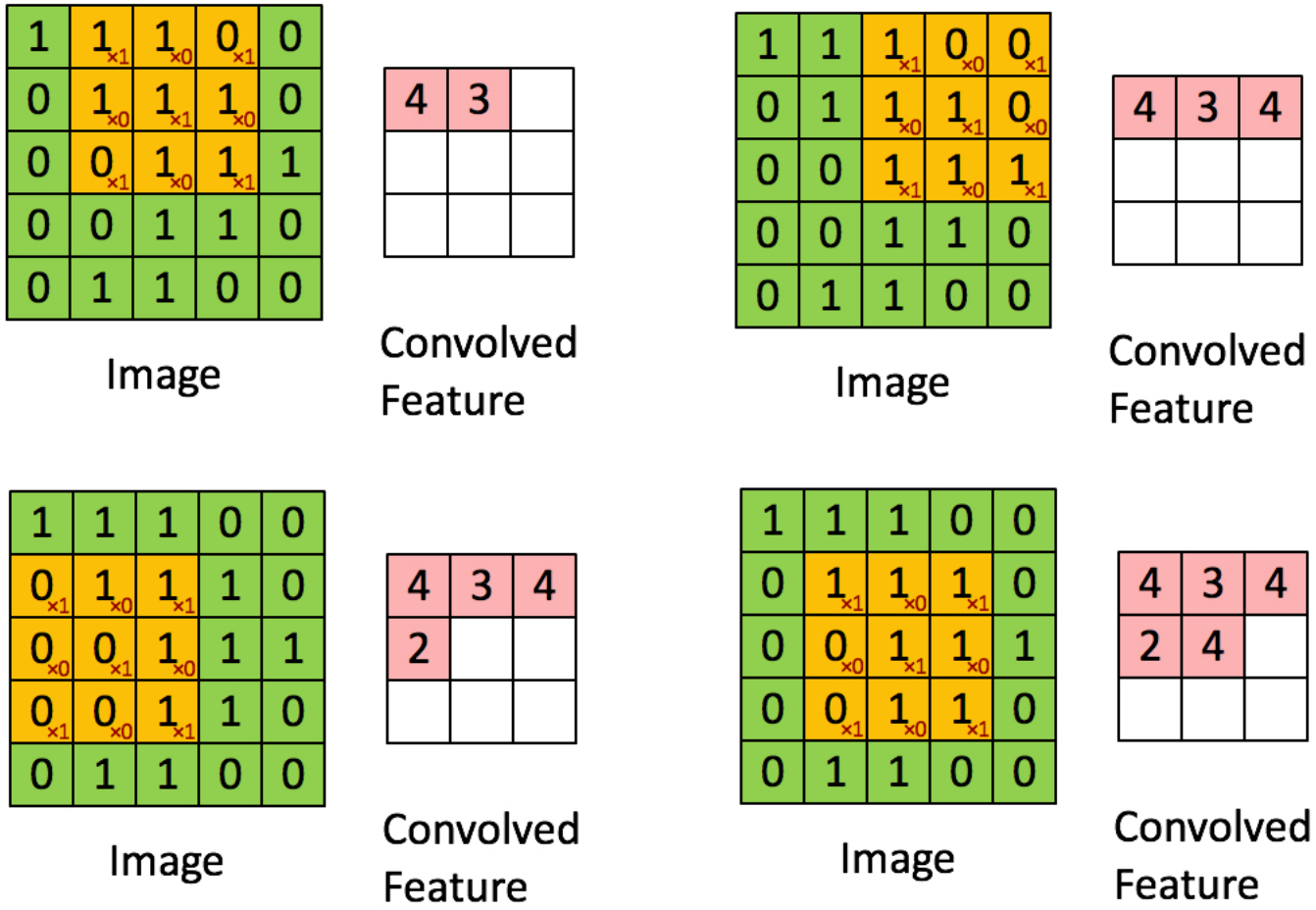
- **produz o primeiro elemento da convoluted feature:**

$$4 = 1 * 1 + 1 * 0 + 1 * 1 + 0 * 0 + 1 * 1 + 1 * 0 + 0 * 1 + 0 * 0 + 1 * 1$$



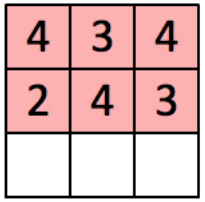
...Convolução de uma Imagem...

- **Desloca-se o filtro e o reaplica até preencher a convoluted feature**

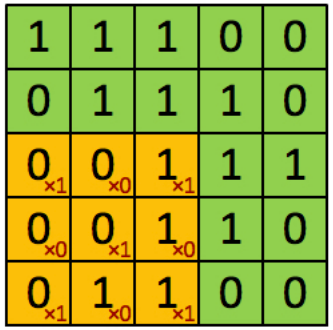


...Convolução de uma Imagem

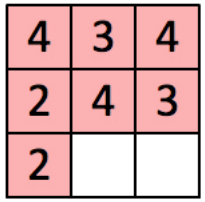
Image



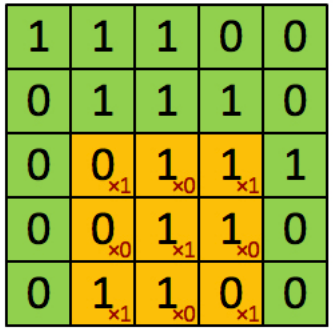
Convolved Feature



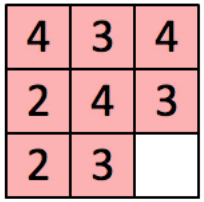
Image



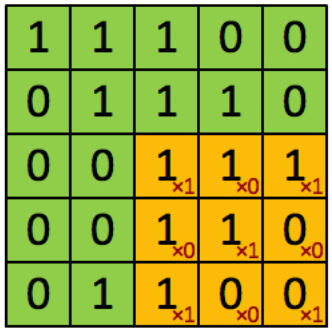
Convolved Feature



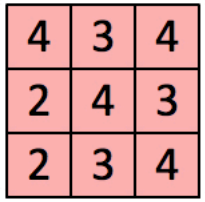
Image



Convolved Feature



Image



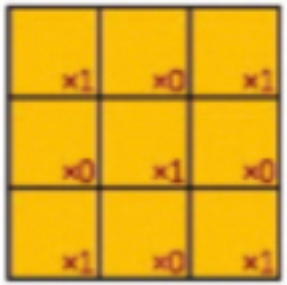
Convolved Feature

Tamanho da Convoluted Feature

- Dada uma matriz de entrada m X m, um filtro k X k e um stride $= p$
- A convoluted feature tem dimensão r X r, onde $r = m - (k + p - 1) + 1$
- Exemplos:
  - entrada 28X28, filtro 5X5 e passo = 1 produz uma convoluted features de 24x24 neurônios
  - entrada 5X5, filtro 3X3 e passo = 1 produz uma convoluted features de 3X3 neurônios

Padding

- **Para obter, na operação de convolução, uma imagem de saída do mesmo tamanho da entrada, deve-se acrescentar, perimetricamente, linhas e colunas com zeros à matriz de entrada**



0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	1	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

2	2	3	1	1
1	4	3	4	1
2	2	4	3	3
1	2	3	4	1
1	2	3	1	1

Output size:$(H - K$

Compartilhamento de Pesos e Biases

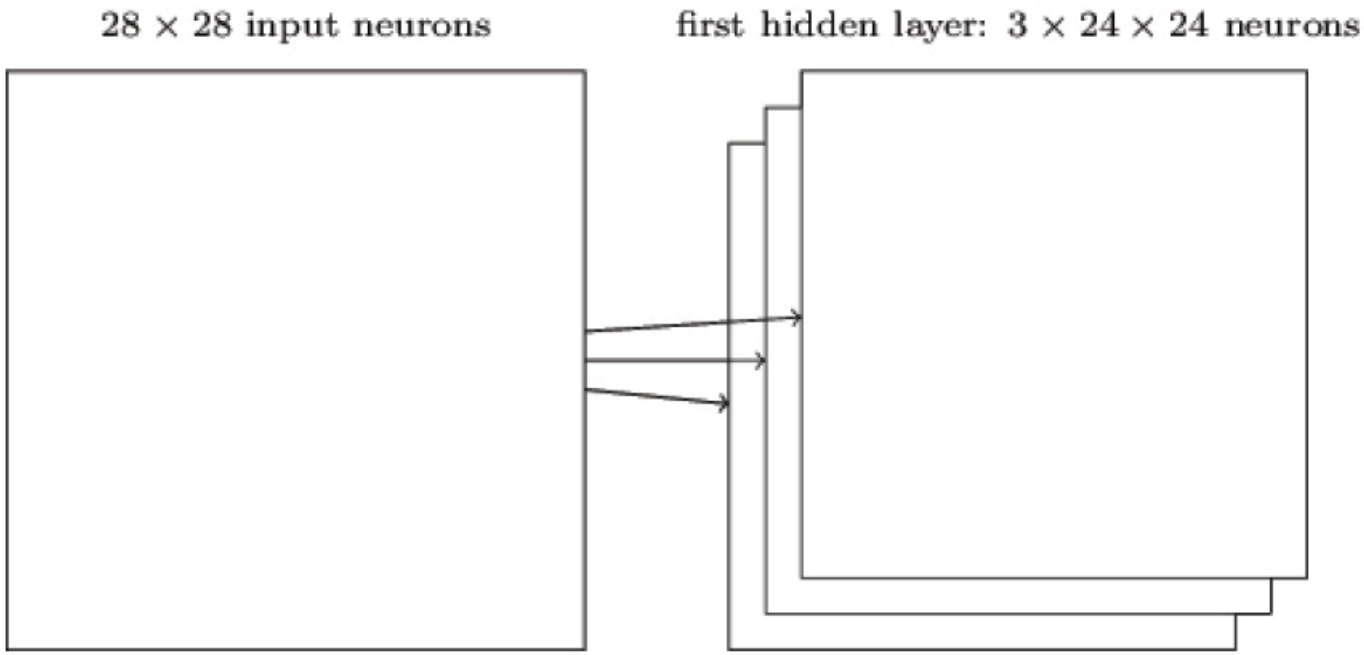
- Supondo um filtro 5X5, cada neurônio da primeira camada oculta tem um bias e 5X5 pesos que o conectam ao campos locais receptivos da entrada
- Especificamente, o neurônio na posição (j, k) da convoluted feature, tem o seguinte valor:

$$a_{i,j} = \sigma\left(b + \sum_{i=0}^{4} \sum_{m=0}^{4} w_{l,m}\ a_{j+l,k+m}\right)$$

- Para detectar outras features da mesma entrada, definem-se outros filtros, que irão produzir novas convoluted features
- A equação acima pode ser re-escrita como $a^1 = \sigma(b + w * a^0)$, onde $*$ é o operador matemático de convolução, daí o nome de rede neuronais convolucionais

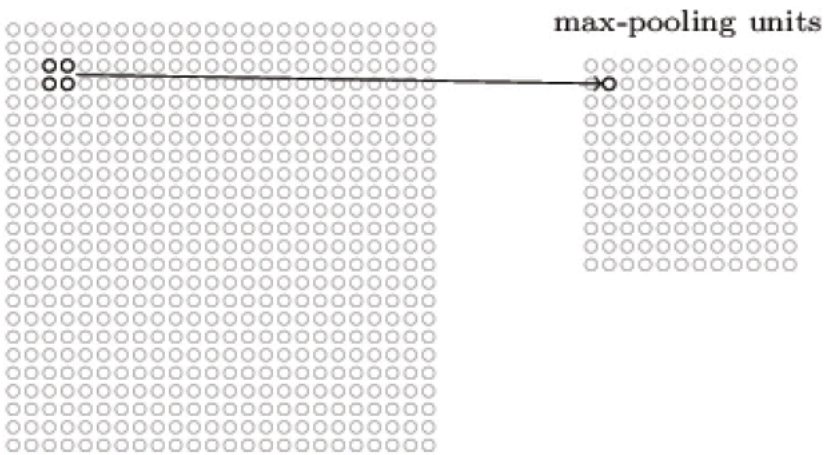
Compartilhamento de Pesos e Biases

- **Se forem usados três filtros 5X5 para a imagem 28X28 apresentada, a primeira camada oculta terá $3 * 24 * 24$ neurônios**



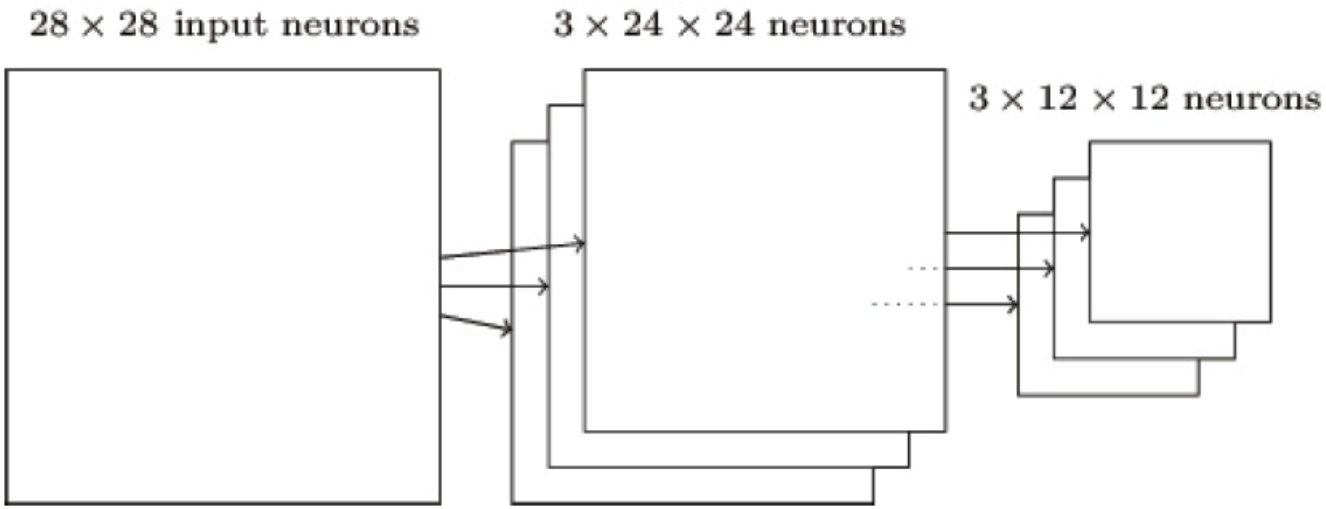
Geração de Mais Camadas - Pooling...

- Operação pooling simplifica as informações da camada produzida pela operação de convolução
- A operação toma uma feature convoluted e prepara uma condensed feature
- Por exemplo, pode-se mapear uma sub-região de neurônios 2 X 2 em uma única unidade com o peso máximo nesta sub-região



...Geração de Mais Camadas - Pooling

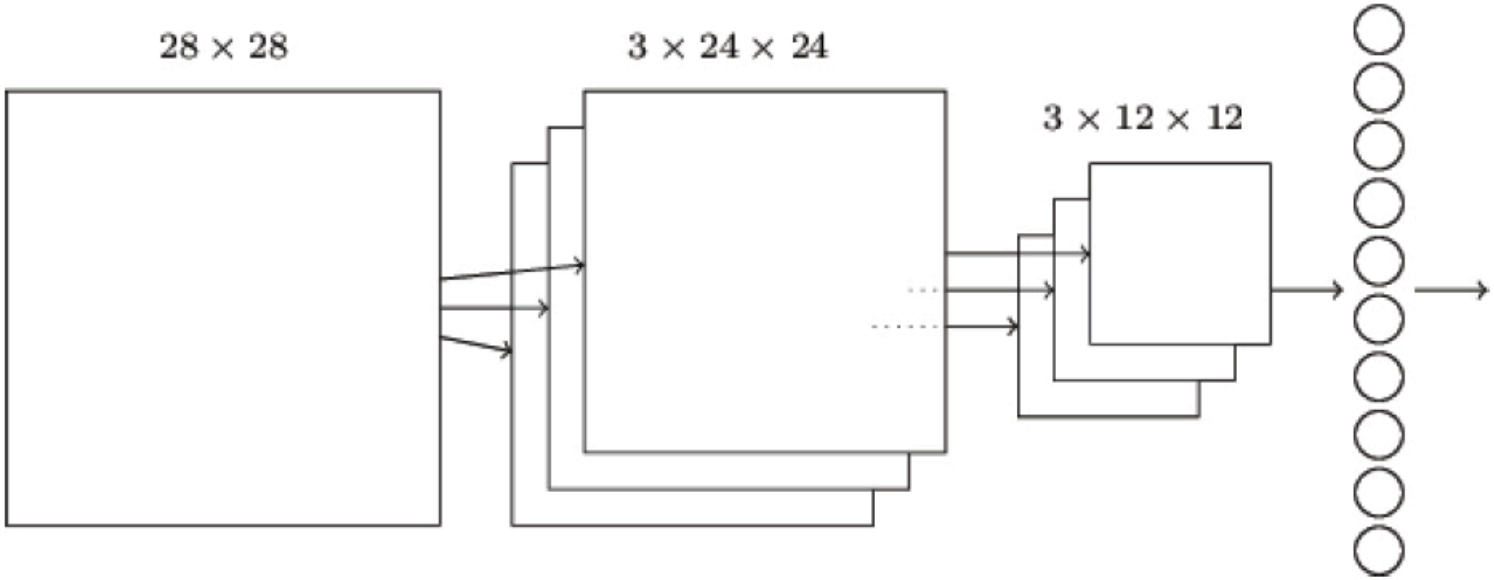
- A aplicação de max-pooling a cada uma das convoluted features já calculadas produz:



- que formam a segunda camada oculta com $3 * 12 * 12$ neurônios

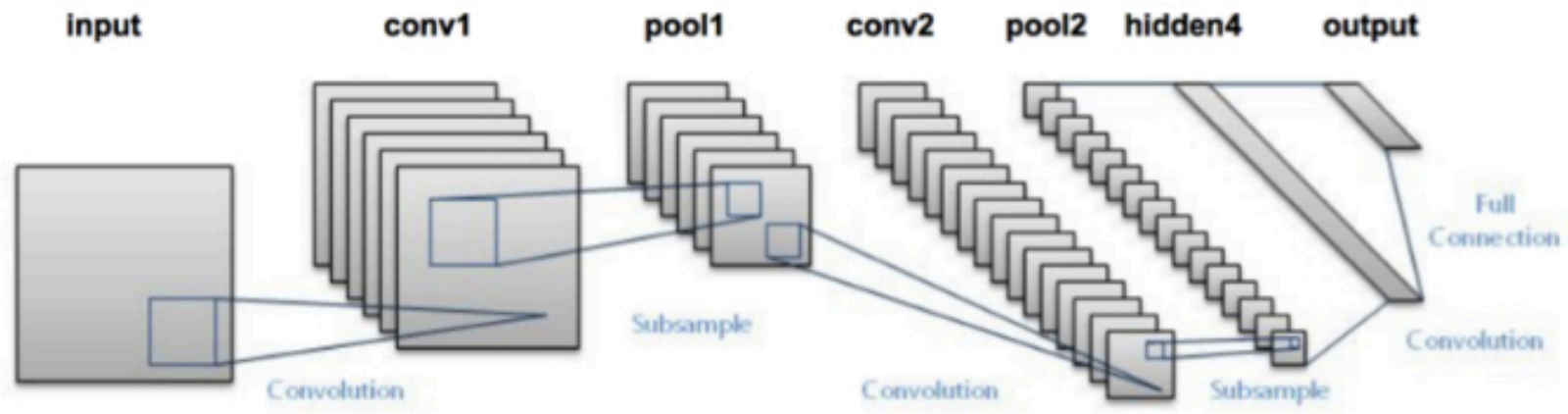
Camada de Saída da CNN

- Os neurônios da camada de saída são densamente conectados a todos os neurônios da camada max-pooled imediatamente anterior



Arquitetura Final da CNN

- **No caso geral, as operação de convolução de features, com vários filtros, e condensação max-pooled podem ser repetidas, gerando arquiteturas mais complexas**

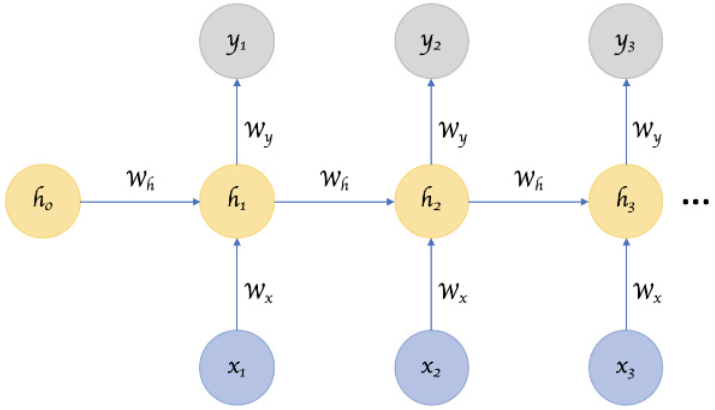


LeNet Architecture

RECURRENT NETWORKS

Conceito de Recurrent Network

- Em redes neurais do tipo **feedforward**, a ativação de um neurônio depende apenas da ativação dos neurônios da camada anterior, não há loops de dependência
- Nas redes neurais recorrentes, a ativação de um neurônio é também função de suas ativações anteriores, i.e., pode haver feedback loops
- Isso modela situações como a de que opiniões sobre a qualidade de um filme são alteradas quando ele ganhar um Oscar



FERRAMENTAS DE MACHINE LEARNING

Principais Ferramentas

- tensorFlow
- pyTorch
- keras
- caffe

CONCLUSÕES

Conclusões

- Deep Learning trouxe uma enorme contribuição para a área de Inteligência Artificial
- Entretanto, ainda há um custo muito alto para isso: para funcionar adequadamente requer-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

FIM