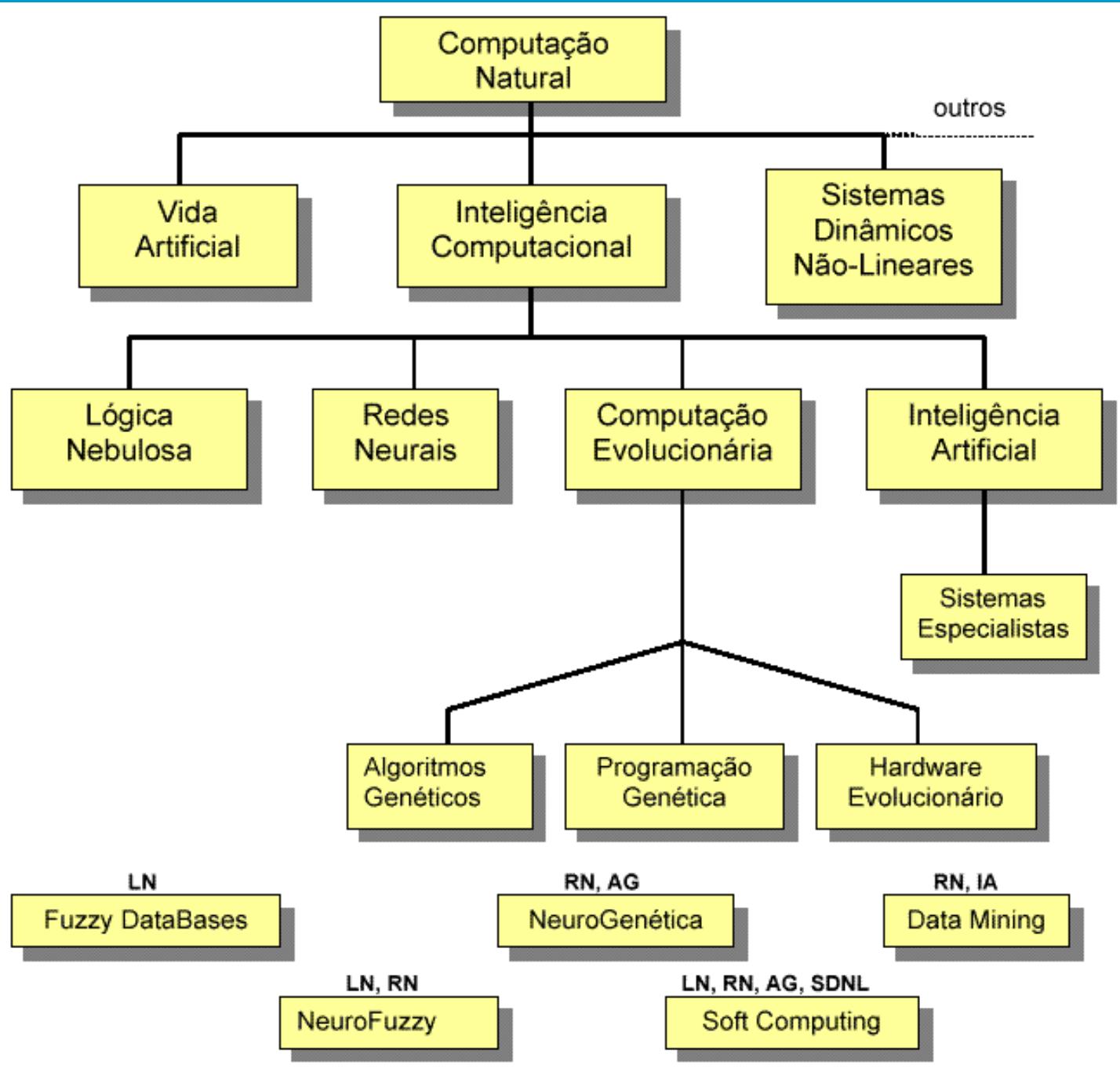


Redes Neurais Profundas

Aula 1

Prof. Anderson Soares
www.inf.ufg.br/~anderson
Instituto de Informática
Universidade Federal de Goiás



Sumário

- Introdução
- Inspiração biológica
- Histórico
- Neurônio artificial
- Treinamento do neurônio
- Implementações
- Recomendações finais

Introdução

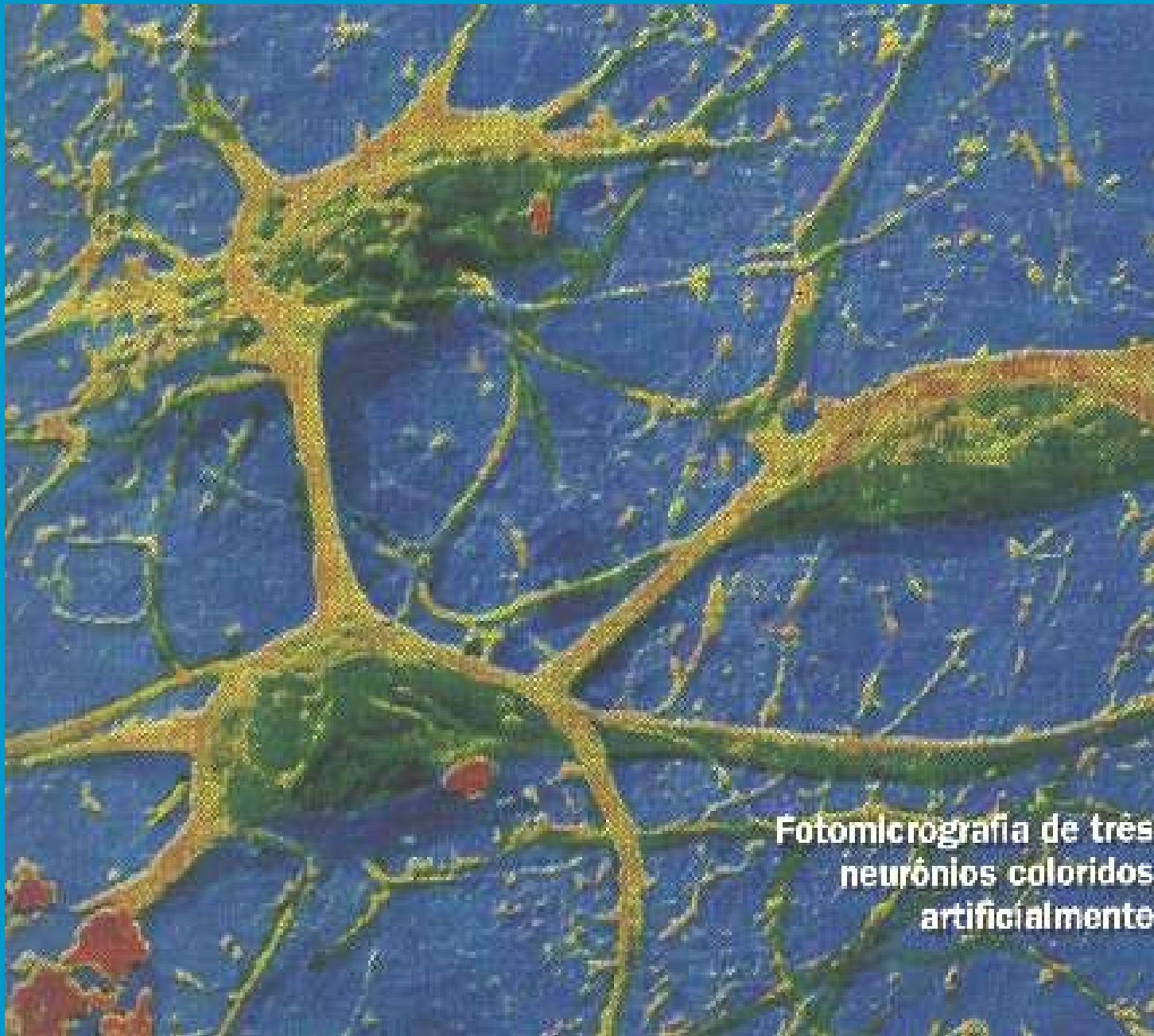
Redes Neurais Artificiais (RNA)

Robert Hecht-Nielsen, definiu rede neural como:

"...um sistema computacional feito a por uma série de elementos de processamentos simples, altamente conectados, cuja informação é processada a partir de entradas de informação externas. "

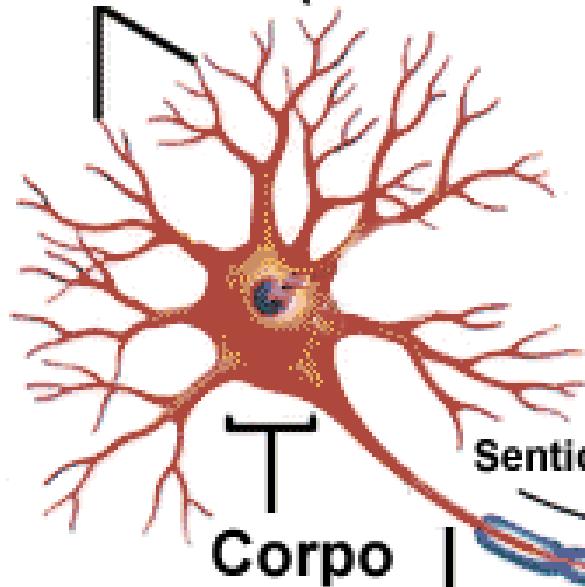
Inspiração biológica

- A unidade básica do cérebro é o neurônio.
- Aproximadamente 86 bilhões
- Aproximadamente 10^{14} — 10^{15} de sinapses.

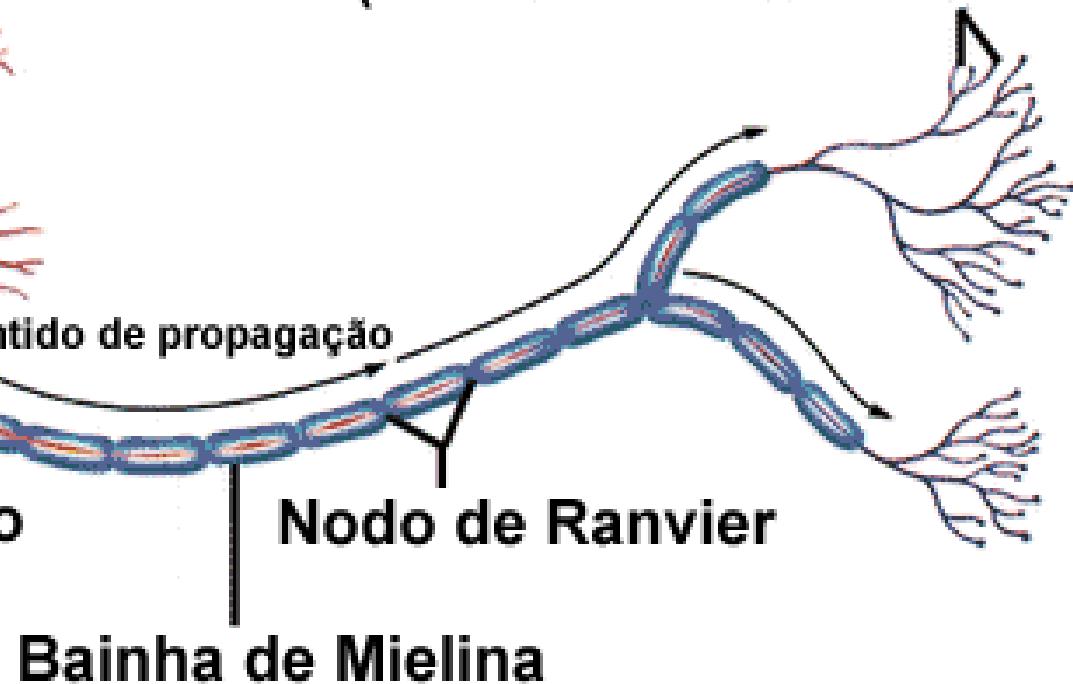


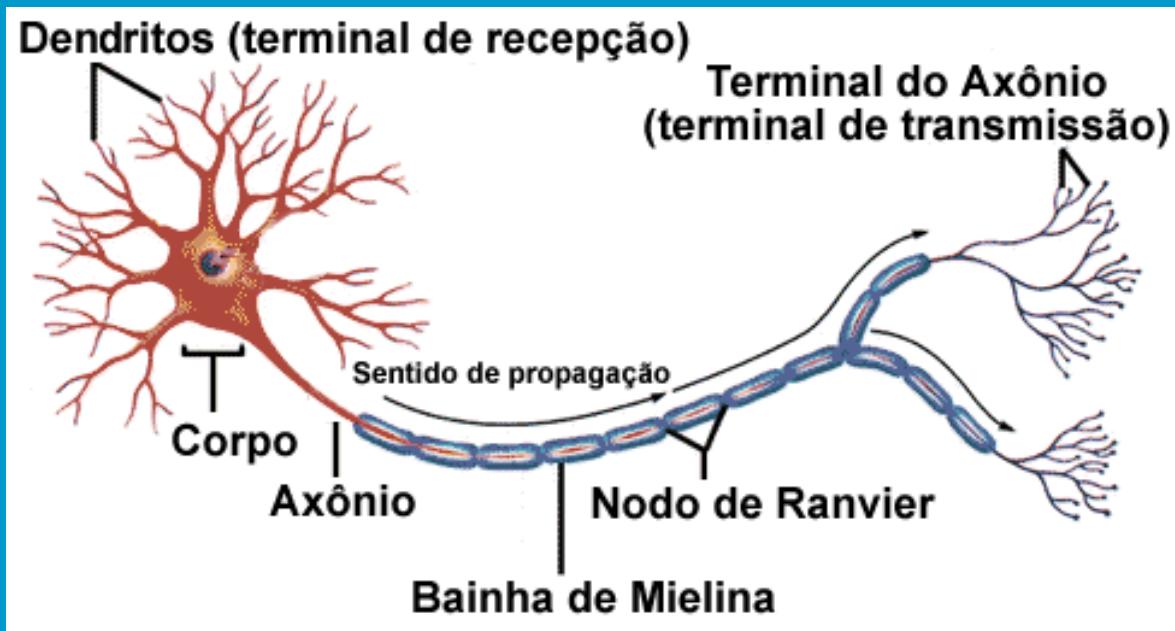
Fotomicrografia de três
neurônios coloridos
artificialmente

Dendritos (terminal de recepção)



Terminal do Axônio (terminal de transmissão)





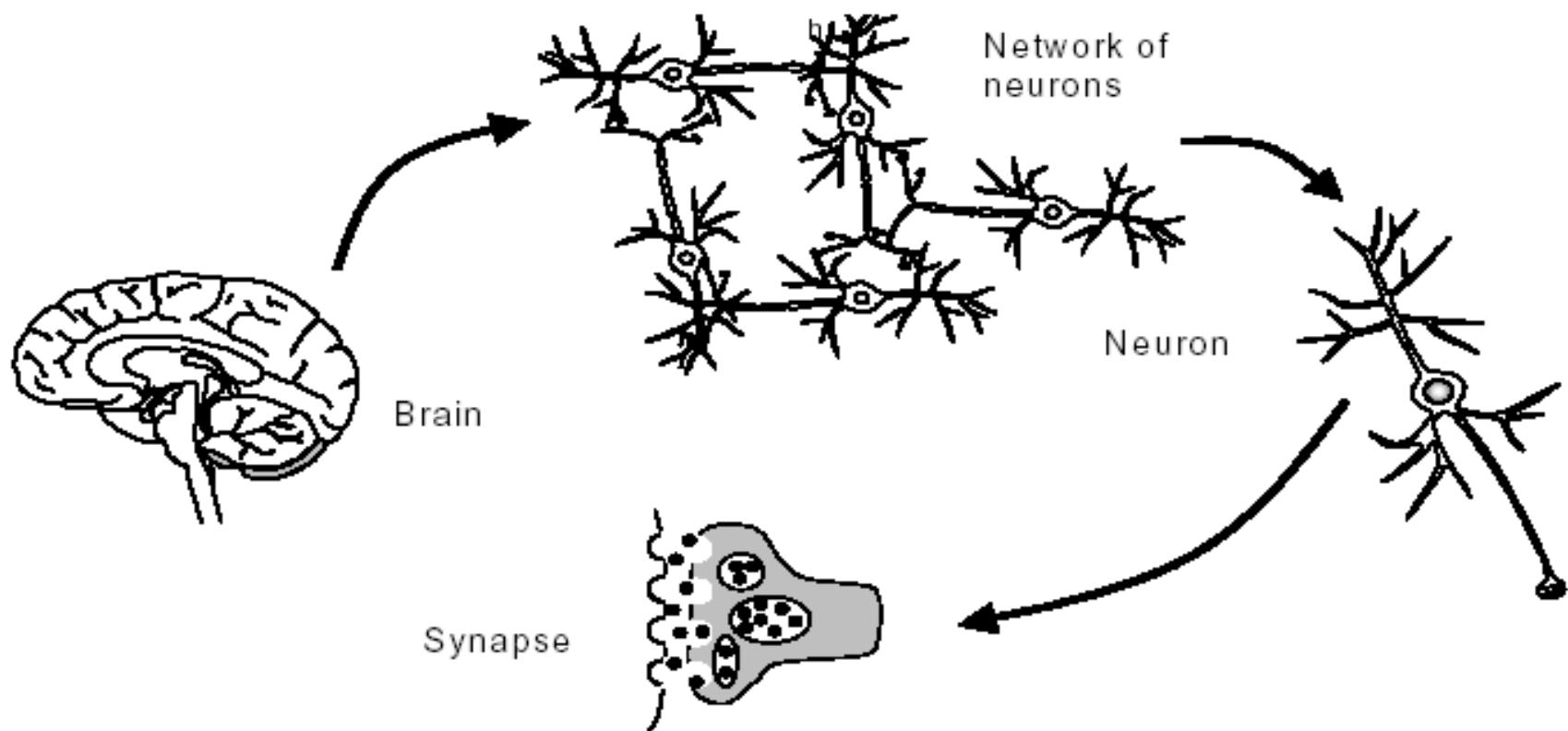
Os dentritos: recebem os estímulos transmitidos pelos outros neurônios;

O corpo de neurônio, também chamado de soma: responsável por coletar e combinar informações vindas de outros neurônios;

Axônio: constituído de uma fibra tubular que pode alcançar até alguns metros, e é responsável por transmitir os estímulos para outras células.

- Em média, cada neurônio forma entre mil e dez mil sinapses.
- O cérebro humano possui cerca de 10^{11} neurônios, e o número de sinapses é de mais de 10^{14} , possibilitando a formação de redes muito complexas.

Cérebro humano



Quadro comparativo entre cérebro e o computador

Parâmetro	Cérebro	Computador
Material	Orgânico	Metal e plástico
Velocidade	Milisegundos	Nanosegundos
Tipo de Processamento	Paralelo	Seqüencial
Armazenamento	Adaptativo	Estático
Controle de Processos	Distribuído	Centralizado
Número de elementos processados	10^{11} à 10^{14}	10^5 à 10^6
Eficiência energética	10^{-16} J/op./seg.	10^{-6} J/op./seg
Ligações entre elementos processados	10.000	<10

Redes Neurais Artificiais (RNA)

Definições:

1. Técnica inspirada no funcionamento do cérebro, onde neurônios artificiais, conectados em rede, são capazes de aprender e de generalizar.
2. Técnica de aproximação de funções por regressão não linear.

Capacidade de Generalização

Isso significa que se a rede aprende a lidar com um certo problema, e lhe é apresentado um similar, mas não exatamente o mesmo, ela tende a reconhecer esse novo problema, oferecendo a mesma solução.

Aproximador de funções

- A característica mais significante de redes neurais está em sua habilidade de aproximar qualquer função contínua ou não contínua com um grau de correção desejado. Esta habilidade das redes neurais as tem tornado útil para modelar sistemas não lineares.

Mapeamento Entrada-Saída para Aproximação de Funções

- *Objetivo da Aprendizagem:* descobrir a função $f(\bullet)$ dado um número finito (desejável pequeno) de pares entrada-saída (x, d) .

Teorema da Projeção Linear

Objetivo da aproximação de funções: em uma área compacta S do espaço de entrada descrever uma função $f(\mathbf{x})$, pela combinação de funções $\varphi_i(\mathbf{x})$ mais simples:

$$\hat{f}(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^N w_i \varphi_i(\mathbf{x})$$

onde w_i são elementos reais do vetor $\mathbf{w} = [w_1, \dots, w_N]$ tais que

$$|f(\mathbf{x}) - \hat{f}(\mathbf{x}, \mathbf{w})| < \varepsilon$$

e ε pode ser arbitrariamente pequeno.

A função $\hat{f}(\mathbf{x}, \mathbf{w})$ é chamada de *aproximante* e as funções $\{\varphi_i(\mathbf{x})\}$ são chamadas de *funções elementares*.

Redes Neurais Artificiais (RNA)

Devido à sua estrutura, as Redes Neurais Artificiais são bastante efetivas no aprendizado de padrões a partir de dados:

- não-lineares,
- incompletos,
- com ruído e até
- compostos de exemplos contraditórios.

Histórico

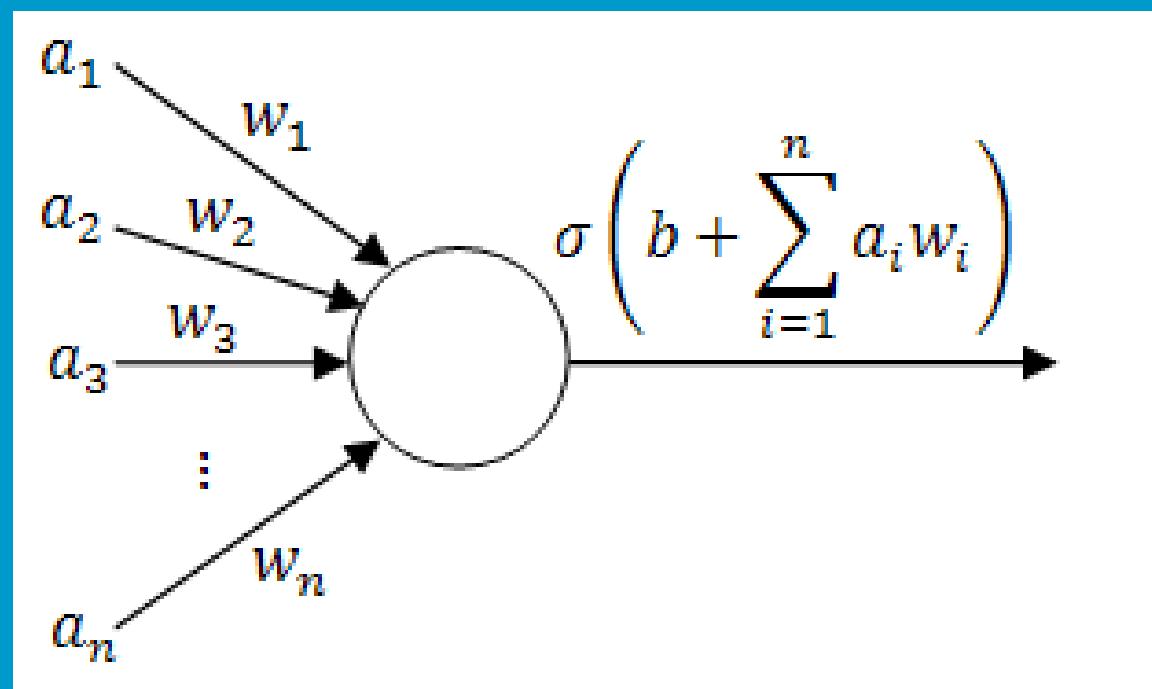


Histórico (1943)

O neurofisiologista McCulloch e matemático Walter Pitts (1943), cujo trabalho fazia uma analogia entre células vivas e o processo eletrônico, simulando o comportamento do neurônio natural, onde o neurônio possuía apenas uma saída, que era uma função do valor de suas diversas entradas.

O neurônio de McCulloch e Pitts

- Consiste basicamente de um neurônio que executa uma função lógica.
- Os nós produzem somente resultados binários e as conexões transmitem exclusivamente zeros e uns.
- As redes são compostas de conexões sem peso, de tipos excitatórios e inibitórios.
- Cada unidade é caracterizada por um certo limiar (*threshold*).



Histórico (1949)

O psicólogo Donald Hebb, demonstrou que a capacidade da aprendizagem em redes neurais biológicas vem da alteração da eficiência sináptica, isto é, a conexão somente é reforçada se tanto as células pré-sinápticas quanto as pós-sinápticas estiverem excitadas;

Hebb foi o primeiro a propor uma lei de aprendizagem específica para as sinapses dos neurônios.

Histórico (1951)

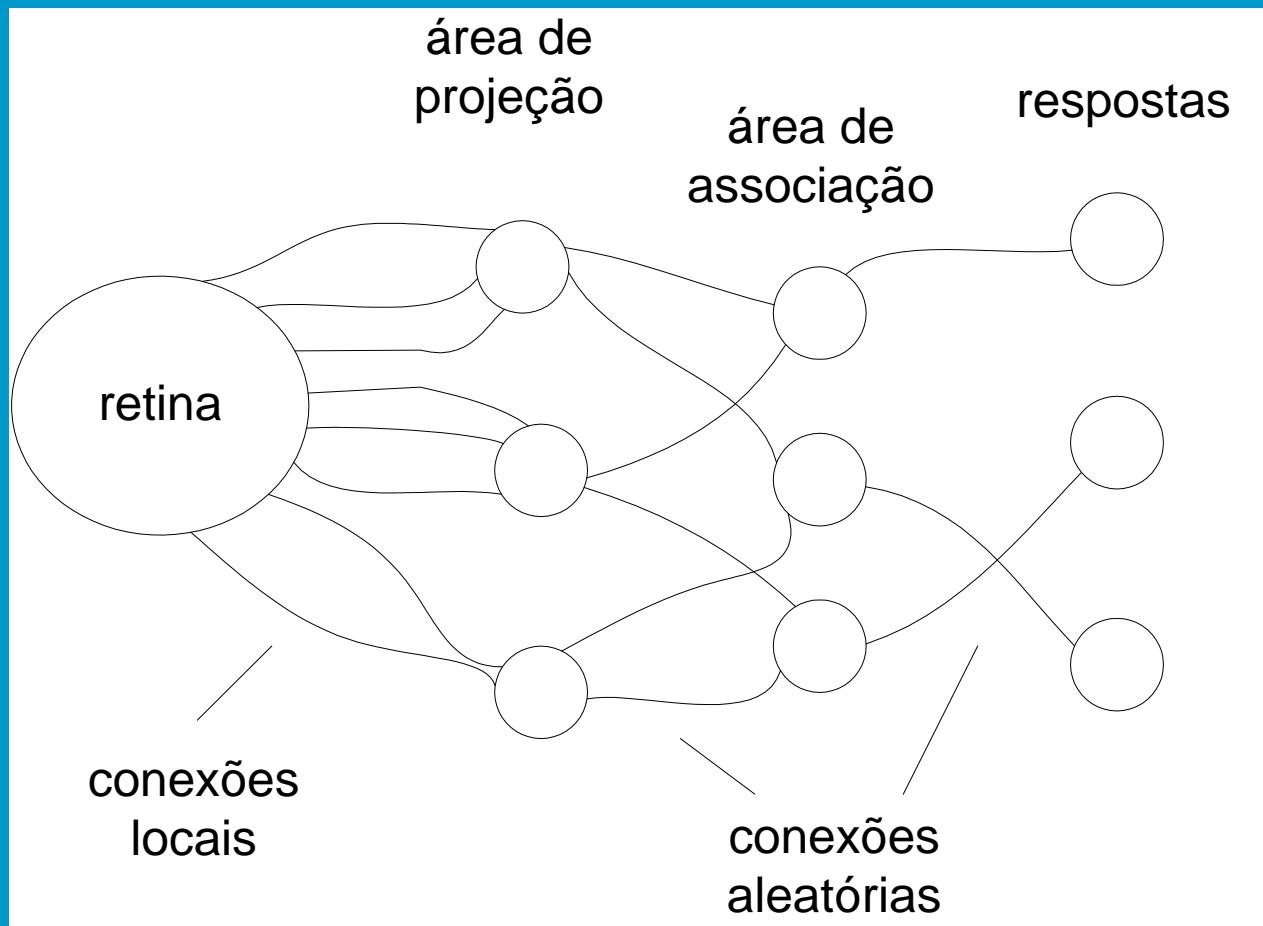
- Construção do primeiro neuro computador, denominado Snark, por Marvin Minsky. O Snark operava ajustando seus pesos automaticamente.

Histórico (1958)

Rosemblatt (1958) mostrou em seu livro (Principles of Neurodynamics) o modelo dos "Perceptrons".

Nele, os neurônios (Perceptrons) eram organizados em camada de entrada e saída, onde os pesos das conexões eram adaptados a fim de se atingir a eficiência sináptica usada no reconhecimento de caracteres.

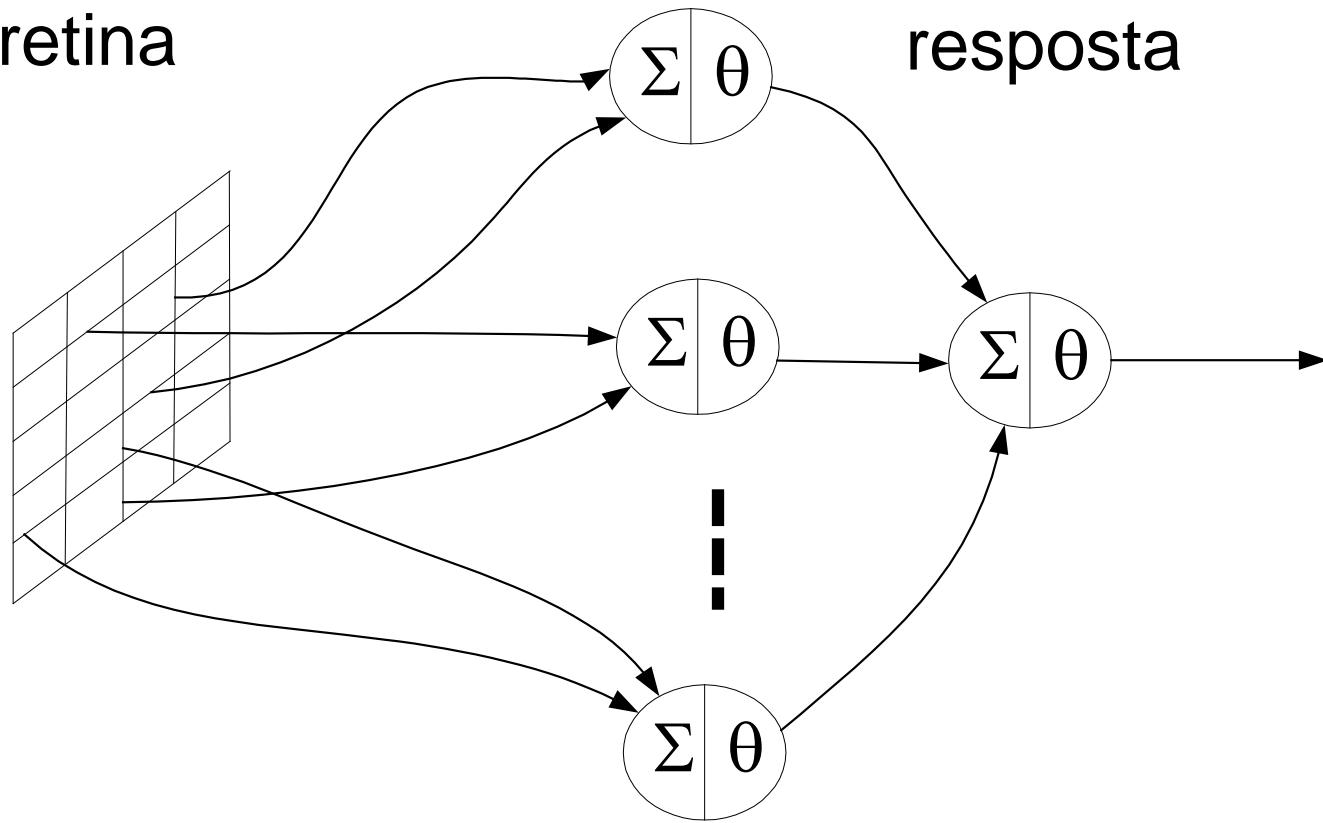
Perceptron Clássico – Rosenblatt (1958)



associação

retina

resposta



Histórico (1960)

Em 1960 surgiu a rede ADALINE (ADAptive LInear NEtwork) e o MADALINE (Many ADALINE), proposto por Widrow e Hoff.

O ADALINE/MADALINE utilizou saídas analógicas em uma arquitetura de três camadas.

Histórico (1969)

- Foi constatado por Minsky & Papert que um neurônio do tipo Perceptron só é capaz de resolver problemas com dados de classes linearmente separáveis.

Histórico (1960-1970)

Muitos historiadores desconsideraram a existência de pesquisa nessa área nos anos 60 e 70.

Histórico (1982)

Retomada das pesquisas com a publicação dos trabalhos do físico e biólogo Hopfield relatando a utilização de redes simétricas para otimização, através de um algoritmo de aprendizagem que estabilizava uma rede binária simétrica com realimentação.

Histórico (1986)

- Rumelhart, Hinton e Williams introduziram o poderoso método de treinamento denominado “Backpropagation”.
- Rumelhart e McClelland escreveram o livro “Processamento Paralelo Distribuído: Explorações na Microestrutura do Conhecimento”.

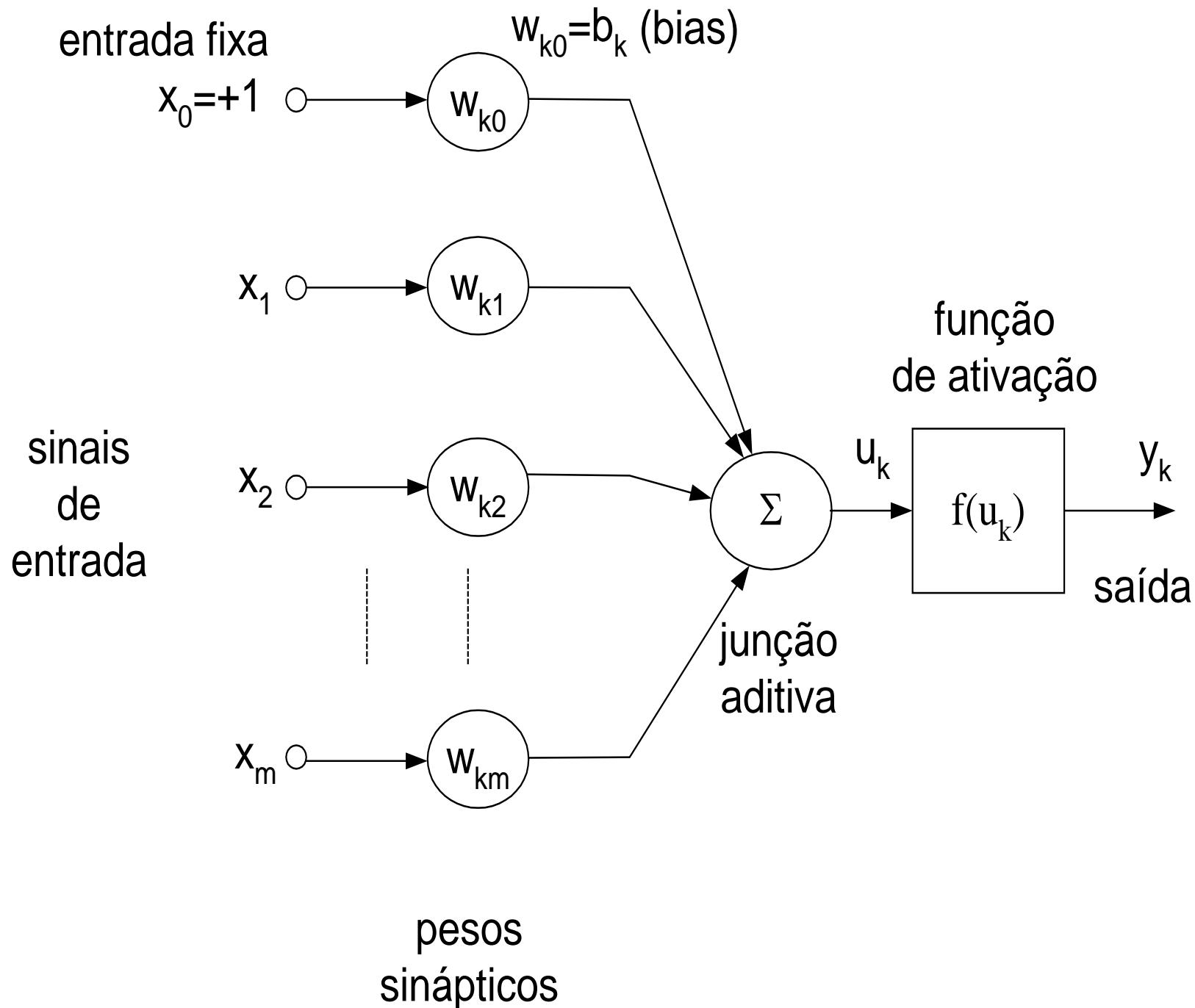
Histórico (1988)

- Broomhead e Lowe descreveram um procedimento para o projeto de uma rede neural (feedforward) usando funções de base radial (Rede de Base Radial – RBF).

Neurônio artificial

Componentes do neurônio artificial

- As sinapses (entradas), com seus pesos associados
- A junção somadora; e
- A função de ativação.



Princípio de funcionamento

A operação de um neurônio artificial se resume em:

- Sinais são apresentados à entrada (x_1 à x_m);
- Cada sinal é multiplicado por um peso que indica sua influência na saída da unidade (w_k);
- É feita a soma ponderada dos sinais que produz um nível de atividade (u_k);
- A função de ativação $f(u_k)$ tem a função de limitar a saída e introduzir não-linearidade ao modelo.
- O bias b_k tem o papel de aumentar ou diminuir a influência do valor das entradas.
- É possível considerar o bias como uma entrada de valor constante 1, multiplicado por um peso igual a b_k .

Expressão matemática do neurônio artificial

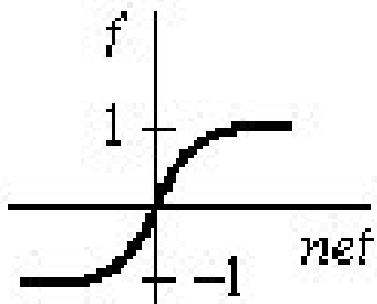
- Matematicamente a saída pode ser expressa por:

$$y_k = f(u_k) = f\left(\sum_{j=1}^m w_{kj}x_j + b_k\right)$$

ou considerando o bias como entrada de valor $x_0=1$ e peso $w_{k0}=b_k$,

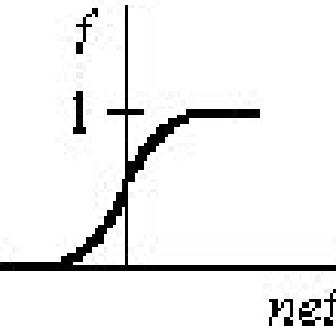
$$y_k = f(u_k) = f\left(\sum_{j=0}^m w_{kj}x_j\right)$$

Funções de ativação



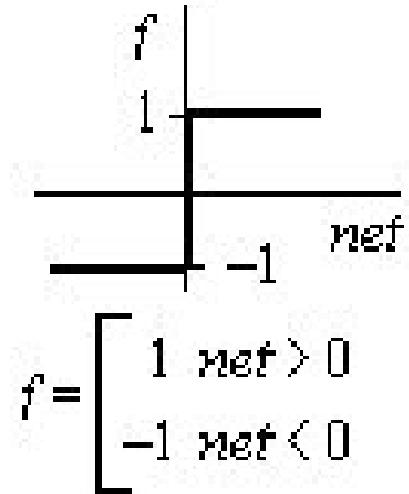
$$f = \tanh(\alpha \cdot \text{net})$$

tanh



$$f = \frac{1}{1 + \exp(-\alpha \cdot \text{net})}$$

logistic

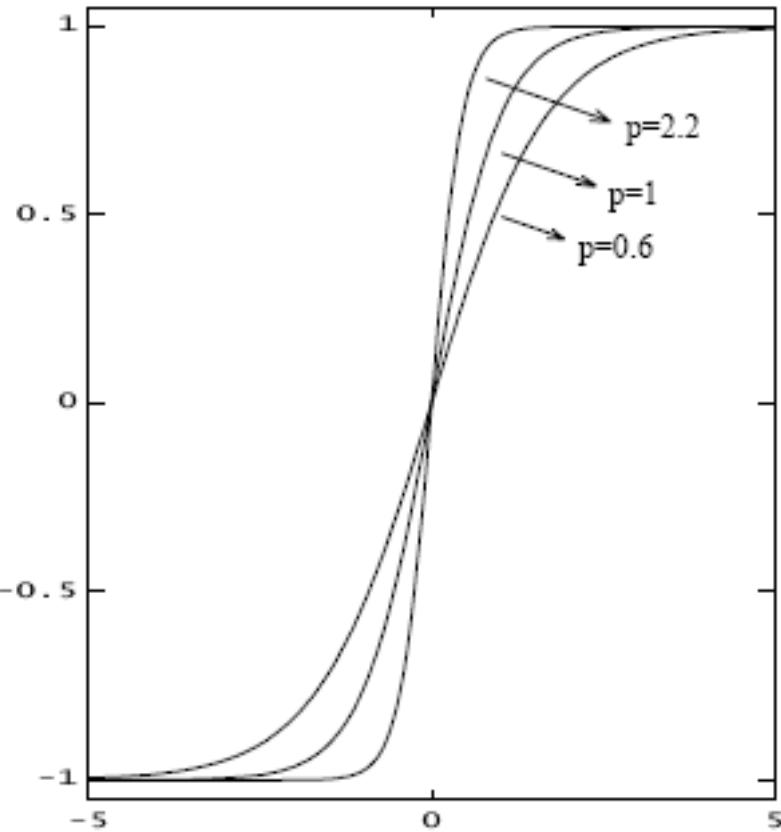


$$f = \begin{cases} 1 & \text{net} > 0 \\ -1 & \text{net} < 0 \end{cases}$$

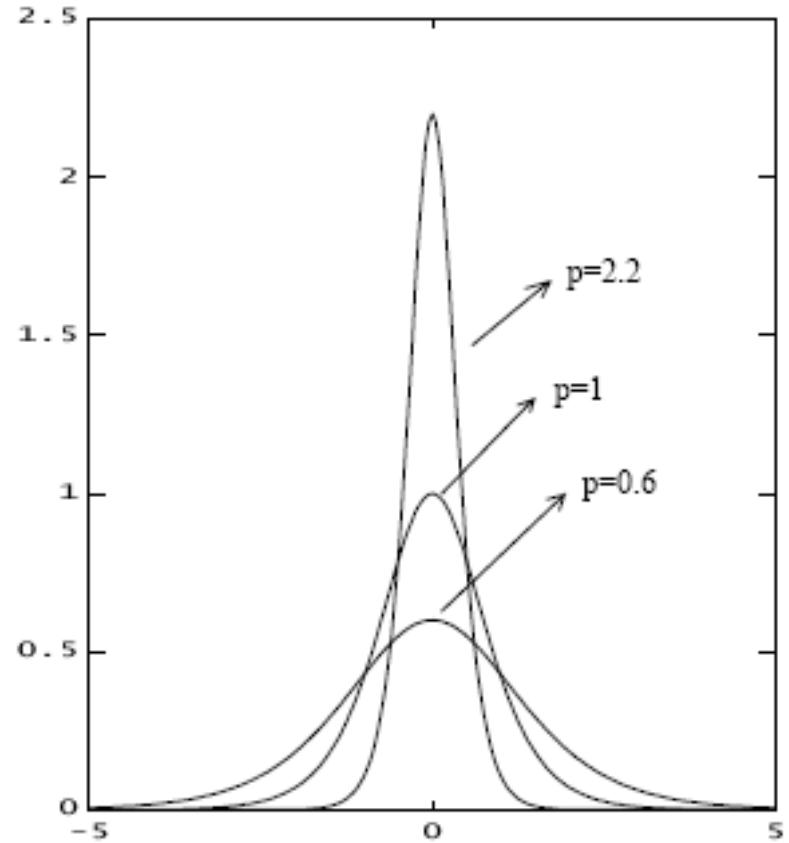
threshold

Função de ativação tangente hiperbólica

$$f(u_k) = \tanh(pu_k) = \frac{e^{pu_k} - e^{-pu_k}}{e^{pu_k} + e^{-pu_k}} \quad \frac{\partial f}{\partial u_k} = p(1 - u_k^2) > 0$$



(a)



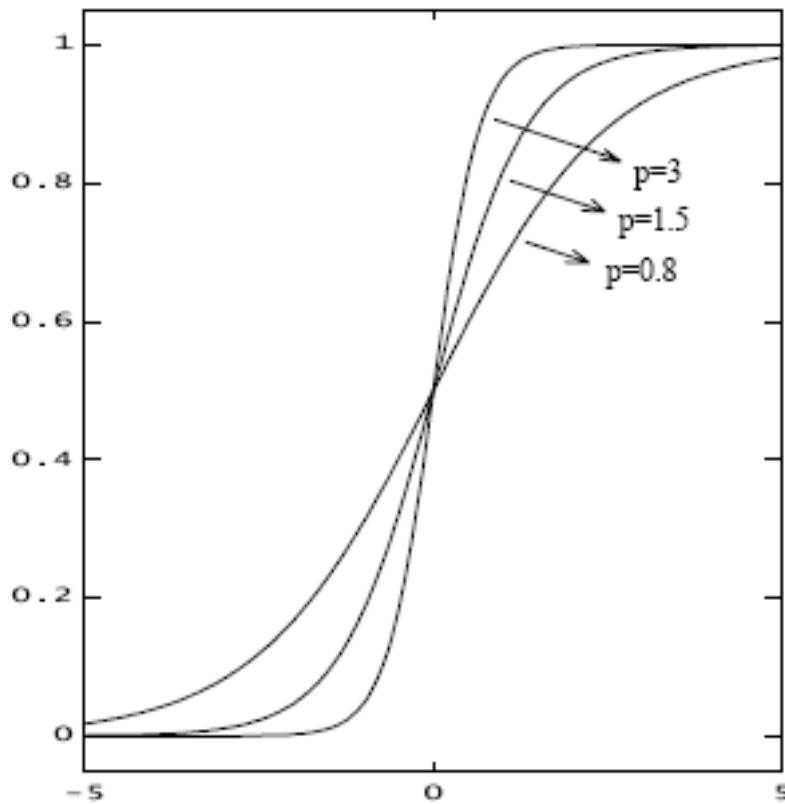
(b)

Tangente hiperbólica (a) e sua derivada (b).

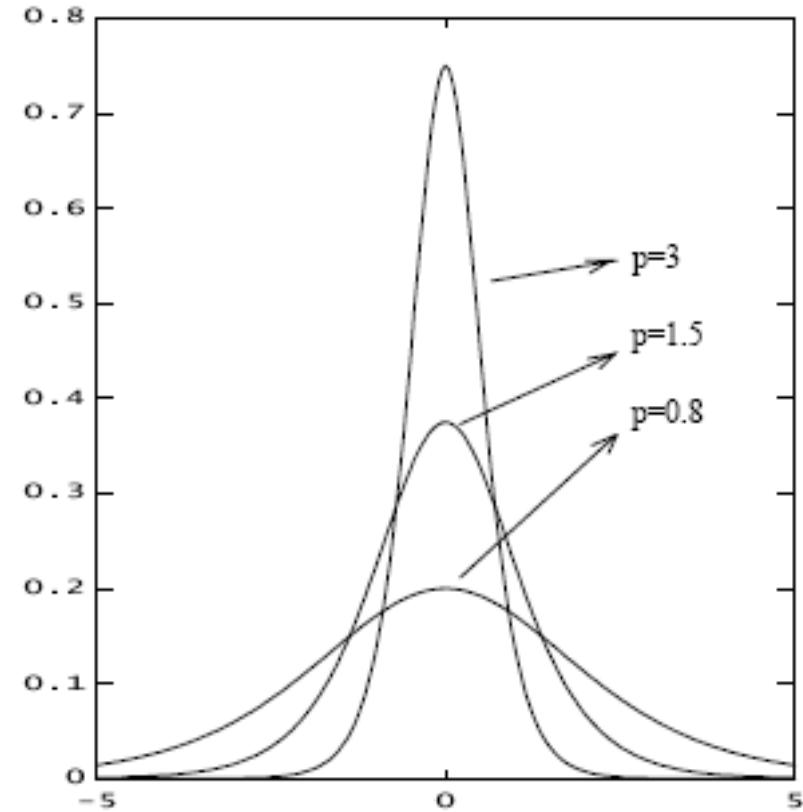
Função de ativação logística (sigmóide)

$$f(u_k) = \frac{e^{pu_k}}{e^{pu_k} + 1} = \frac{1}{1 + e^{-pu_k}}$$

$$\frac{\partial f}{\partial u_k} = pu_k(1 - u_k) > 0$$



(a)



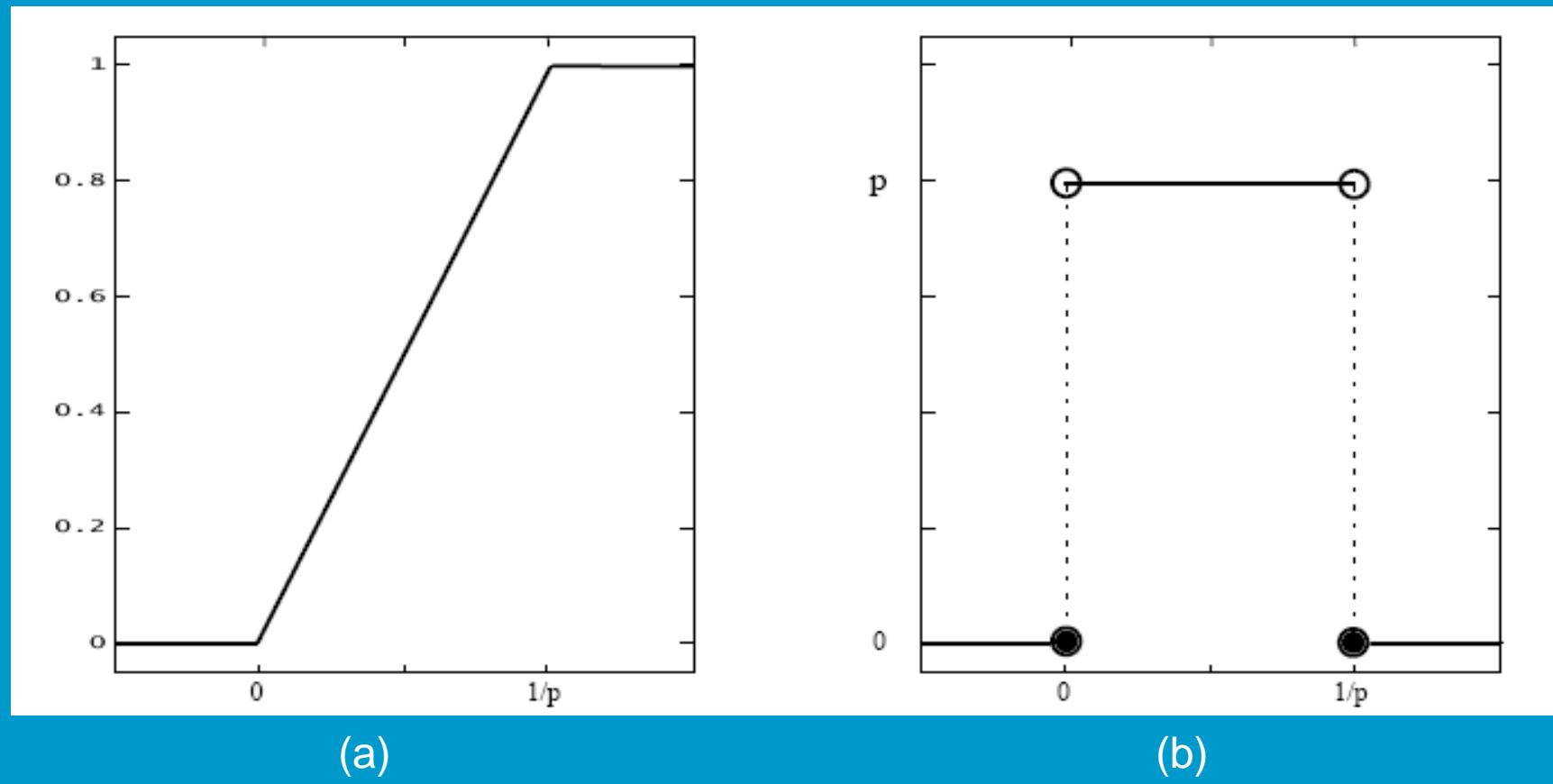
(b)

Sigmóide (a) e sua derivada (b).

Função de ativação semi-linear

$$f(u_k) = \begin{cases} 1 & \text{se } pu_k \geq 1 \\ pu_k & \text{se } 0 < pu_k < 1 \\ 0 & \text{se } pu_k \leq 0 \end{cases}$$

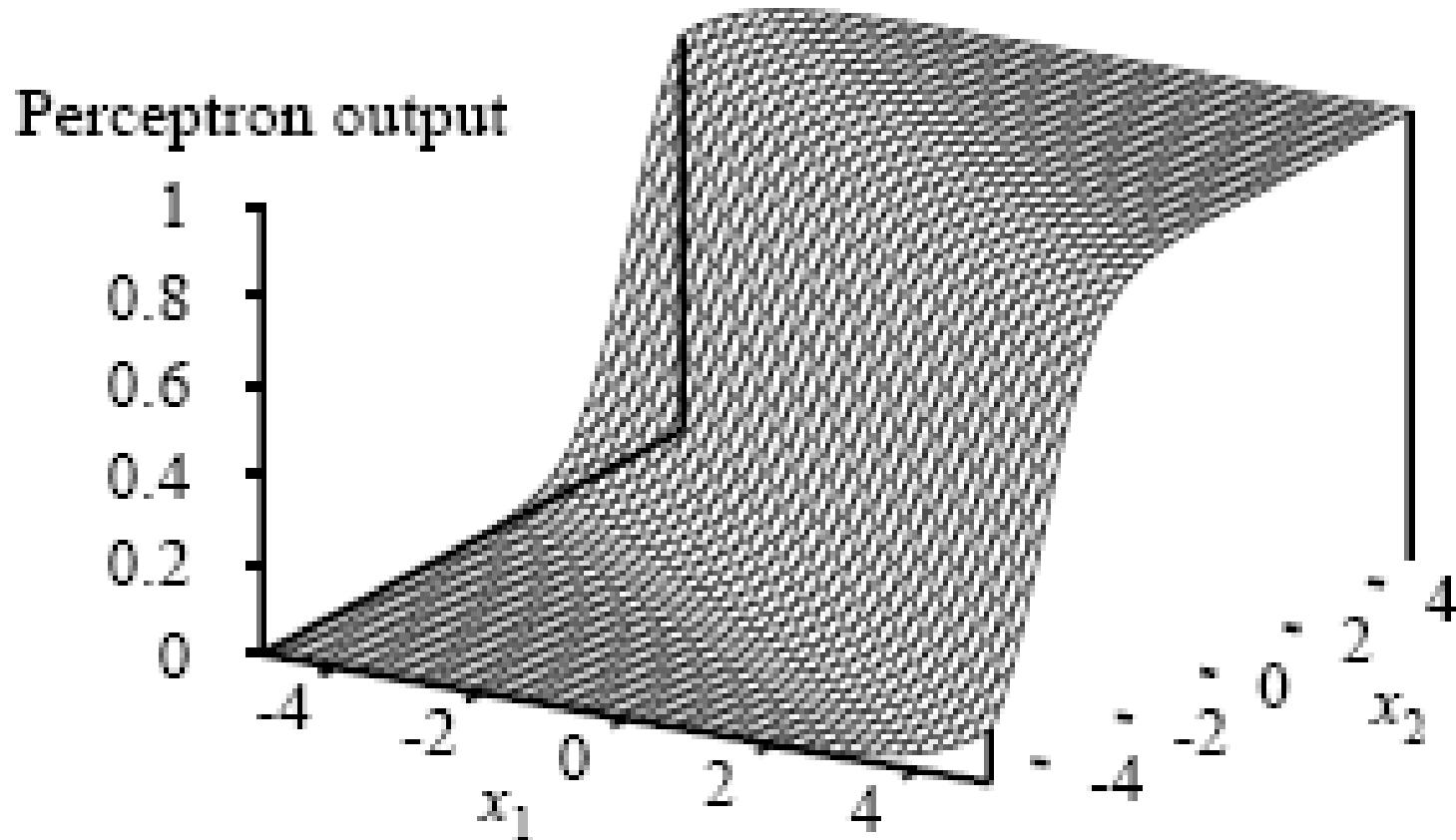
p constante e positivo



Função de ativação semi-linear (a) e sua derivada (b).

Resposta do neurônio (saída)

Considerando um neurônio artificial com duas entradas (x_1, x_2) e função de ativação sigmóide:



Separação Linear

Sabe-se que se formarmos uma combinação linear de duas variáveis, e igualá-la a um número, então os pontos no espaço bidimensional podem ser divididos em três categorias:

a) pontos pertencentes à linha com coordenadas tais que

$$w_1 \cdot x_1 + w_2 \cdot x_2 = q$$

b) pontos em um lado da linha tem coordenadas tais que

$$w_1 \cdot x_1 + w_2 \cdot x_2 < q$$

c) pontos no outro lado da linha tem coordenadas tais que

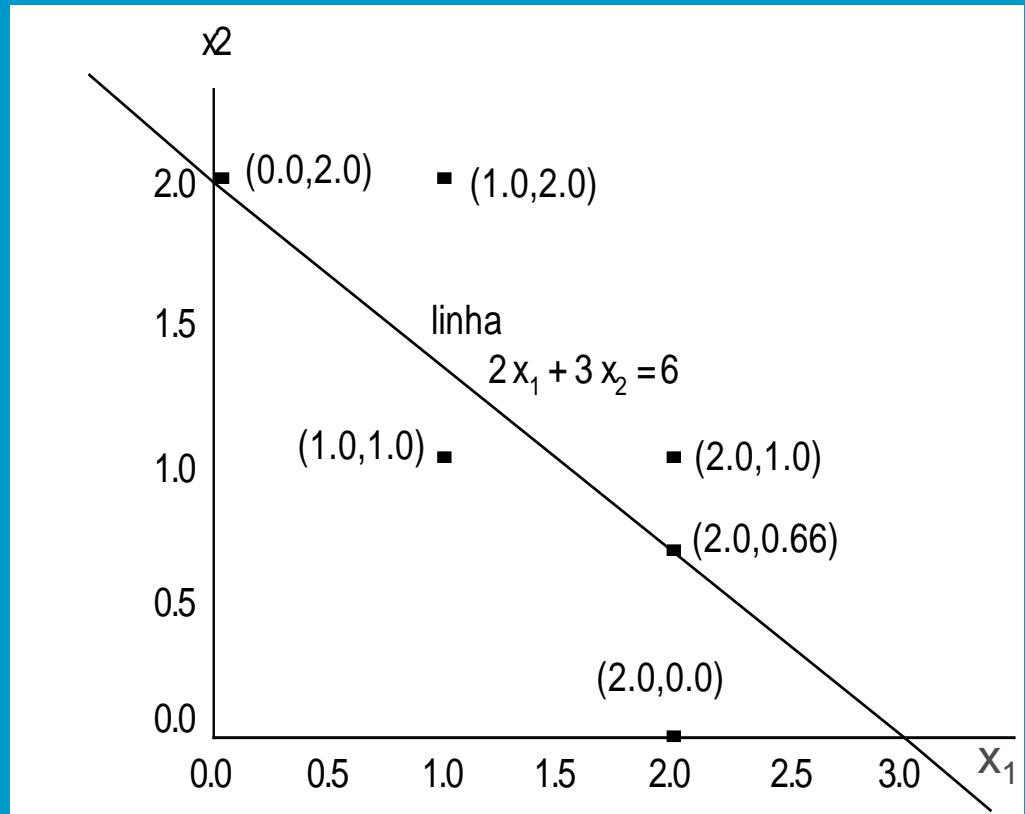
$$w_1 \cdot x_1 + w_2 \cdot x_2 > q.$$

Nota: bias $b = -q$, pois:

$$y = f(w_1 x_1 + w_2 x_2 + w_0), \text{ onde } w_0 = b$$

Exemplo

Pontos	$2x_1 + 3x_2$	posição
(x_1, x_2)		
$(0.0, 2.0)$	6	linha
$(1.0, 1.0)$	5	abaixo
$(1.0, 2.0)$	8	acima
$(2.0, 0.0)$	4	abaixo
$(2.0, 0.66)$	6	linha
$(2.0, 1.0)$	7	acima

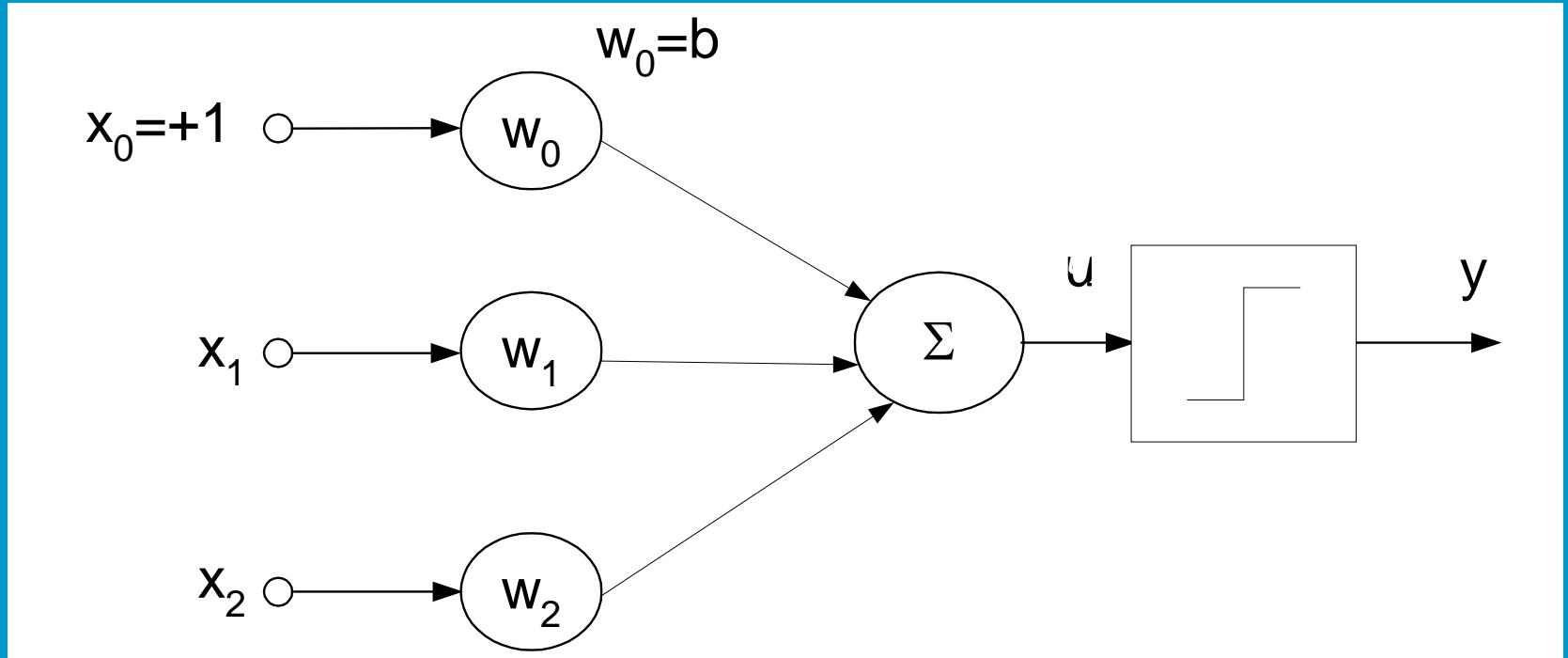


Posição dos pontos em função da linha $2x_1 + 3x_2 = 6$ de delimitação.

Linha: $2x_1 + 3x_2 = 6$

Acima: $2x_1 + 3x_2 > 6$ $q = 6$

Abaixo: $2x_1 + 3x_2 < 6$



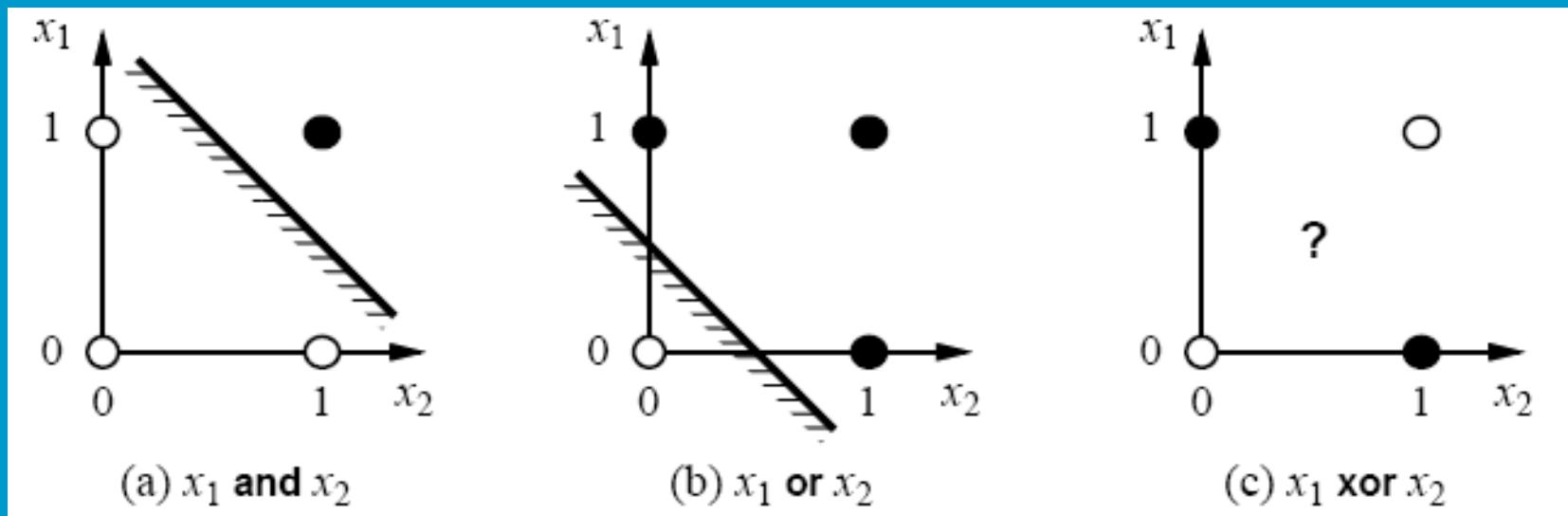
$$y = f(w_1x_1 + w_2x_2 + w_0), \text{ sendo } \begin{cases} f(u) = 1 & \text{se } u \geq 0 \\ f(u) = 0 & \text{se } u < 0 \end{cases}$$

Com os parâmetros w_0 , w_1 e w_2 , a função $f(u)$ separa o espaço de entradas em duas regiões, usando uma linha reta dada por:

$$w_1x_1 + w_2x_2 + w_0 = 0$$

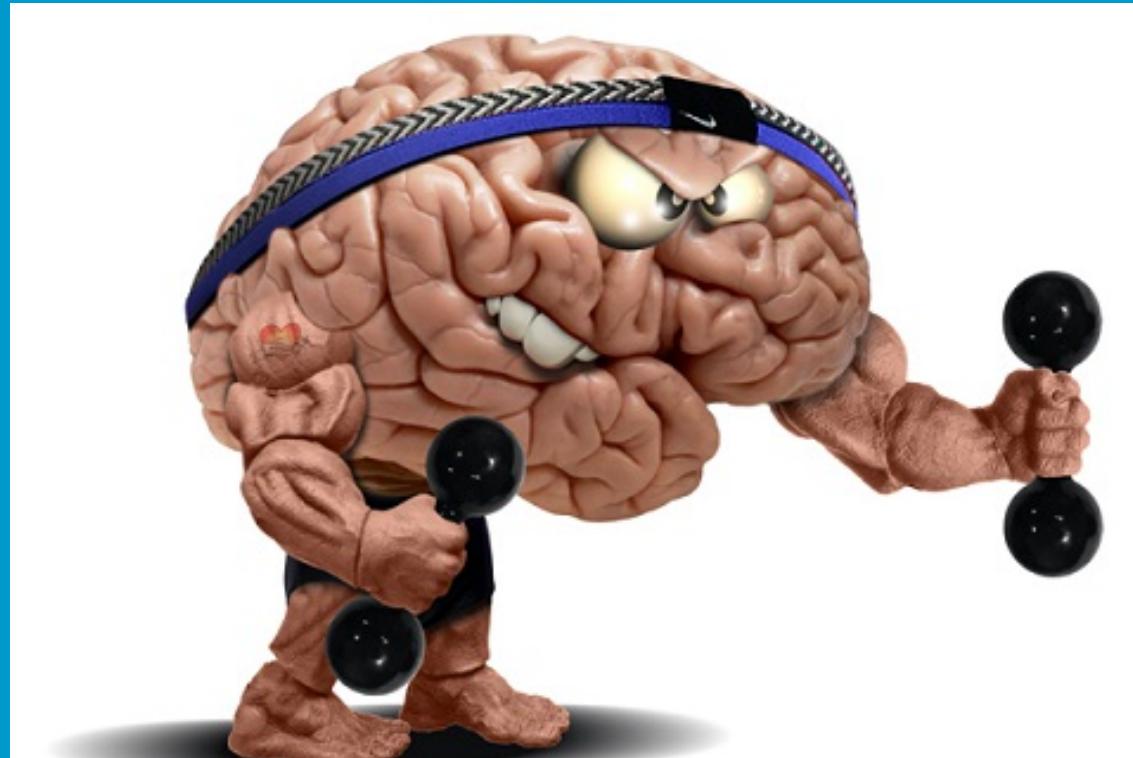
Perceptron de limiar

- O perceptron de limiar é chamado **separador linear**
 - Porque traça um plano entre os pontos de entrada onde a saída é zero ou um



Funções linearmente separáveis (a) e (b)

Treinamento do neurônio



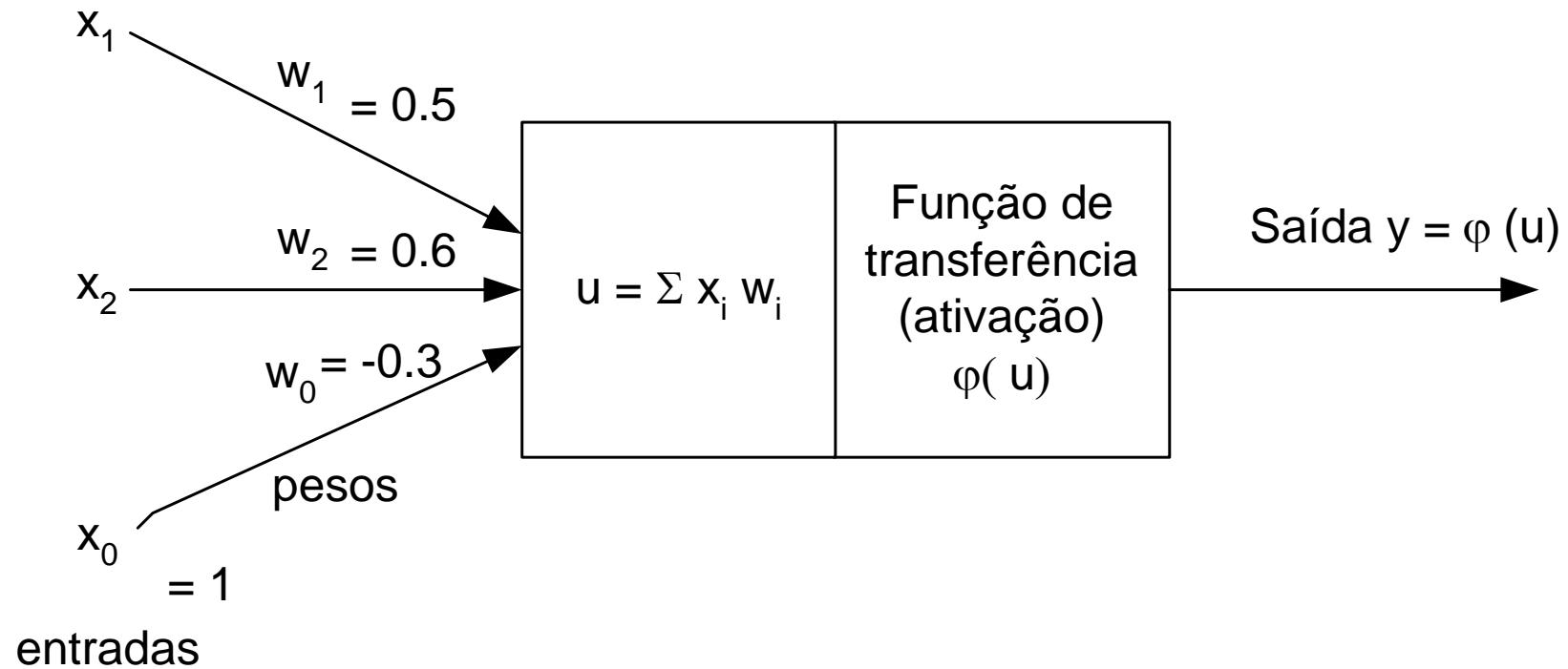
Exemplo: um neurônio para a função AND de 2 entradas

Iniciamos o neurônio com os pesos 0.5 e 0.6 para as duas entradas, e -0.3 para o limiar (w_0). Isso é equivalente à equação:

$$u = 0.5 \ x_1 + 0.6 \ x_2 - 0.3 \ x_0$$

onde x_0 é a entrada estável sempre igual a 1.

Assim, a saída y deve disparar quando $u \geq 0$).



x_1	x_2	u	y
0	0	-0.3	0
0	1	0.3	1
1	0	0.2	1
1	1	0.8	1

a saída é 1 para os padrões 01, 10 e 11, enquanto desejamos que a saída seja 1 somente para o padrão 11.

Seqüência de passos na aplicação do algoritmo

Início

Entrada 0 0 $u = -0.3$ $y = 0$ correta

Entrada 0 1 $u = 0.3$ $y = 1$ incorreta

Correção dos pesos de 0.1 para baixo →

Entrada 1 0 $u = 0.1$ $y = 1$ incorreta

Correção dos pesos de 0.1 para baixo →

Entrada 1 1 $u = 0.4$ $y = 1$ correta

Entrada 0 0 $u = -0.5$ $y = 0$ correta

Entrada 0 1 $u = 0$ $y = 1$ incorreta

Correção dos pesos de 0.1 para baixo →

Entrada 1 0 $u = -0.2$ $y = 0$ correta

Entrada 1 1 $u = 0.2$ $y = 1$ correta

Entrada 0 0 $u = -0.6$ $y = 0$ correta

Entrada 0 1 $u = -0.2$ $y = 0$ correta

Entrada 1 0 $u = -0.2$ $y = 0$ correta

Entrada 1 1 $u = 0.2$ $y = 1$ correta

Fim

$w_1 = 0.5$ $w_2 = 0.6$ $w_0 = -0.3$

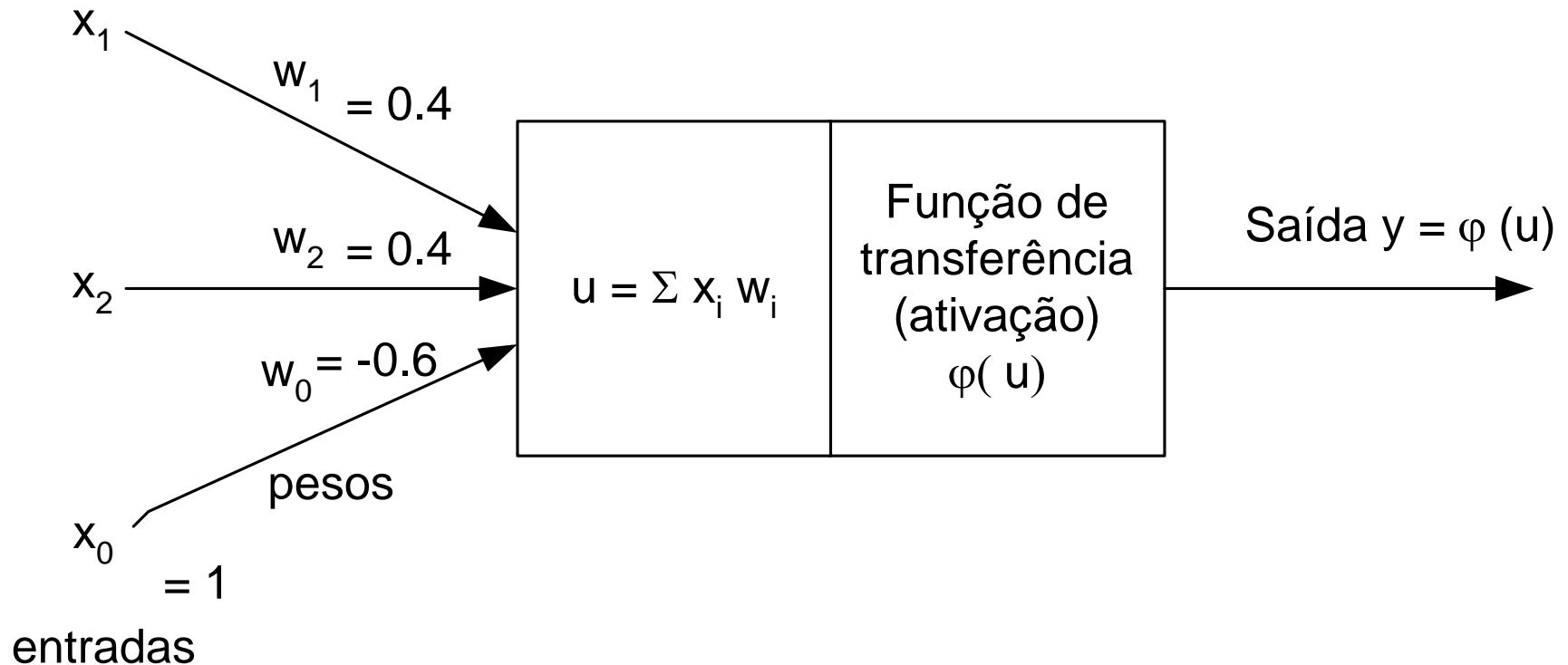
$w_1 = 0.5$ $w_2 = 0.5$ $w_0 = -0.4$

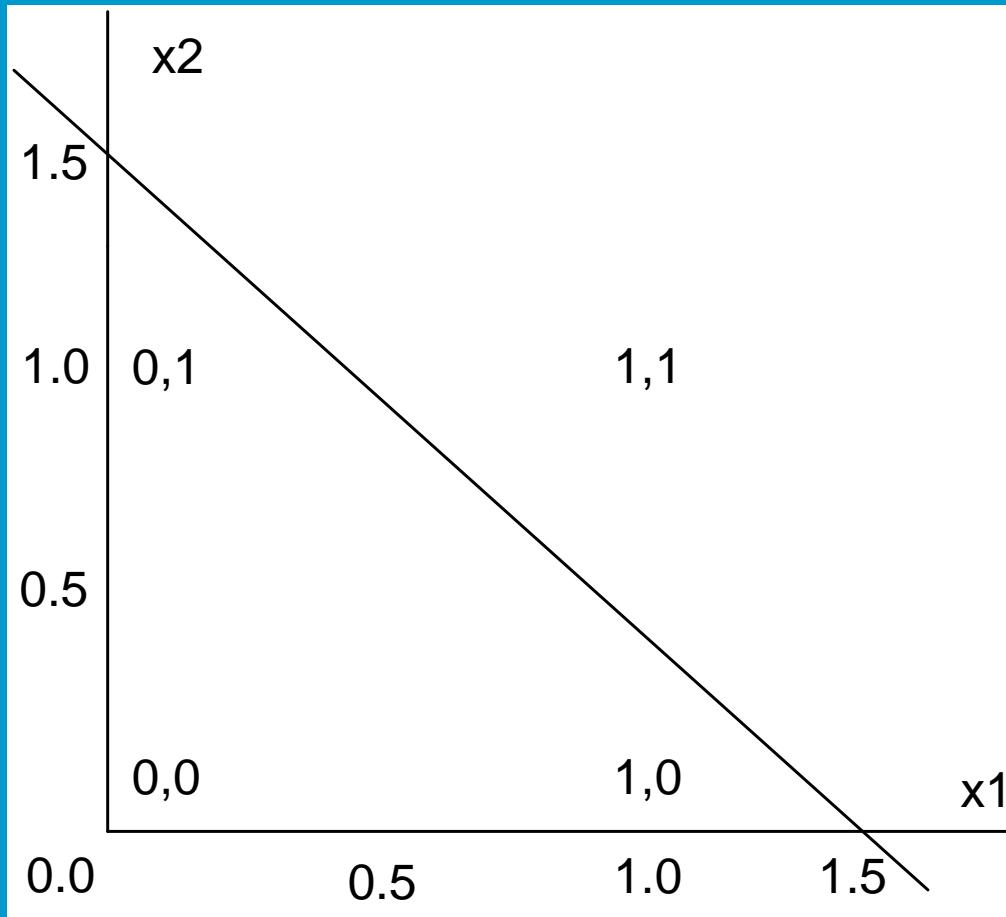
$w_1 = 0.4$ $w_2 = 0.5$ $w_0 = -0.5$

$w_1 = 0.4$ $w_2 = 0.4$ $w_0 = -0.6$

$w_1 = 0.4$ $w_2 = 0.4$ $w_0 = -0.6$

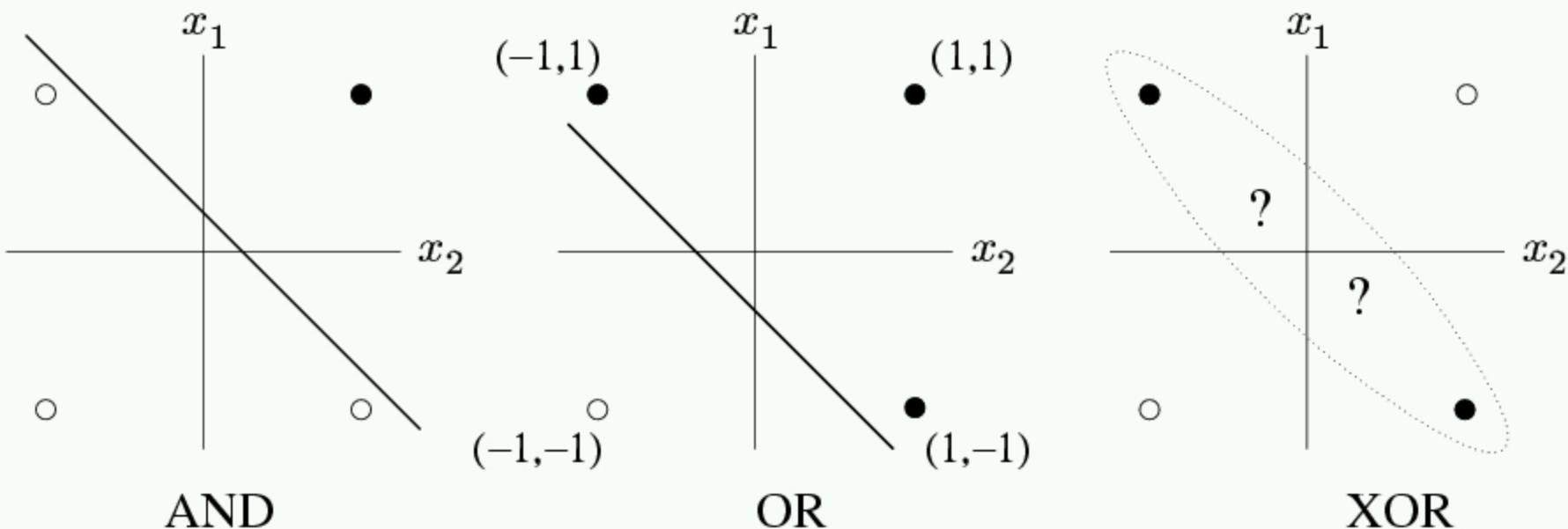
Resultado do aprendizado





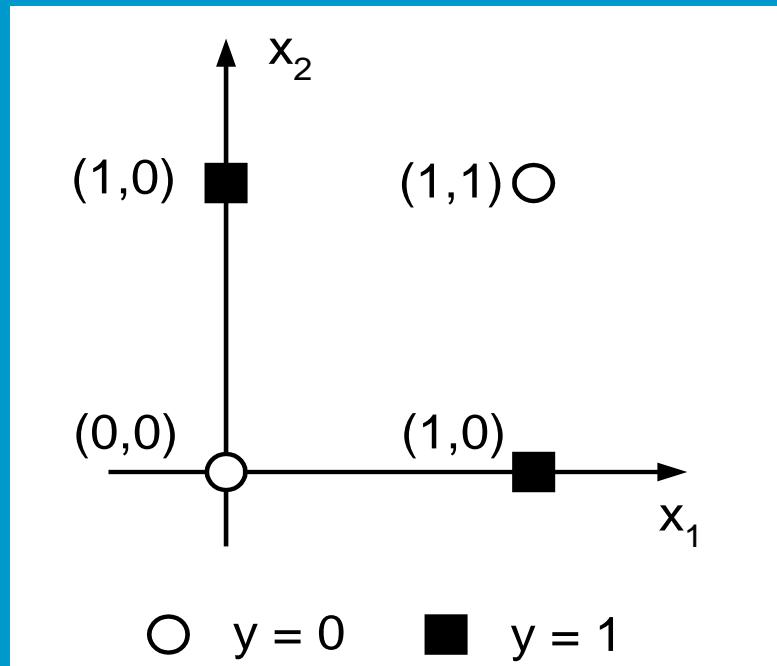
A reta $0.4x_1 + 0.4x_2 - 0.6 = 0$ separa os pontos 00 , 01 e 10 , do ponto 11 .

Exemplo: um neurônio para a função XOR



No caso do XOR, não existe uma única reta que divide os pontos $(0,0)$ e $(1,1)$ para um lado, e $(0,1)$ e $(1,0)$ do outro lado.

Função xor:



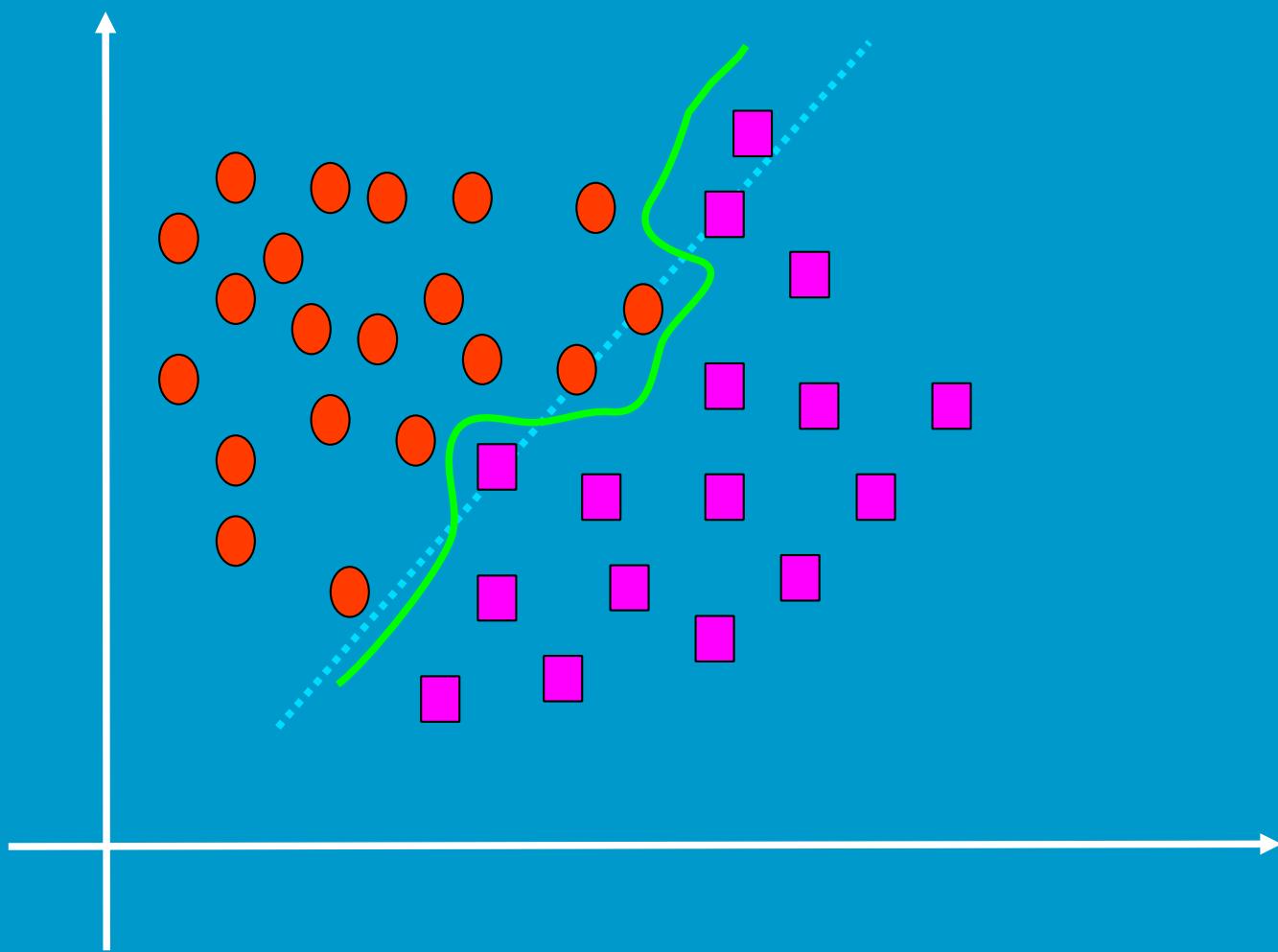
Conclui-se que um neurônio do tipo Perceptron não implementa uma função ou-exclusivo (constatado por Minsky & Papert, em 1969).

Algoritmo de treinamento do Perceptron

Para classificação padrões de entrada como pertencentes ou não a uma dada classe, considere o conjunto de treinamento formado por N amostras $\{\mathbf{x}_1, d_1\}, \{\mathbf{x}_2, d_2\}, \dots, \{\mathbf{x}_N, d_N\}$, onde \mathbf{x}_j é o vetor de entradas e d_j a saída desejada (classe), que em notação vetorial tem-se $\{\mathbf{X}, \mathbf{d}\}$, onde:

$$\mathbf{X} \in \Re^{m \times N}$$

$$\mathbf{d} \in \Re^{1 \times N}$$



Aprendizagem de Perceptrons

■ O algoritmo

- Executa os exemplos de treinamento através da rede;
- Ajusta os pesos depois de cada exemplo para reduzir o erro;
- Cada ciclo através dos exemplos é chamado de **época**;
- As épocas são repetidas até que se alcance algum critério de parada.
 - Em geral, quando as mudanças nos pesos forem pequenas.

Se os padrões de entrada forem linearmente separáveis, o algoritmo de treinamento possui convergência garantida, i.é, tem capacidade para encontrar um conjunto de pesos que classifica corretamente os dados.

Perceptrons de limiar

- Algoritmo de aprendizagem simples:
 - Adaptará o perceptron de limiar a qualquer conjunto de treinamento linearmente separável
 - Idéia: ajustar os pesos da rede para minimizar alguma medida de erro no conjunto de treinamento
 - Aprendizagem = busca de otimização no espaço de pesos
 - Medida clássica de erros = soma dos erros quadráticos

Função [w] = perceptron (max_it, E, α, X,d)

inicializar w // para simplicidade, com zeros

inicializar b // para simplicidade, com zero

t \leftarrow 1

while t < max_it & E > 0 do

for i from 1 to N do

$y_i \leftarrow f(w x_i + b)$

$e_i \leftarrow d_i - y_i$

$w \leftarrow w + \alpha e_i x_i$

$b \leftarrow b + \alpha e_i$

end for

$E \leftarrow \text{sum } (e_i)$

$t \leftarrow t + 1$

end while

end procedure

// para cada padrão de entrada

// determinar a saída

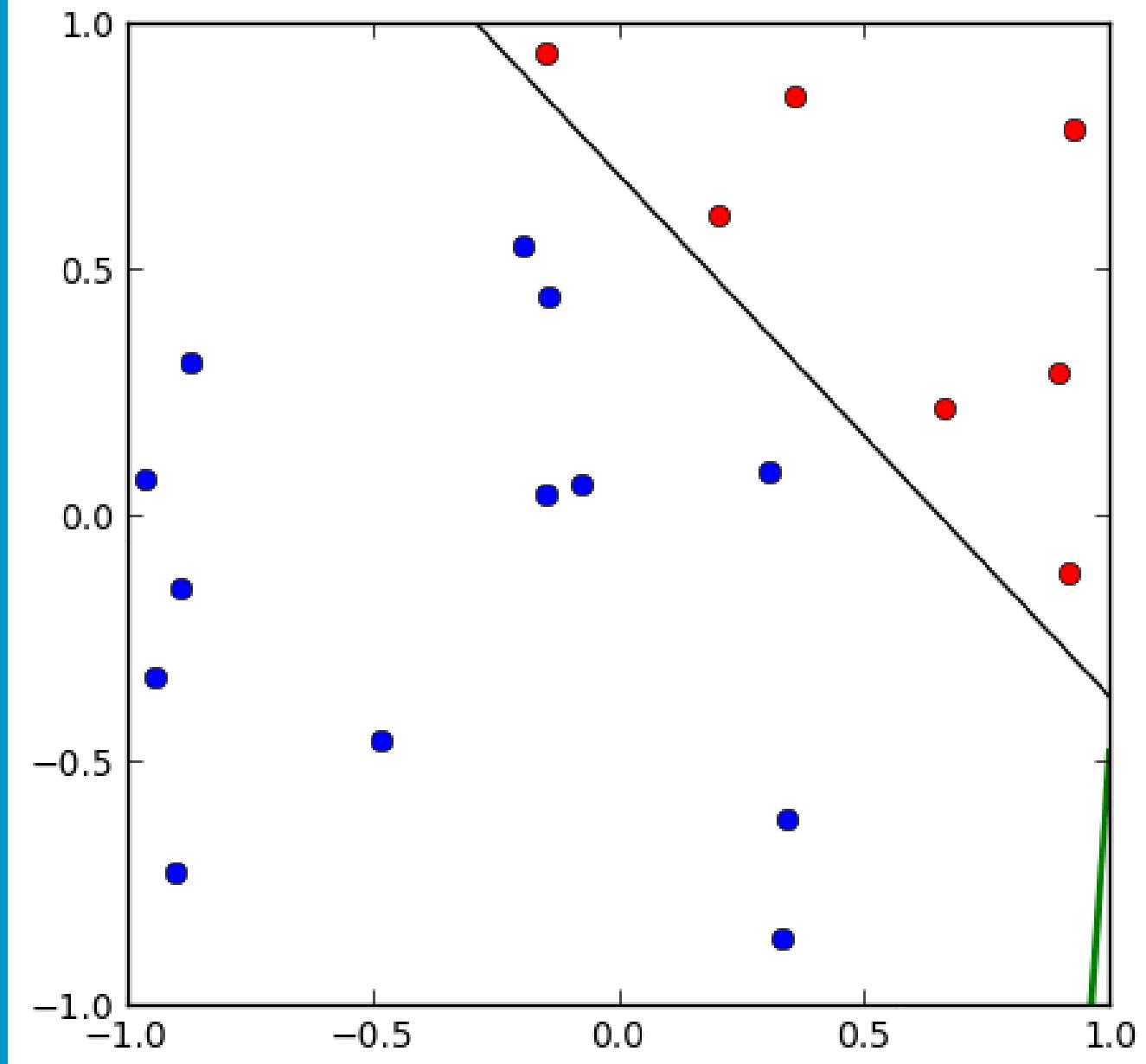
// determinar o erro

// atualizar o vetor peso

// atualizar o bias

//quantidade de erros

$N = 20$, Iteration 1



Adaline

- Na mesma época em que Rosenblatt propôs o Perceptron, Widrow e Hoff propuseram o algoritmo dos mínimos quadrados (regra delta) para a rede Adaline (Adaptive Linear Element), similar ao Perceptron, porém com função de ativação linear ao invés de função degrau.
- O objetivo do algoritmo de treinamento é minimizar o erro quadrático médio (MSE) entre a saída de rede e a saída desejada.

- A soma dos erros quadráticos para um determinado padrão é dada por:

$$E = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (d_i - y_i)^2$$

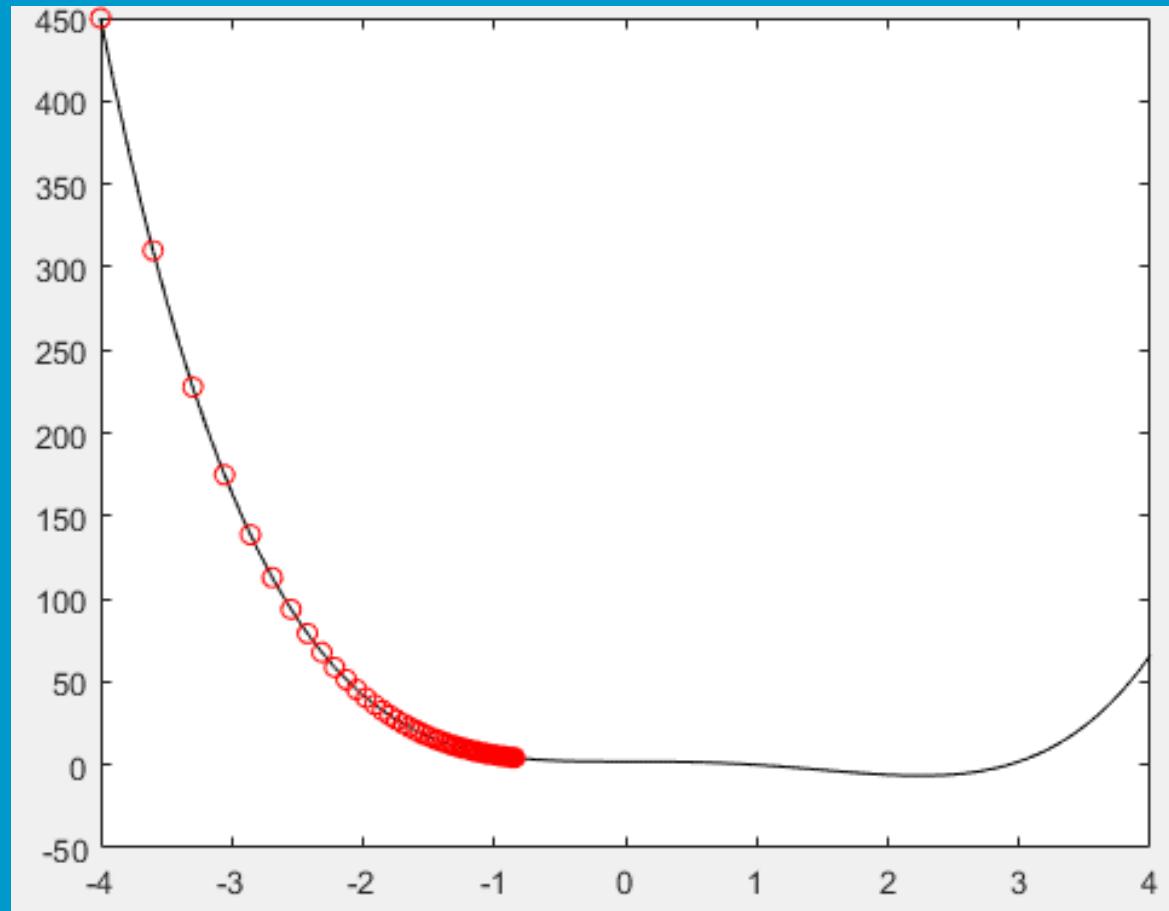
- O gradiente de E, também denominado de índice de desempenho ou função custo, fornece a direção de crescimento mais rápido de E.
- Portanto, a direção oposta ao gradiente de E é a direção de maior decrescimento.

- Um gradiente é a razão segundo a qual uma quantidade variável aumenta ou diminui.
- A derivada de uma função $y = f(x)$ é a razão entre os acréscimos infinitesimais da função y e da variável x . A derivada é portanto uma taxa de variação instantânea

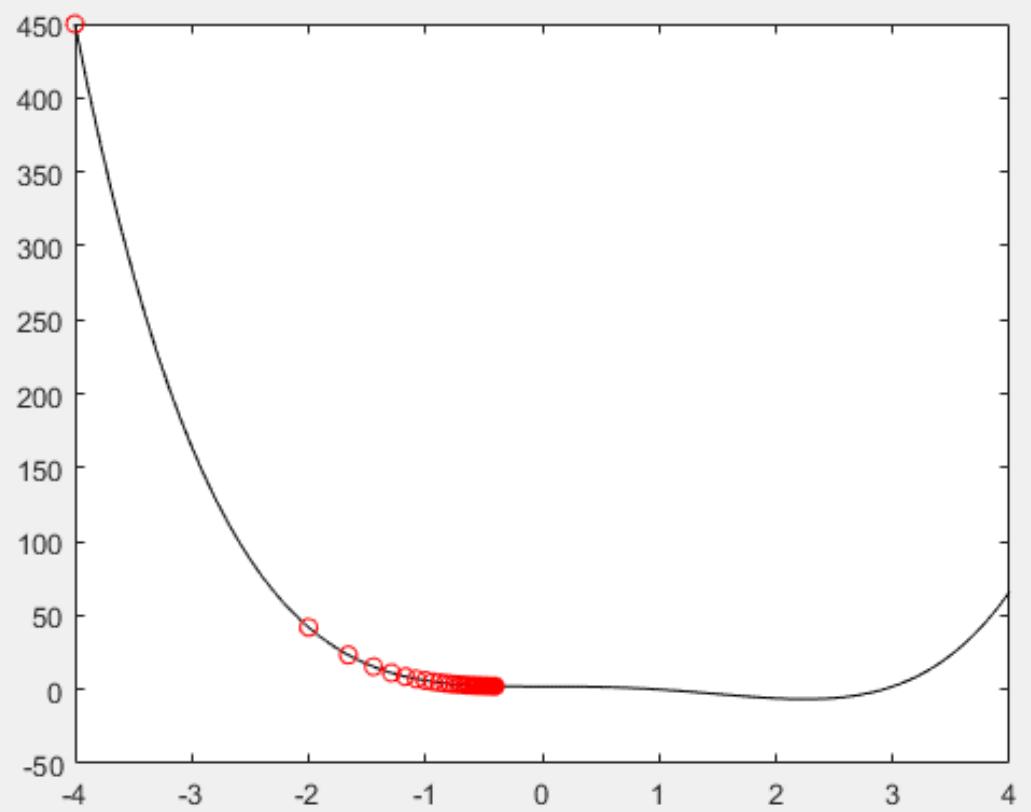
- Vamos considerar a função $y = f(x) = x^4 * 3x^3 + 2$
- Queremos “descobrir” o valor de x em que $f(x)$ seja mínimo
- Considere que temos o ponto $x=-1,5$ definido aleatoriamente

- função $y = f(x) = x^4 * 3x^3 + 2$
- Precisamos da derivada da função, lembrando que, a derivada de x^a é igual a $a * x^{a-1}$
- Logo, a derivada da função no ponto $x=-1,5$
- $\frac{dy(1,5)}{dx} = 4 * x^3 + 9x^2 = 4 * (1,5)^3 * 9 * (1,5)^2$
- $\frac{dy(1,5)}{dx} = -33,75$
- Vejam a “mágica” do gradiente que nos indica a direção do mínimo da função
- $x_{novo} = (-1,5) + 0,001 * 33,75((-1,5) = -1,49$

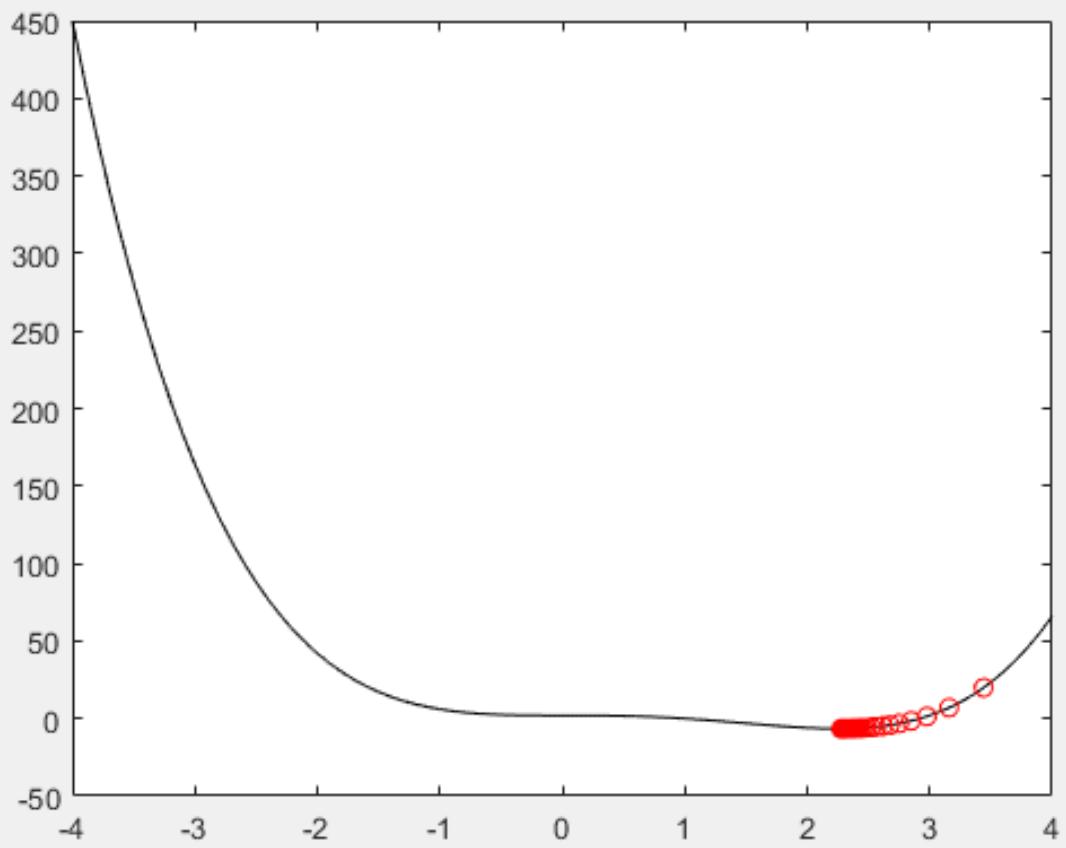
Taxa de aprendizado baixa



Valor inicial: -4
Taxa de aprend.: 0,001

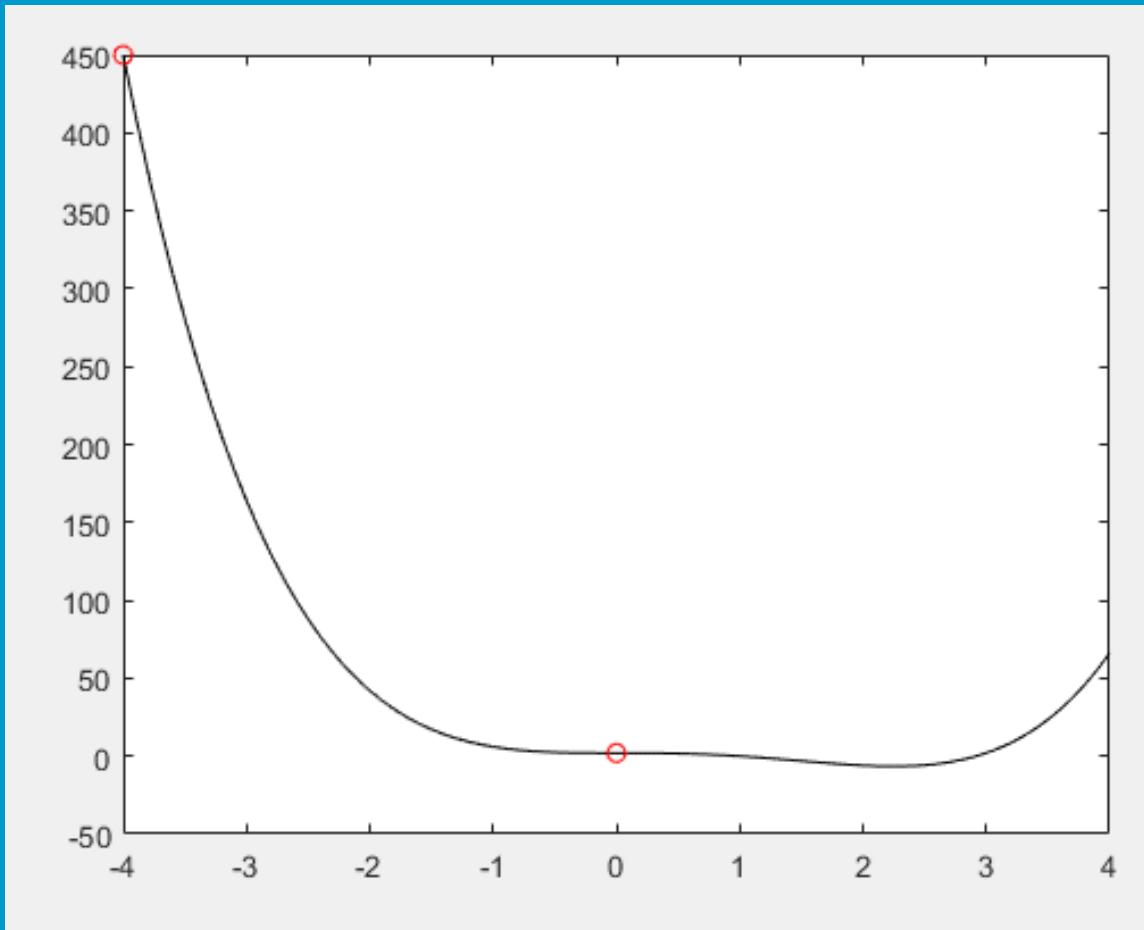


Valor inicial: -4
Taxa de aprend.: 0,005

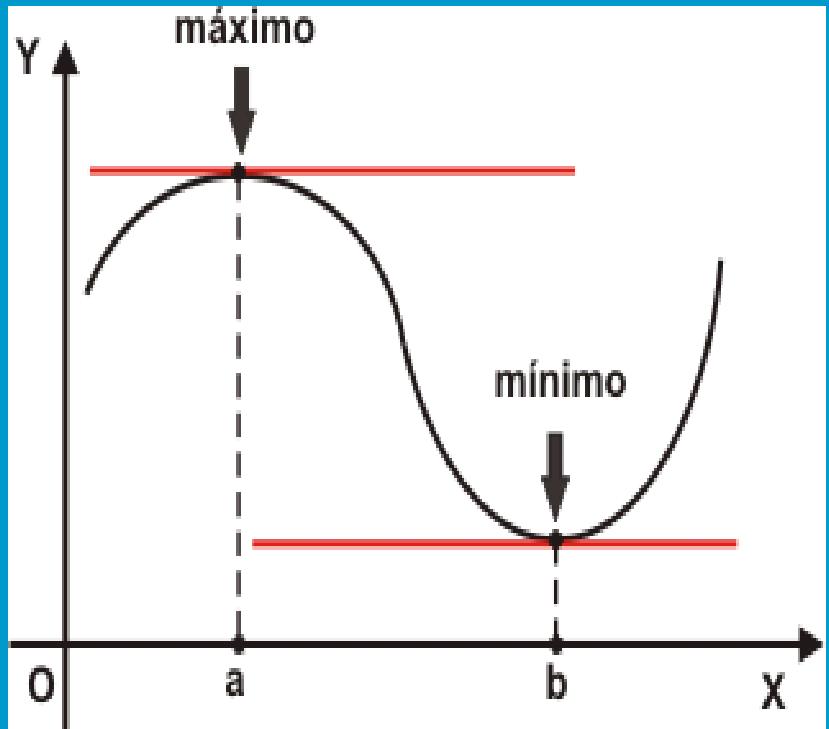


Valor inicial: 4
Taxa de aprend.: 0,005

Taxa de aprendizado alta

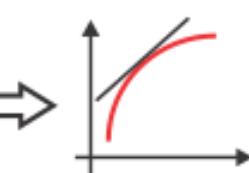
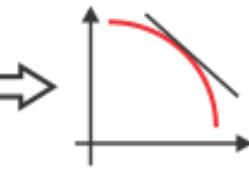
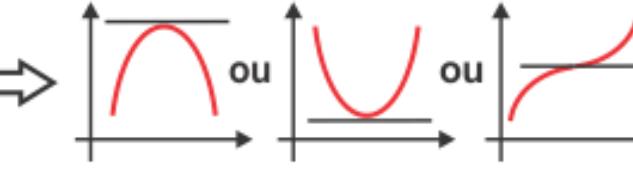


Valor inicial: -4
Taxa de aprend.: 0,05



- Qual é o valor da derivada quando a função passa por um valor máximo ou mínimo ?
- Quando a função passa por um máximo ou por um mínimo a tangente é paralela ao eixo X .

- A derivada primeira informa sobre a declividade do gráfico da função

Propriedade da derivada primeira		
Derivada primeira	Função	Gráfico
$f'(x) > 0$	crescente	
$f'(x) < 0$	decrescente	
$f'(x) = 0$	máximo mínimo inflexão	

Adaline (cont.)

O erro pode ser reduzido ajustando-se os pesos da rede de acordo com:

$$w_{IJ} = w_{IJ} - \alpha \frac{\partial E}{\partial w_{IJ}}$$

onde w_{IJ} é o peso específico para o neurônio pós-sináptico I , da entrada J , e α é a taxa de aprendizagem.

Como w_{IJ} influencia apenas o neurônio I,

$$\frac{\partial E}{\partial w_{IJ}} = \frac{\partial}{\partial w_{IJ}} \sum_{i=1}^n (d_i - y_i)^2 = \frac{\partial}{\partial w_{IJ}} (d_I - y_I)^2$$

Como

$$y_I = f(\mathbf{w}_I \cdot \mathbf{x}) = f(\sum_j w_{Ij} x_j) = \sum_j w_{Ij} x_j$$

$$\frac{\partial E}{\partial w_{IJ}} = -2(d_I - y_I) \frac{\partial y_I}{\partial w_{IJ}} = -2(d_I - y_I) x_J$$

Regra delta

Portanto a regra delta para o Adaline resume-se em:

$$w_{IJ} = w_{IJ} + \alpha(d_I - y_I)x_J$$
$$b_I = b_I + \alpha(d_I - y_I)$$

Em notação vetorial tem-se:

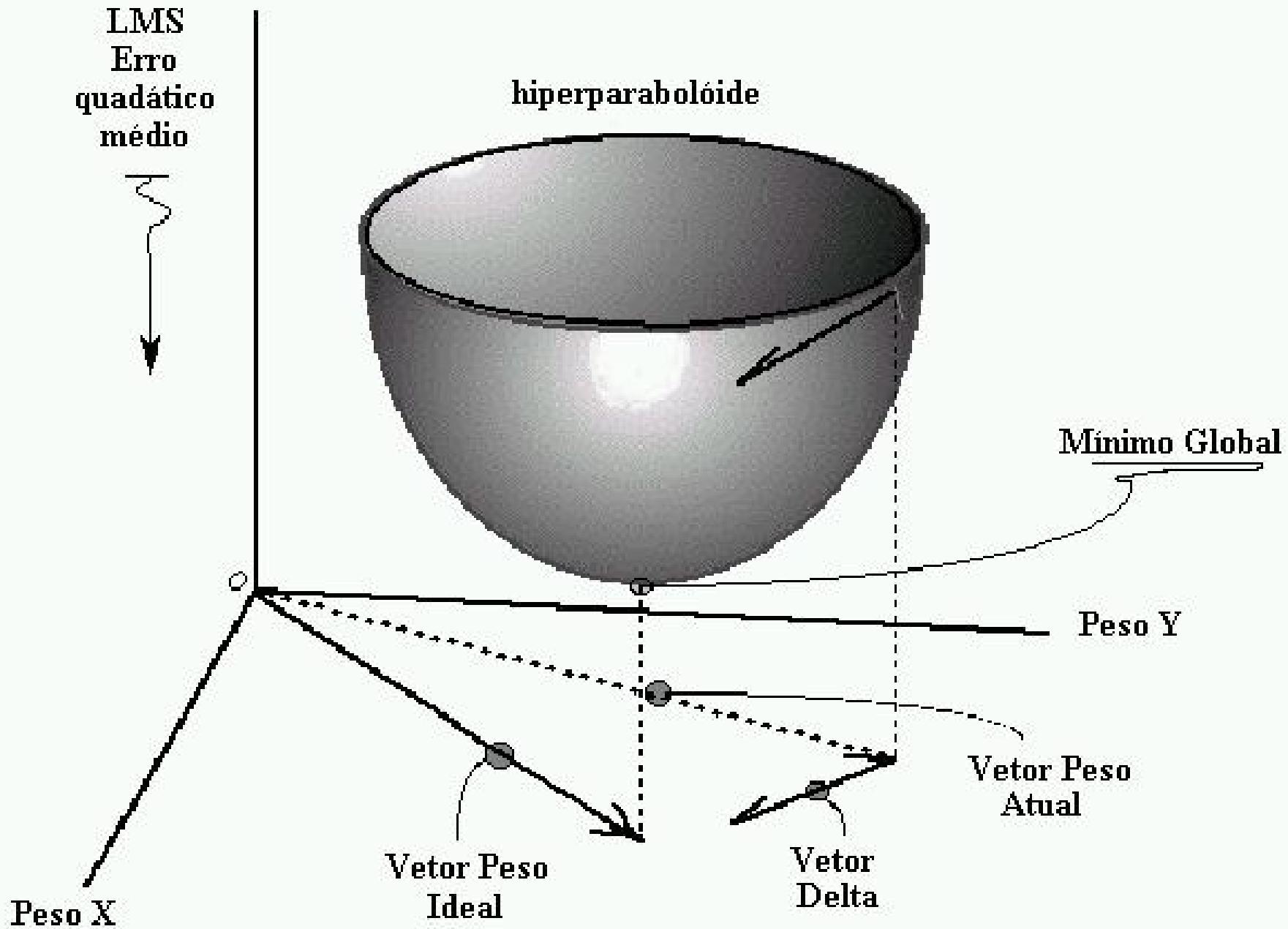
$$\mathbf{W} = \mathbf{W} + \alpha \mathbf{e}_i \mathbf{x}_i^T$$

$$\mathbf{b} = \mathbf{b} + \alpha \mathbf{e}_i$$

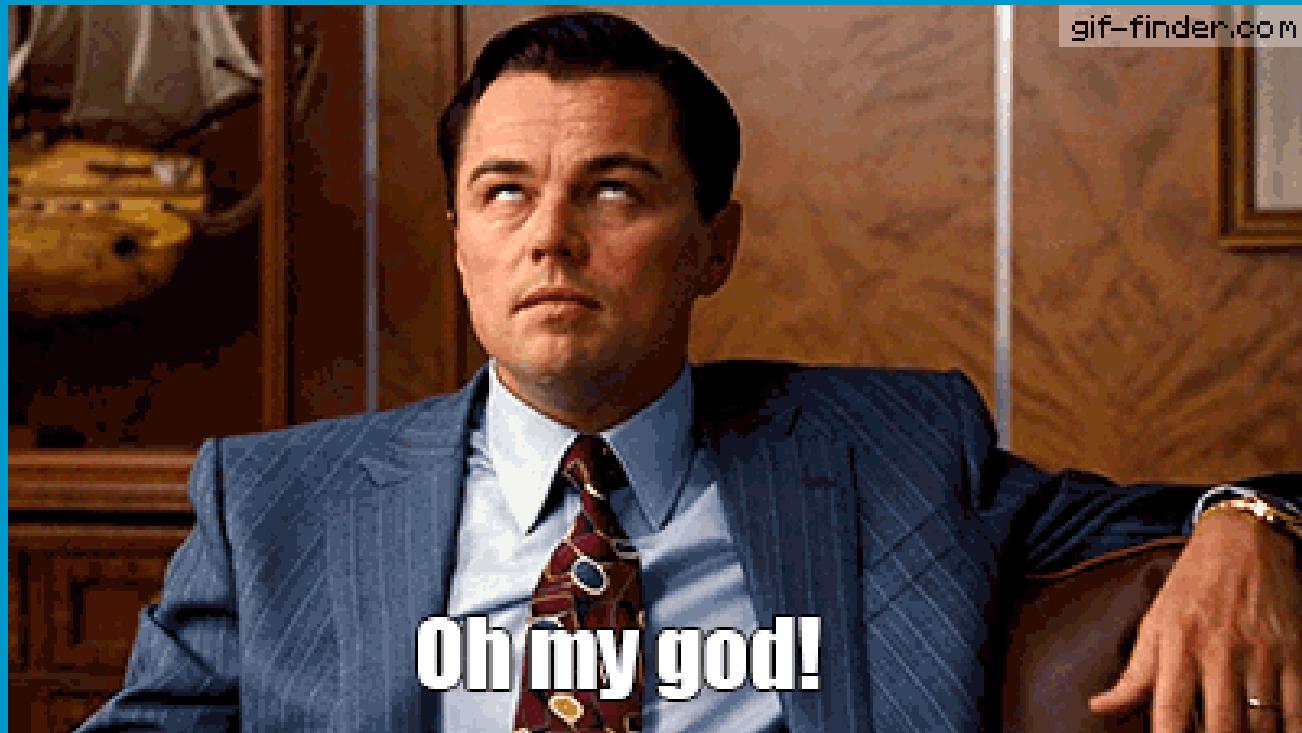
onde :

$$\mathbf{W} \in \mathcal{R}^{oxm}, \mathbf{x}_i \in \mathcal{R}^{mx1}, i = 1, \dots, N,$$

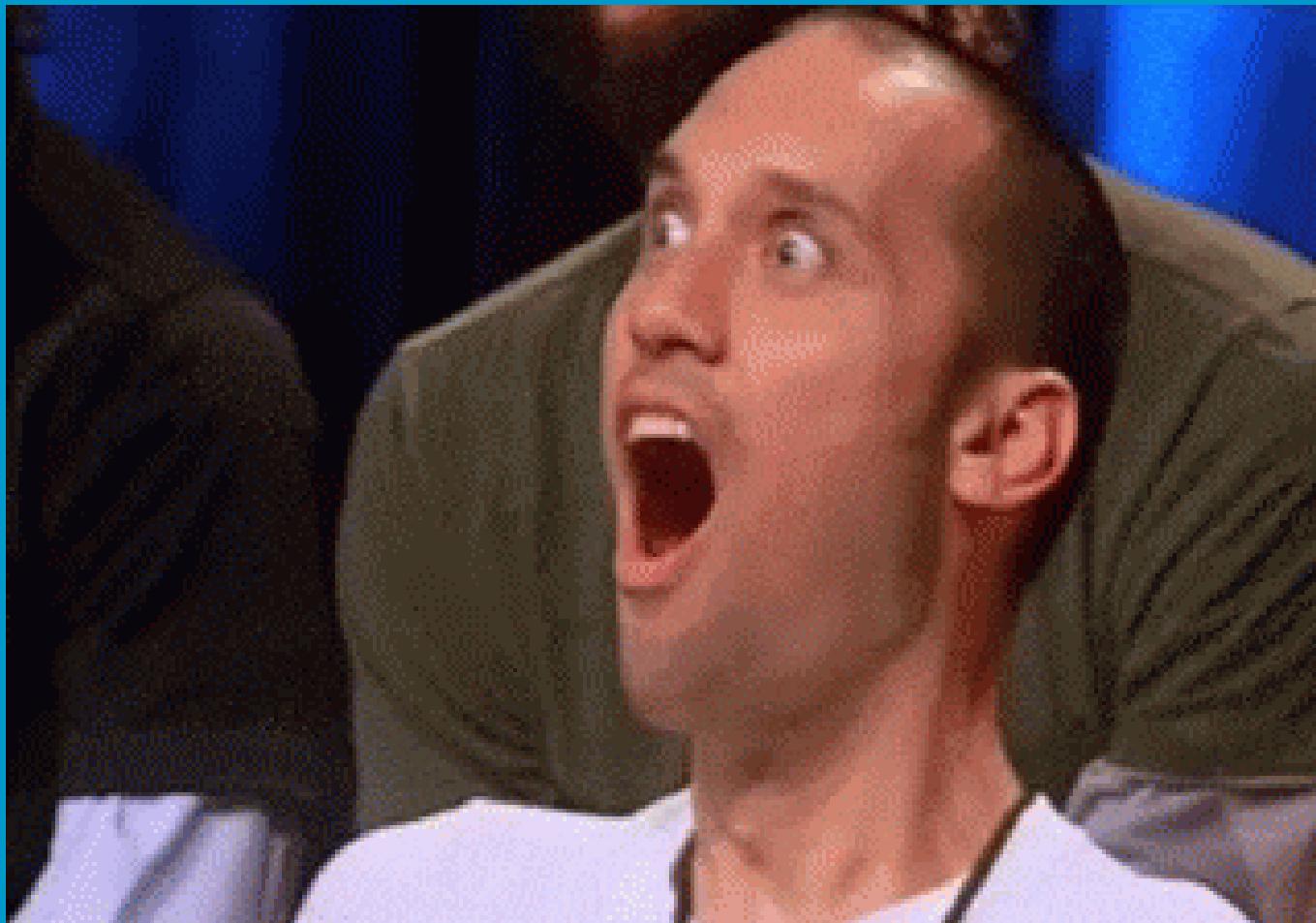
$$\mathbf{e}_i \in \mathcal{R}^{ox1}, \text{ e } \mathbf{b} \in \mathcal{R}^{ox1}$$



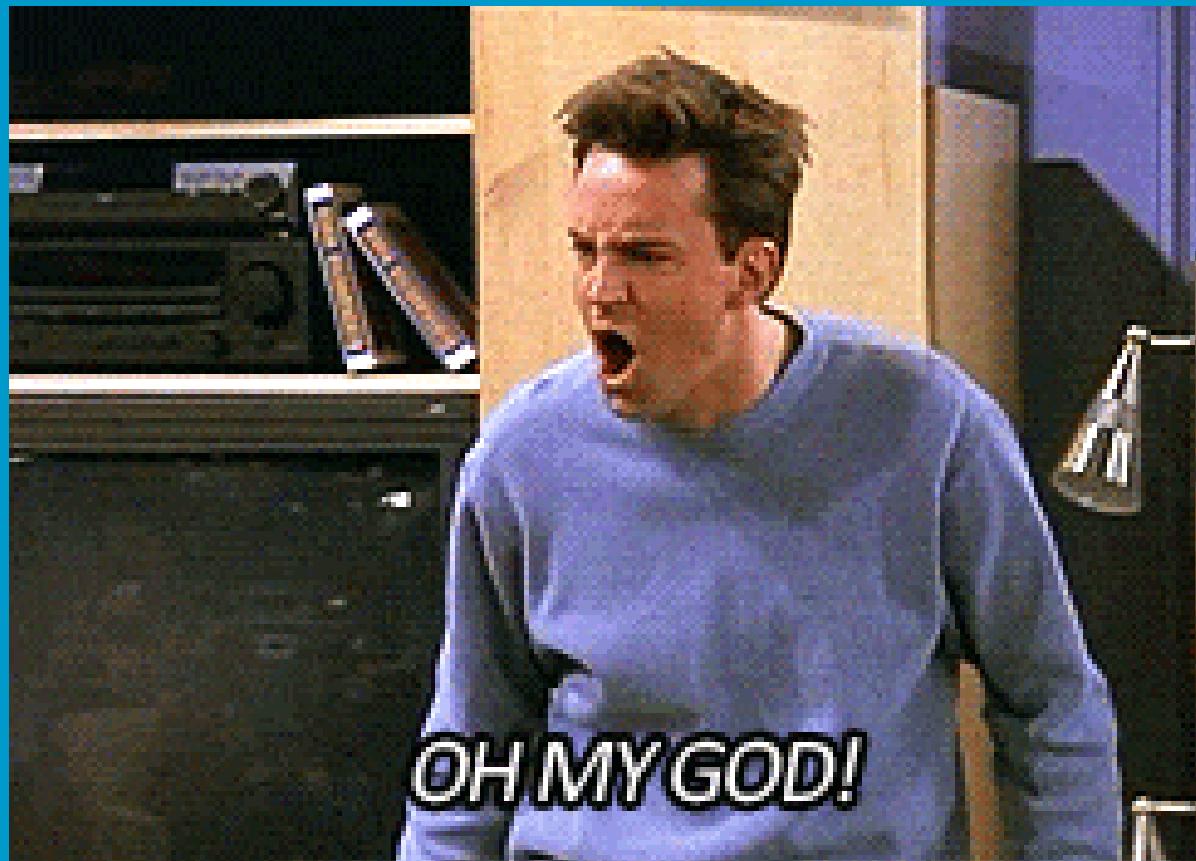
Nesse momento temos...



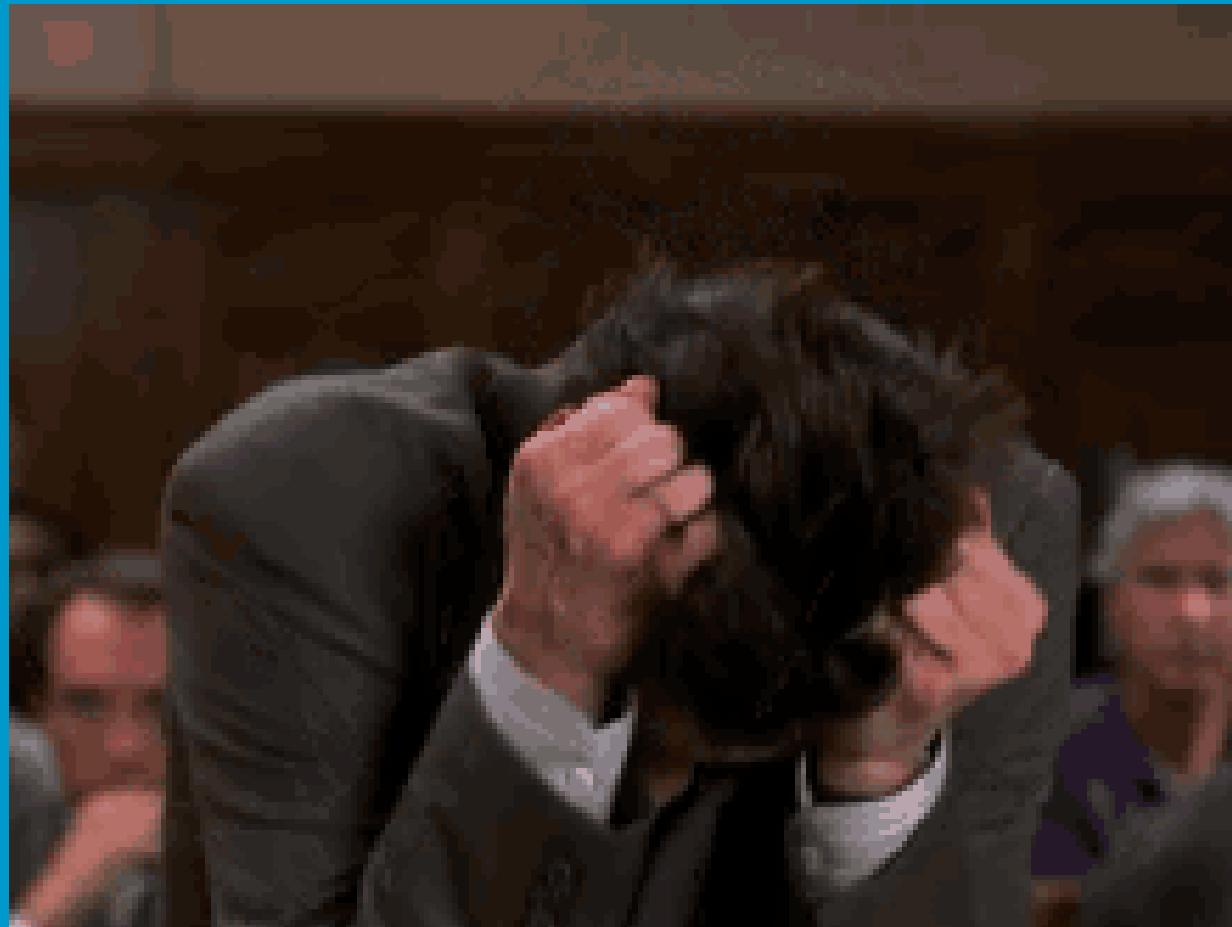
Nesse momento temos...

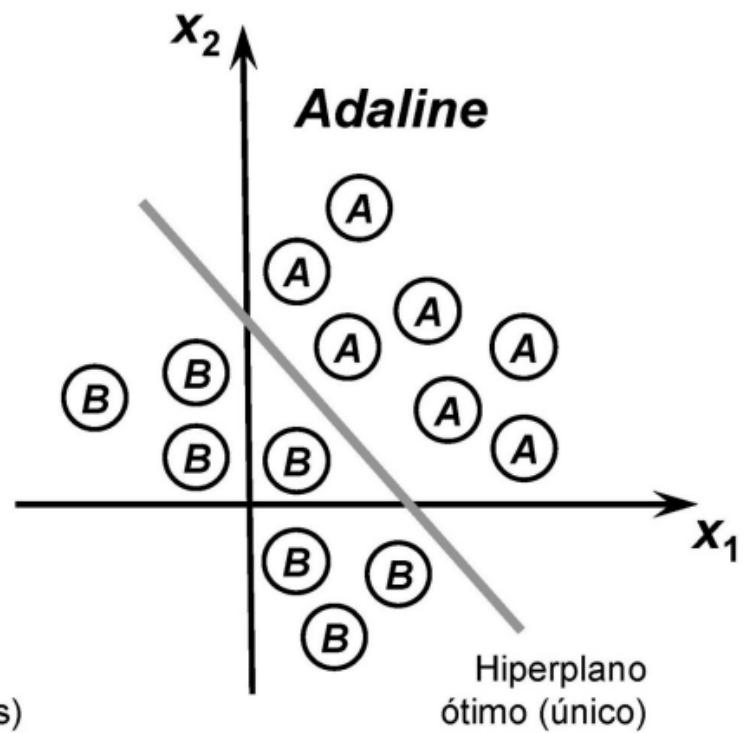
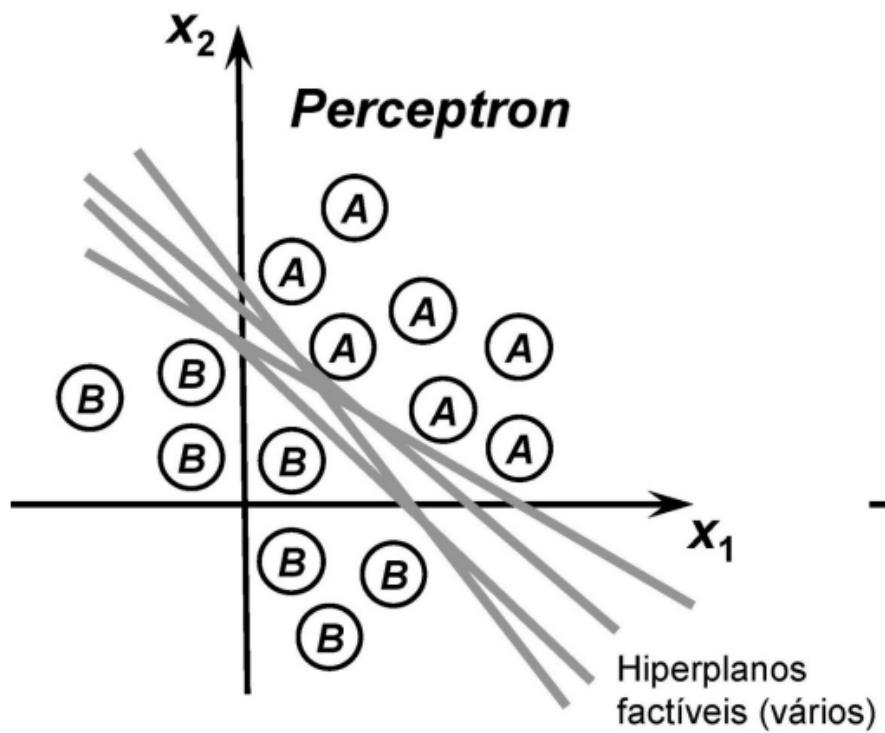


Nesse momento temos...



Nesse momento temos...





Implementação

Kit de sobrevivência...



Kit de sobrevivência...



Ubuntu 16.04



Mr. Turing
VR Education



Notebook

Um interface web iterative que combina códigos, equações, textos e visualizações



<http://www.jupyter.org>

Um shell iterative aberto no browser

Conhecido como Jupyter Notebook ou IPython Notebook

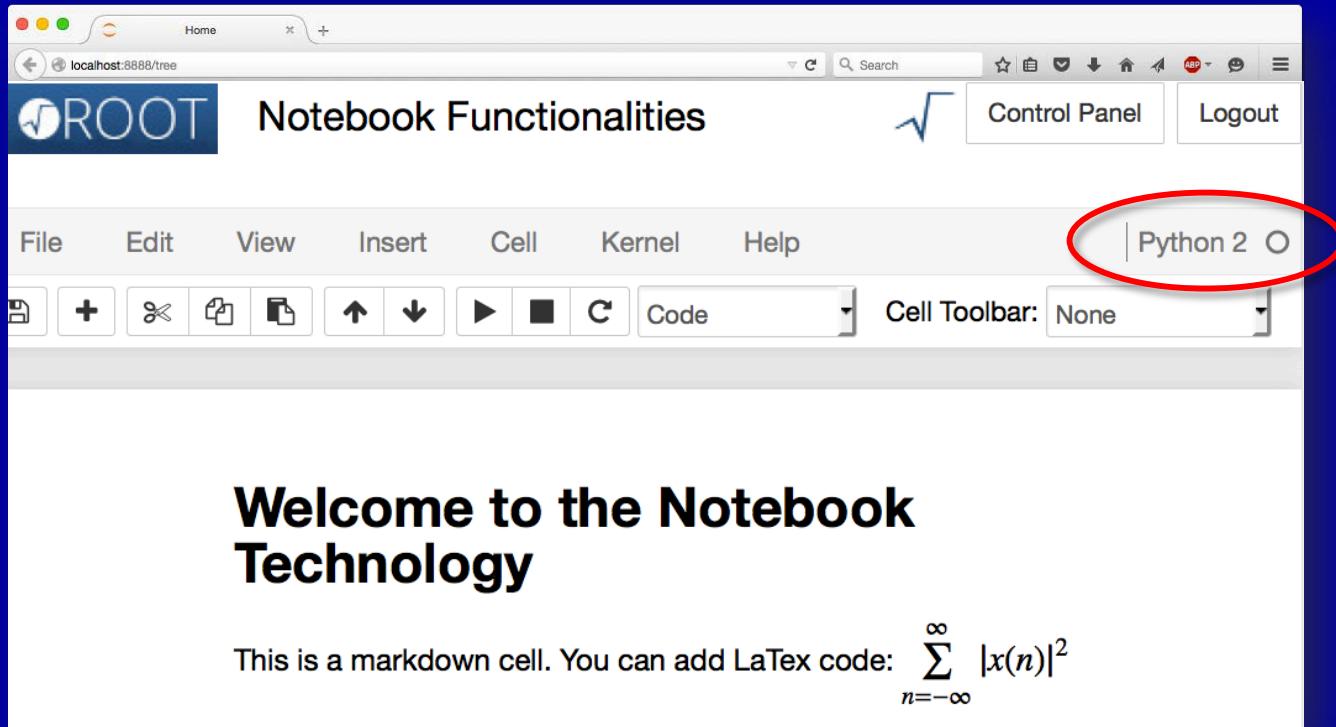
Suporta diversas linguagens: Python, Haskell, Julia, R ...

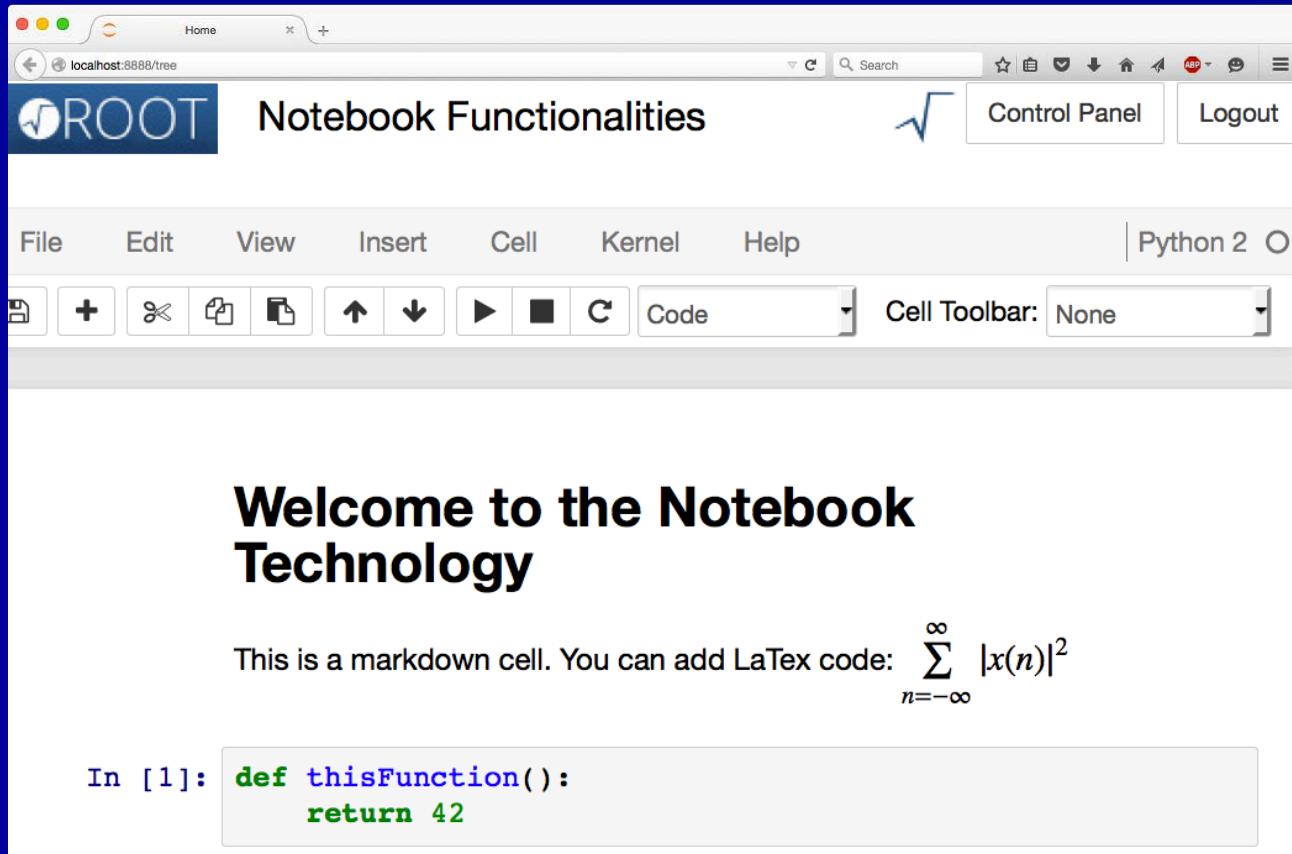
Sem desculpas para compreender!

A screenshot of a web browser displaying the CERNBox interface. The title bar shows the URL `localhost:8888/tree`. The header includes the ROOT logo, the CERNBox logo, and links for Terminal, Control Panel, and Logout. A green callout bubble in the top right corner says "In a browser". The main area has tabs for Files, Running, and Clusters, with "Files" selected. A message says "Select items to perform actions on them." Below is a file tree:

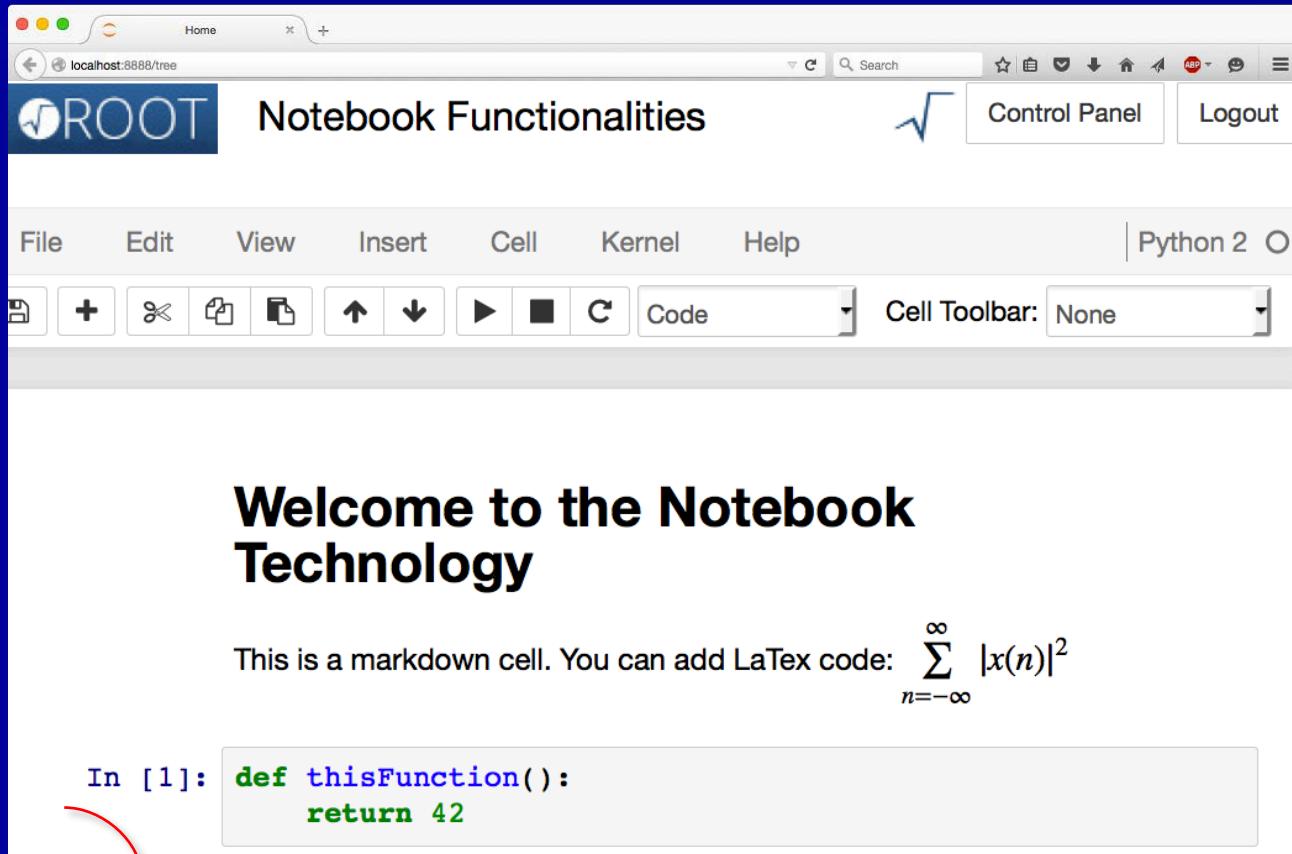
- PresentationNotebooks
- cernbox
- HowTo_ROOT-Notebooks.ipynb
- HowTo_ROOT-Notebooks_Long.ipynb
- My First Notebook.ipynb
- Untitled.ipynb

An arrow points from a green callout bubble labeled "A Choice of Kernels" to the "My First Notebook.ipynb" item in the file tree. A context menu is open on the right side of the screen, listing options: Upload, New, Text File, Folder, Terminal, Notebooks, Python 2, Python 3, and ROOT Prompt.





Código



Isto é um notebook em Python

Código

The screenshot shows a Jupyter Notebook interface running on a local host. The title bar reads "Notebook Functionalities". The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. A toolbar below the menu bar contains icons for file operations like save, new, and delete, along with navigation and cell execution buttons. The main content area displays a large bold heading "Welcome to the Notebook Technology". Below it, a text cell states "This is a markdown cell. You can add LaTex code:" followed by a mathematical expression
$$\sum_{n=-\infty}^{\infty} |x(n)|^2$$
. Two code cells are shown: In [1] contains the Python code

```
def thisFunction():
    return 42
```

, and In [2] shows the result of executing this function: Out[2]: 42. A green button labeled "Código" is overlaid on the bottom right of the notebook window.

localhost:8888/tree

ROOT Notebook Functionalities Control Panel Logout

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code:

$$\sum_{n=-\infty}^{\infty} |x(n)|^2$$

In [1]:

```
def thisFunction():
    return 42
```

In [2]:

```
thisFunction()
```

Out[2]: 42

Código

Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code:

$$\sum_{n=-\infty}^{\infty} |x(n)|^2$$

```
In [1]: def thisFunction():
         return 42
```

```
In [2]: thisFunction()
```

```
Out[2]: 42
```

```
In [3]: %%bash
curl rootaaasdemo.web.cern.ch/rootaaasdemo/SaaSFee.jpg \
> SF.jpg
```

Invocando comandos para o shell do SO...

Comandos Shell

Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code:
$$\sum_{n=-\infty}^{\infty} |x(n)|^2$$

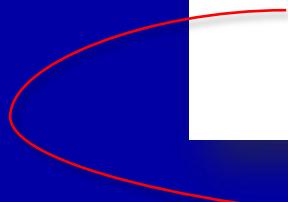
```
In [1]: def thisFunction():
    return 42
```

```
In [2]: thisFunction()
```

```
Out[2]: 42
```

```
In [3]: %%bash
curl roottaasdemo.web.cern.ch/roottaasdemo/SaaSFee.jpg \
> SF.jpg
```

% Total Time	% Received Time	% Xferd Current	Average Speed Dload	Speed Upload	Time Total
Spent	Left	Speed			
100	128k	100	128k	0	0 2731k
---	---	---	---	---	0 ---:---:---
			2787k		



... e captura a saída

93

Comandos shell

Welcome to the Notebook Technology

This is a markdown cell. You can add LaTeX code:
$$\sum_{n=-\infty}^{\infty} |x(n)|^2$$

```
In [1]: def thisFunction():
    return 42
```

```
In [2]: thisFunction()
```

```
Out[2]: 42
```

```
In [3]: %%bash
curl rootaaasdemo.web.cern.ch/rootaaasdemo/SaaSFee.jpg \
> SF.jpg
```

```
% Total    % Received % Xferd  Average Speed   Time
Time      Time     Current
                                         Dload  Upload   Total
Spent     Left   Speed
100  128k  100  128k    0       0  2731k       0 --::--::--
--::--::-- --::--::-- 2787k
```

```
In [4]: from IPython.display import Image
Image(filename="./SF.jpg",width=225)
```

Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code:
$$\sum_{n=-\infty}^{\infty} |x(n)|^2$$

```
In [1]: def thisFunction():
    return 42
```

```
In [2]: thisFunction()
```

```
Out[2]: 42
```

```
In [3]: %%bash
curl roottaasdemo.web.cern.ch/roottaasdemo/SaasFee.jpg \
> SF.jpg
```

```
% Total      % Received % Xferd  Average Speed   Time
Time      Time     Current
                                         Dload  Upload   Total
Spent      Left   Speed
100  128k  100  128k    0       0  2731k        0  --::--::--
--::--::-- --::--::-- 2787k
```

```
In [4]: from IPython.display import Image
Image(filename=".//SF.jpg",width=225)
```

```
Out[4]:
```



http://

Imagenes

Em um Browser

```
function():
    42
```

```
In [2]: thisFunction()
```

```
Out[2]: 42
```

Código

```
curl -O https://saasdemo.web.cern.ch/root/aasdemo/SaasFee.jpg \
```

```
> SF.jpg
```

```
% Total      % Received % Xferd  Average Speed   Time  
Time      Time     Current  
  
          Spent      Left  Speed  
100  128k  100  128k    0  0  2781K  0  --::--  
--::-- --::-- 2787k
```

```
In [4]: from IPython.display import Image
Image(filename=".//SF.jpg",width=225)
```

```
Out[4]:
```



Textos e
fórmulas

Comandos Shell

Imagens

O que é TensorFlow

- Biblioteca Open source para computação numérica usando data flow graphs
- Originalmente desenvolvido pelo time Google Brain Team para pesquisas de machine learning e deep learning

Comparação de frameworks de deep learning na Wikipedia

Software	Creator	Software license ^[a]	Open source	Platform	Written in	Interface	OpenMP support	OpenCL support	CUDA support	Automatic differentiation ^[1]	Has pretrained models	Recurrent nets	Convolutional nets	RBM/DBNs	Parallel execution (multi node)
Apache Singa	Apache Incubator	Apache 2.0	Yes	Linux, Mac OS X, Windows	C++	Python, C++, Java	No	Yes	Yes	?	Yes	Yes	Yes	Yes	Yes
CNTK	Microsoft Research	MIT license ^[2]	Yes	Windows, Linux ^[3] (OSX via Docker on roadmap)	C++	Python, C++, Command line, ^[4] BrainScript ^[5] (.NET on roadmap ^[6])	Yes ^[7]	No	Yes	Yes	Yes ^[8]	Yes ^[9]	Yes ^[9]	No ^[10]	Yes ^[11]
Deeplearning4j	Skymind engineering team; Deeplearning4j community; originally Adam Gibson	Apache 2.0	Yes	Linux, Mac OS X, Windows, Android (Cross-platform)	C, C++	Java, Scala, Clojure, Python (Keras)	Yes	On roadmap ^[12]	Yes ^[13]	Computational Graph	Yes ^[14]	Yes	Yes	Yes	Yes ^[15]
Dlib	Davis King	Boost Software License	Yes	Cross-Platform	C++	C++	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Keras	François Chollet	MIT license	Yes	Linux, Mac OS X, Windows	Python	Python	Only if using Theano as backend	Under development for the Theano backend (and on roadmap for the TensorFlow backend)	Yes	Yes	Yes ^[16]	Yes	Yes	Yes	Yes ^[17]
MXNet	Distributed (Deep) Machine Learning Community	Apache 2.0	Yes	Linux, Mac OS X, Windows, AWS, Android, iOS, JavaScript ^[21]	Small C++ core library	C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl	Yes	On roadmap ^[22]	Yes	Yes ^[23]	Yes ^[24]	Yes	Yes	Yes	Yes ^[25]
Neural Designer	Artelnics	Proprietary	No	Linux, Mac OS X, Windows	C++	Graphical user interface	Yes	No	No	?	?	No	No	No	?
OpenNN	Artelnics	GNU LGPL	Yes	Cross-platform	C++	C++	Yes	No	No	?	?	No	No	No	?
TensorFlow	Google Brain team	Apache 2.0	Yes	Linux, Mac OS X, Windows ^[26]	C++, Python	Python, (C/C++ public API only for executing graphs ^[27])	No	On roadmap ^{[28][29]}	Yes	Yes ^[30]	Yes ^[31]	Yes	Yes	Yes	Yes
Theano	Université de Montréal	BSD license	Yes	Cross-platform	Python	Python	Yes	Under development ^[32]	Yes	Yes ^{[33][34]}	Through Lasagne's model zoo ^[35]	Yes	Yes	Yes	Yes ^[36]
Torch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	BSD license	Yes	Linux, Mac OS X, Windows, ^[37] Android, ^[38] iOS	C, Lua	Lua, LuaJIT, ^[39] C, utility library for C++/OpenCL ^[40]	Yes	Third party implementations ^{[41][42]}	Yes ^{[43][44]}	Through Twitter's Autograd ^[45]	Yes ^[46]	Yes	Yes	Yes	Yes ^[47]
Wolfram Mathematica	Wolfram Research	Proprietary	No	Windows, Mac OS X, Linux, Cloud computing	C++	Command line, Java, C++	No	Yes	Yes	Yes	Yes ^[48]	Yes	Yes	Yes	Yes

Escolha do TensorFlow

- Python API
- Google open source project no Github
- Novembro de 2015
- Ambientes heterogêneos
- Fácil execução em múltiplas GPUs
- Visualização (TensorBoard)
- Maior comunidade (> 10,000 commits)

Pq Tensorflow?

 tensorflow / tensorflow

 Watch ▾

2,764

 Star

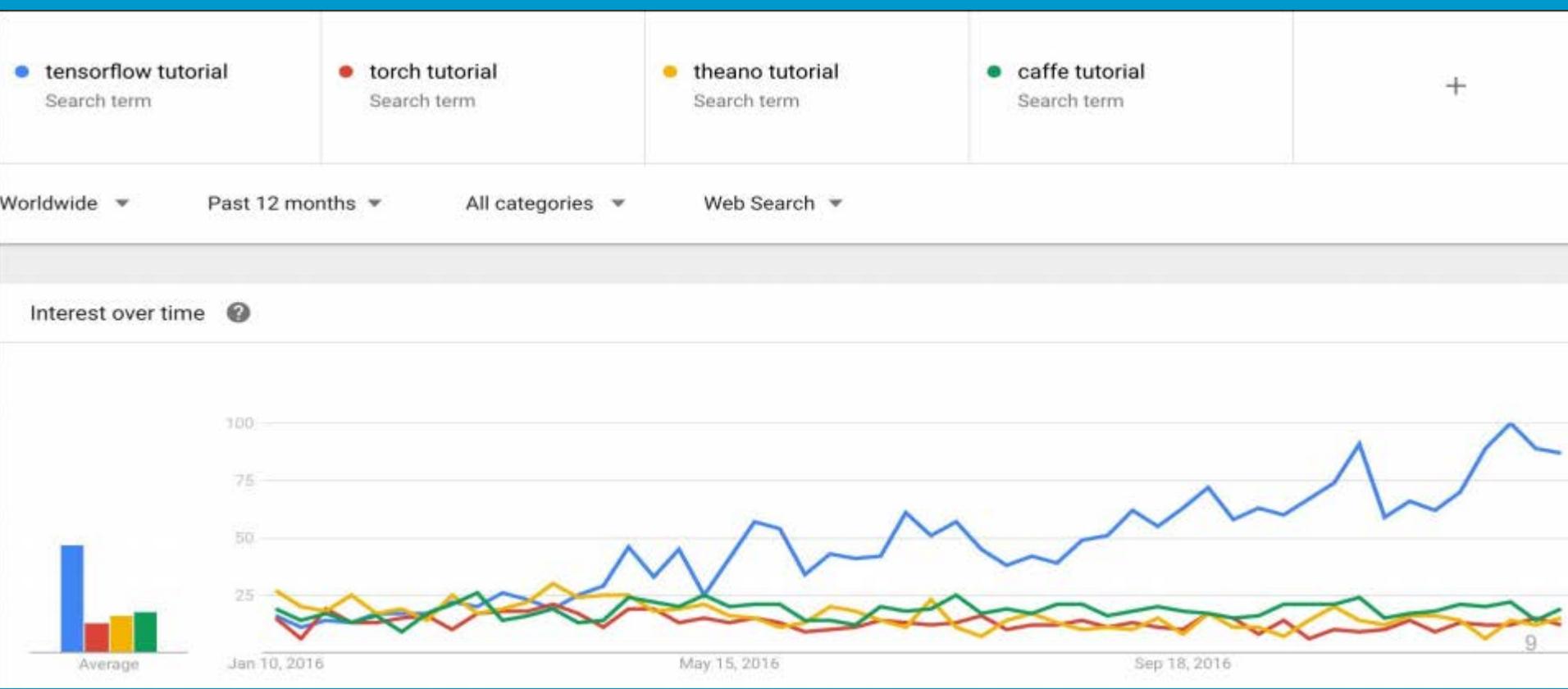
28,873

 Fork

11,895

Gostamos pq é Google?

Pq Tensorflow?



Dataflow Graph

- TensorFlow usa grafo de fluxo de dados unificado para representar a computação de um algoritmo e os estados em que ele opera
- TensorFlow separa a definição de uma estrutura de sua execução

Dataflow Graph

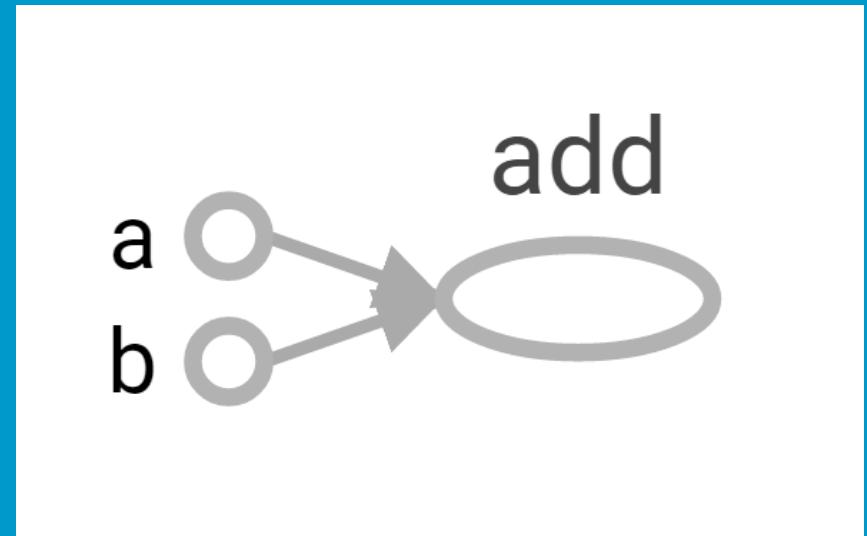
- Os nós no grafo representam operações matemáticas, enquanto as arestas representam vetores multidimensionais de dados (tensors)

Dataflow Graph

```
import tensorflow as tf  
a = tf.constant(2, name='a')  
b = tf.constant(3, name='b')  
A = tf.add(a, b, name =  
'add')
```

Nós: operadores, variáveis e constantes

Arestas: tensors(vetores n-d)

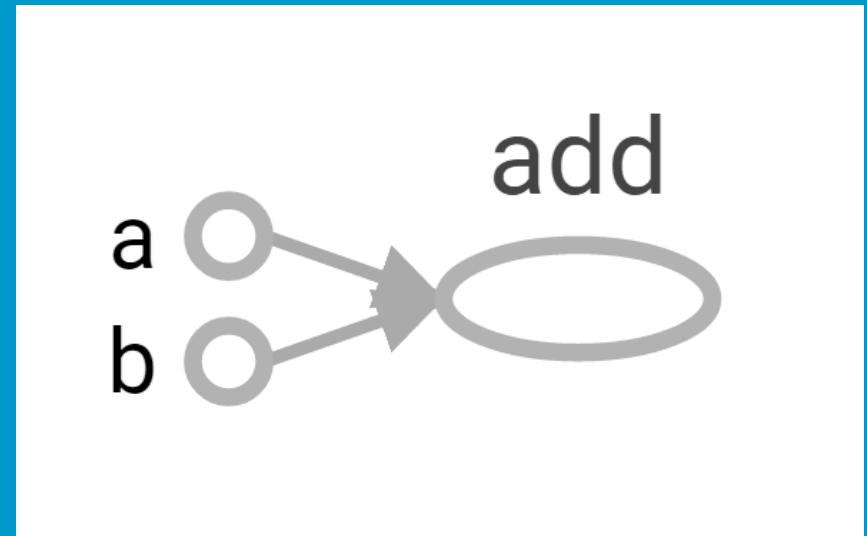


Dataflow Graph

Criar uma sessão,
avalia o grafo para
buscar o valor do nó.

with tf.Session() as
sess:

print sess.run(A)



Dataflow Graph

```
import tensorflow as tf

a = tf.placeholder("float")
b = tf.placeholder("float")
y = tf.mul(a, b)

sess = tf.Session()
print sess.run(
    y, feed_dict={a: 3, b: 3})
```

Dataflow

- tf.add sum
- tf.sub substraction
- tf.mul multiplication
- tf.div division
- tf.mod module
- tf.abs return the absolute value
- tf.neg return negative value
- tf.sign return the sign

Dataflow

- `tf.inv` returns the inverse
- `tf.square` calculates the square
- `tf.round` returns the nearest integer
- `tf.sqrt` calculates the square root
- `tf.pow` calculates the power
- `tf.exp` calculates the exponential
- `tf.log` calculates the logarithm
- `tf.maximum` returns the maximum
- `tf.minimum` returns the minimum

Dataflow

- `tf.minimum` returns the minimum
- `tf.cos` calculates the cosine
- `tf.sin` calculates the sine

Dataflow

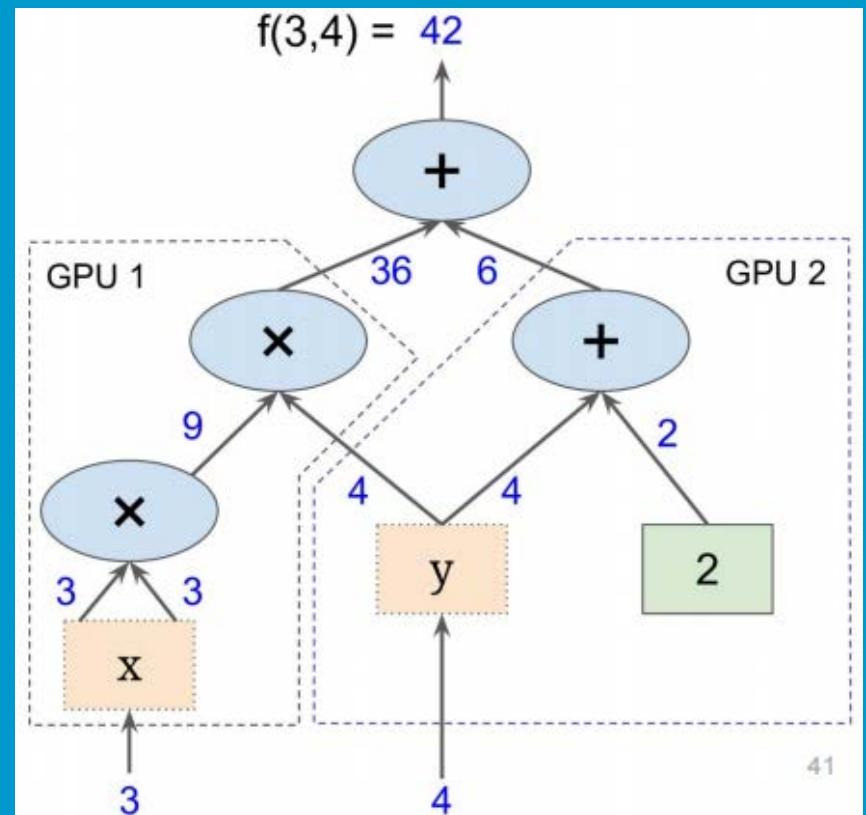
- `tf.diag` returns a diagonal tensor with a given diagonal values
- `tf.transpose` returns the transposes of the argument
- `tf.matmul` returns a tensor product of multiplying two tensors listed as arguments

Dataflow

- `tf.matrix_determinant` returns the determinant of the square matrix specified as an argument
- `tf.matrix_inverse` returns the inverse of the square matrix specified as an argument

Dataflow Graph

Permite quebrar grafos em vários pedaços e executá-los paralelamente em várias CPUs, GPUs ou dispositivos



Como começar?

- Passo 1:
 - pip install tensorflow (para CPU)
 - pip install tensorflow-gpu (com GPU)

Nvidia GPUs



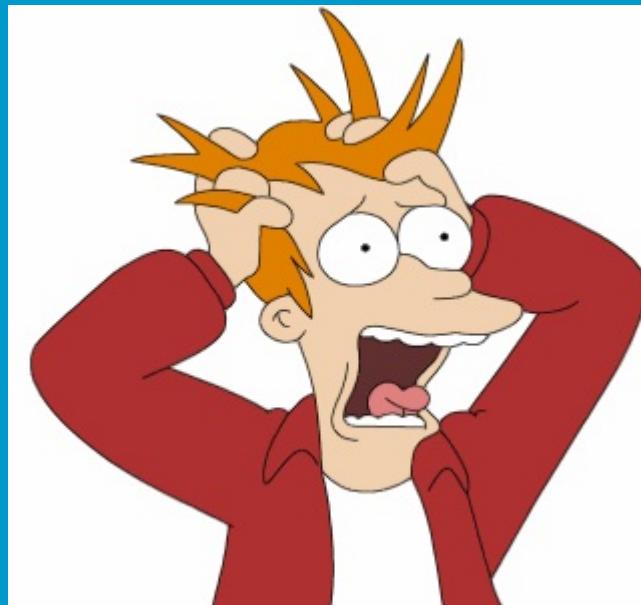
Nvidia GPUs

- GTX 1080 lançada em Junho de 2016 usa a arquitetura Pascal com suporte específico para processamento de redes neurais, 8 GB de memória (320 GB/sec bandwidth) 8.9 TFlops.
- Entre R\$ 3000,00 a R\$ 4000,00
- Outros modelos para iniciantes
 - GTX 650 R\$ 400,00 (GPU Capability 3.0)

Aguardem a lista de 87
exercícios....



- Próxima aula:
 - Como treinar uma rede neural com vários neurônios (Fortes emoções).



Fim

www.deeplearningbrasil.com.br