

# Redes Neurais e Aprendizagem Profunda

## REDES NEURAIS ARTIFICIAIS

### INTRODUÇÃO

---

Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

# Rede Neural Artificial (sem metáfora cognitiva)

(Antes) Função de predição:

$$f = Wx$$

# Rede Neural Artificial (sem metáfora cognitiva)

(Antes) Função de predição:

$$f = Wx$$

(Agora) Rede neural de 2 camadas:

$$f = W_2 \max(0, W_1 x)$$

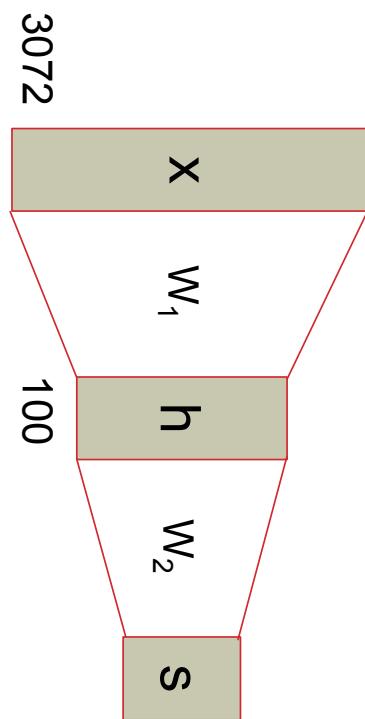
# Rede Neural Artificial (sem metáfora cognitiva)

(Antes) Função de predição:

$$f = Wx$$

(Agora) Rede neural de 2 camadas:

$$f = W_2 \max(0, W_1 x)$$



# Rede Neural Artificial (sem metáfora cognitiva)

(Antes) Função de predição:

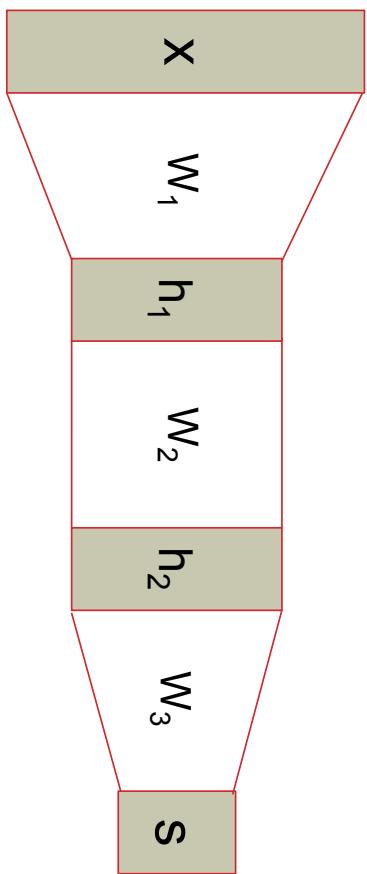
$$f = Wx$$

(Agora) Rede neural de 2 camadas:

$$f = W_2 \max(0, W_1 x)$$

ou Rede neural de 3 camadas:

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$



# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```

# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```



# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```



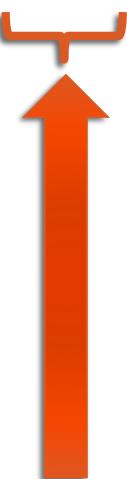
# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```



# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```



# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```



# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```



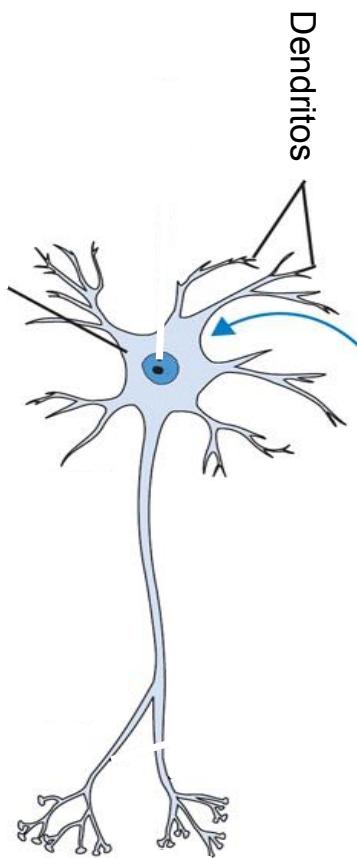
# Código para Treino de Rede Neural de 2 Camadas

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```

Apenas ~20 linhas!

# Inspiração Biológica

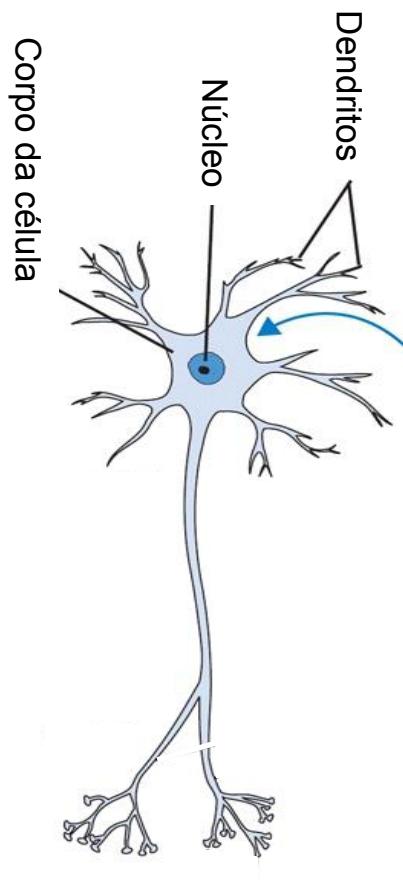
Impulso levado para  
o corpo da célula



Dendritos

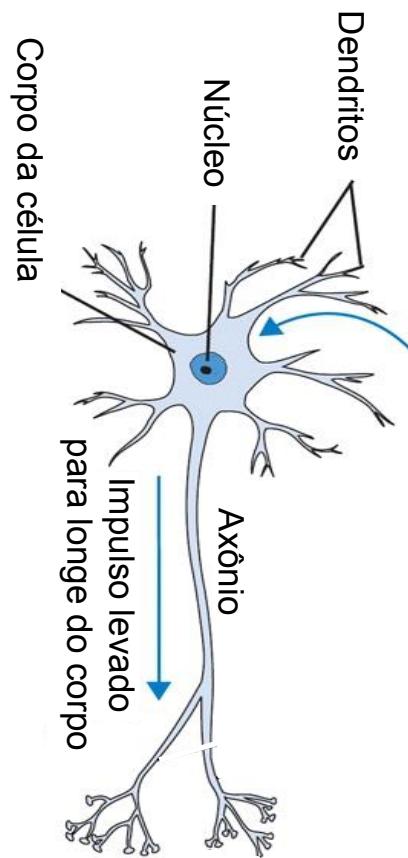
# Inspiração Biológica

Impulso levado para  
o corpo da célula

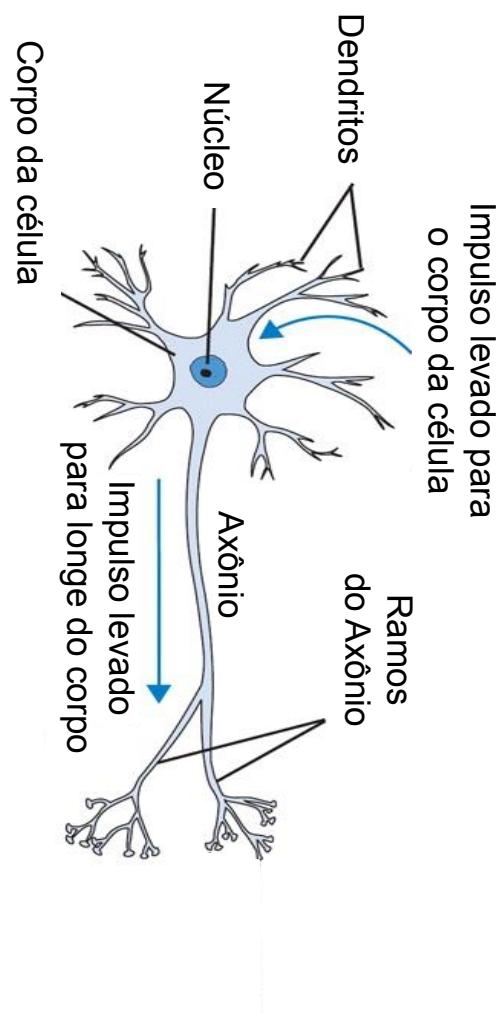


# Inspiração Biológica

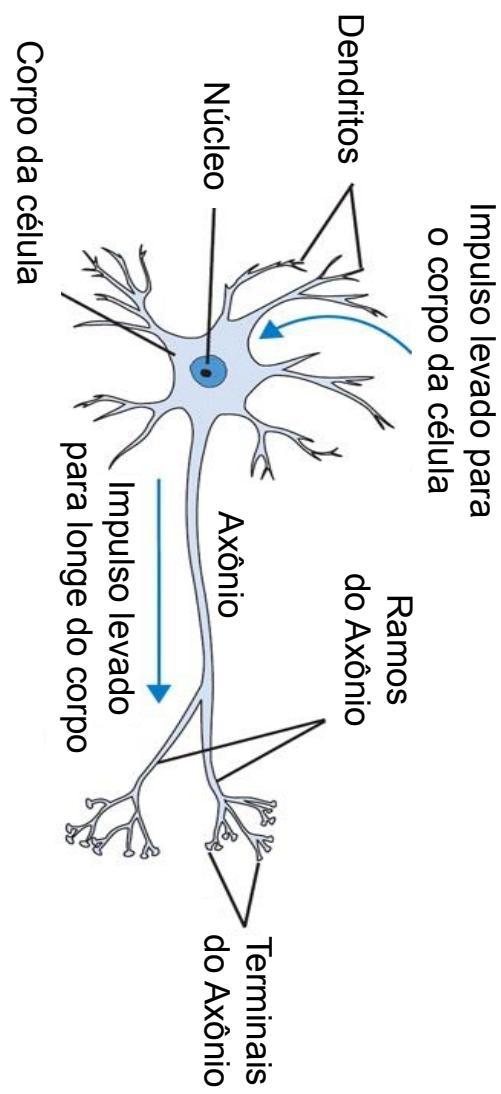
Impulso levado para  
o corpo da célula



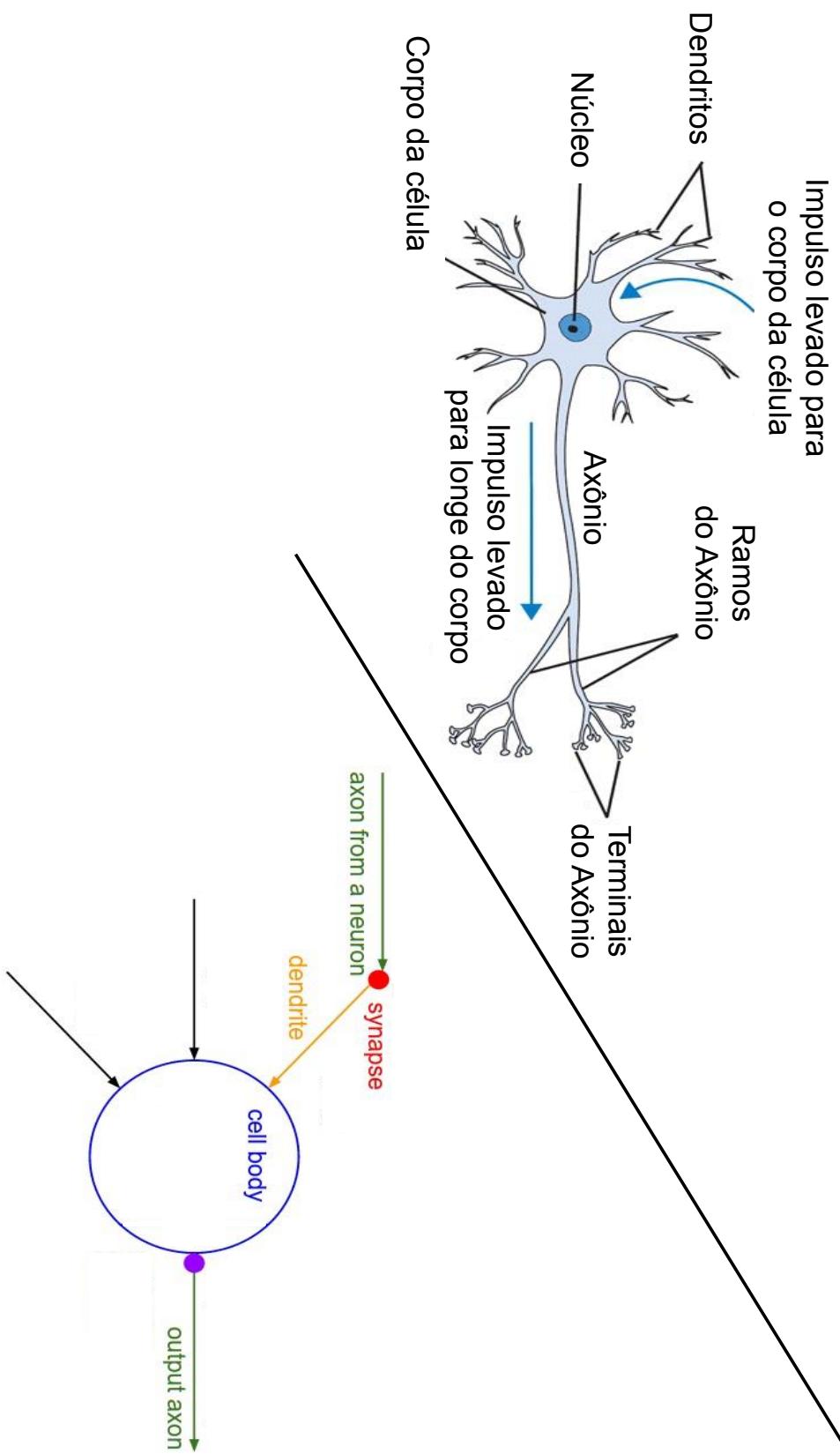
# Inspiração Biológica



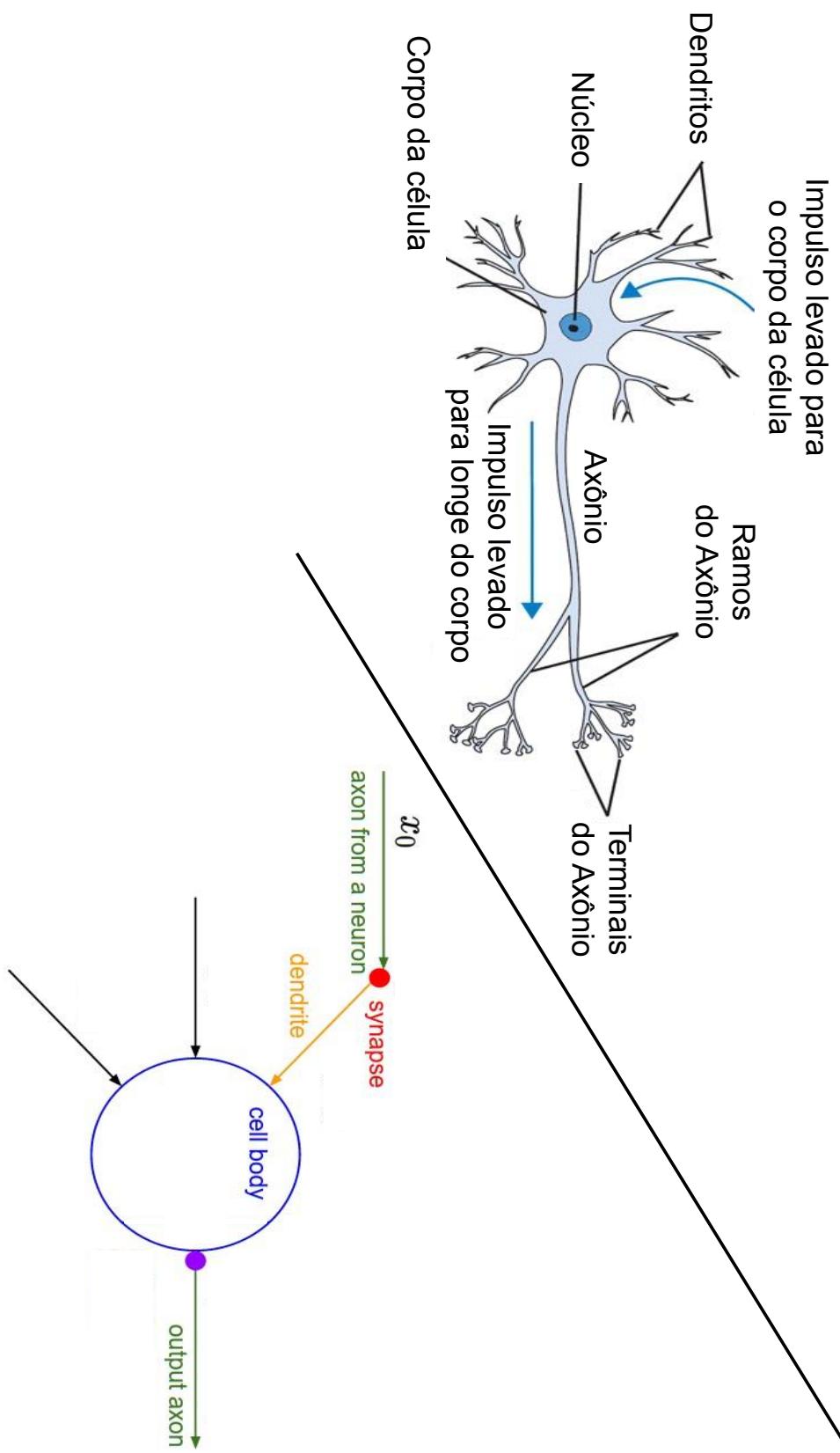
# Inspiração Biológica



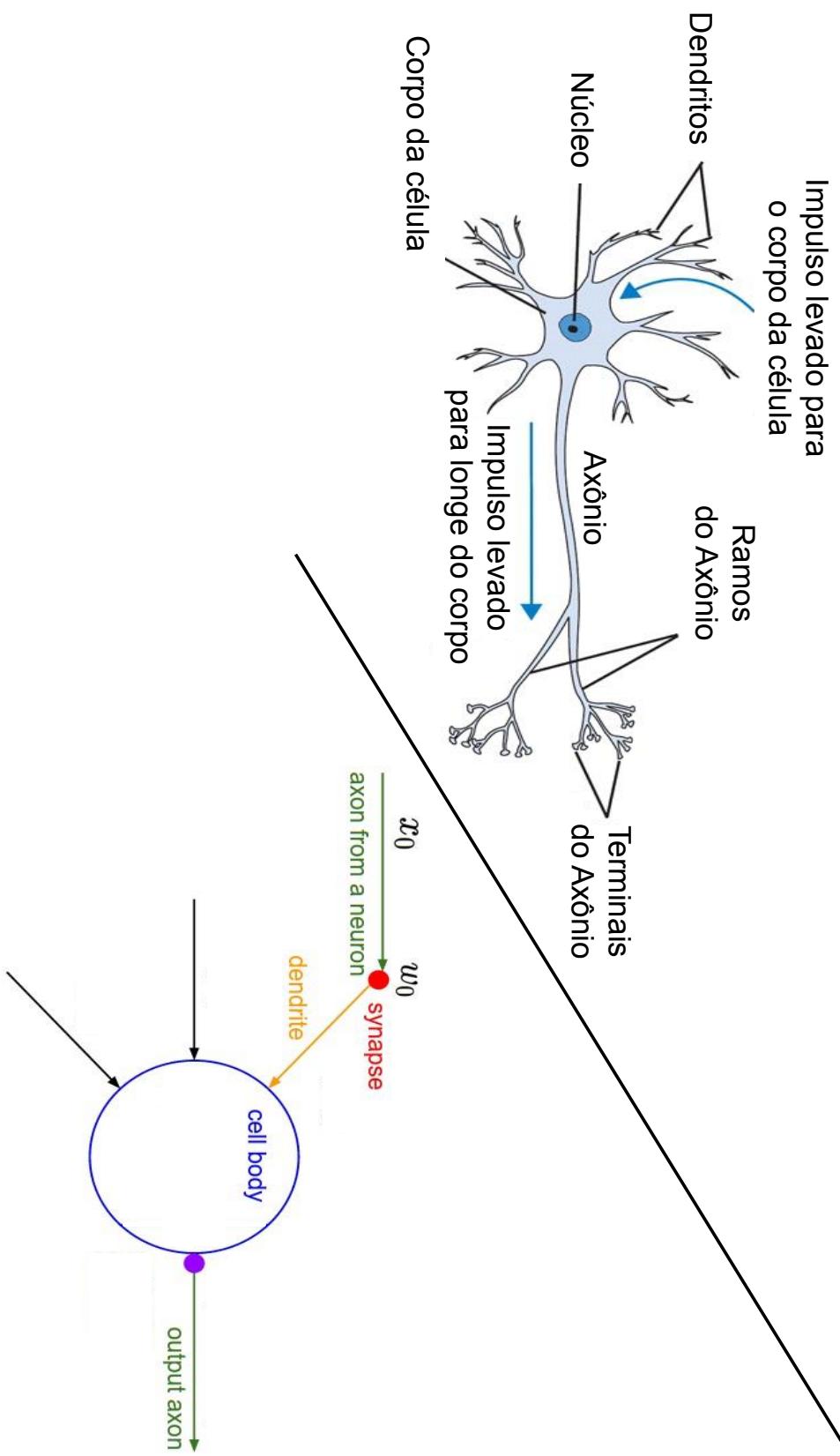
# Inspiração Biológica



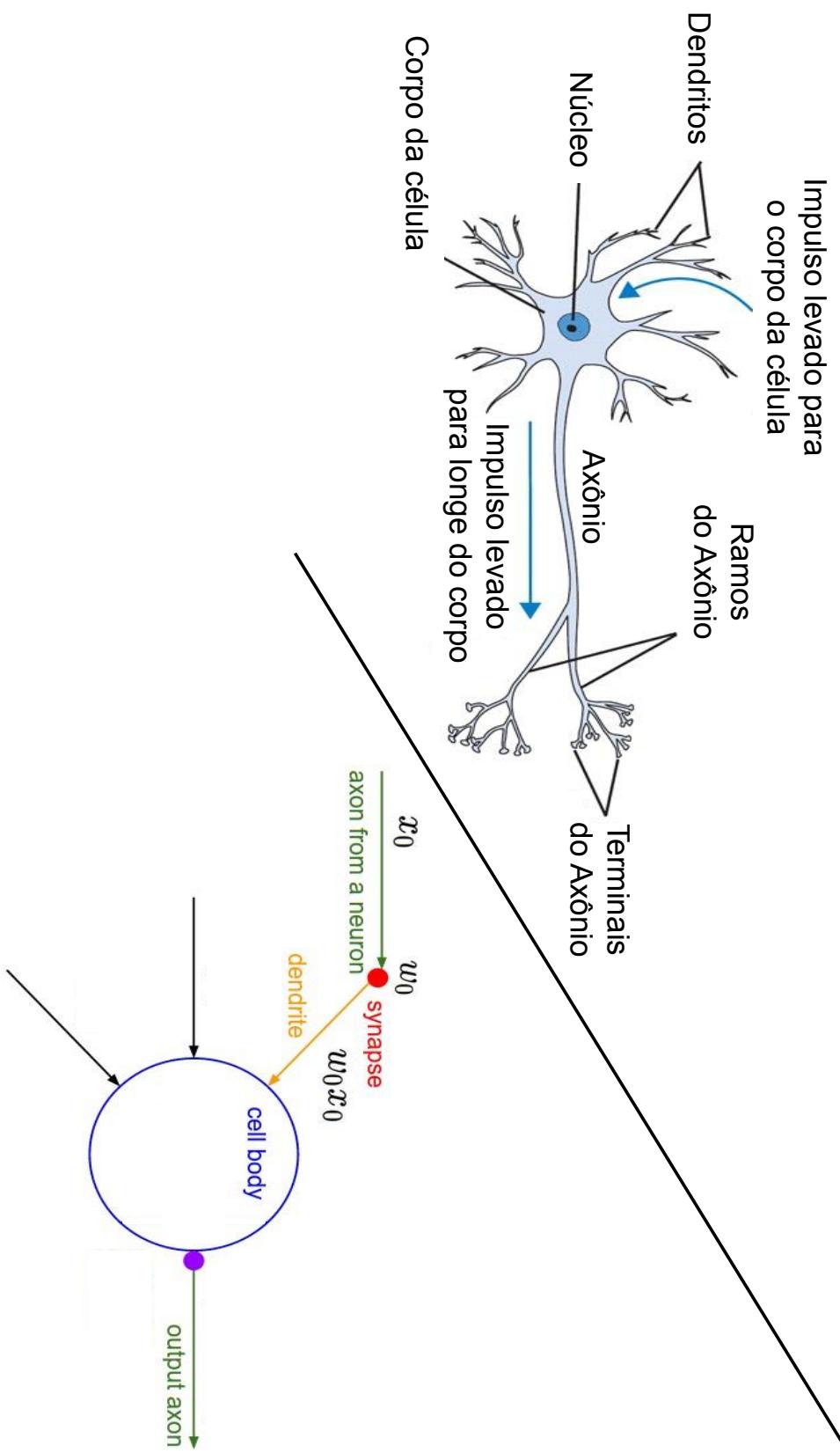
# Inspiração Biológica



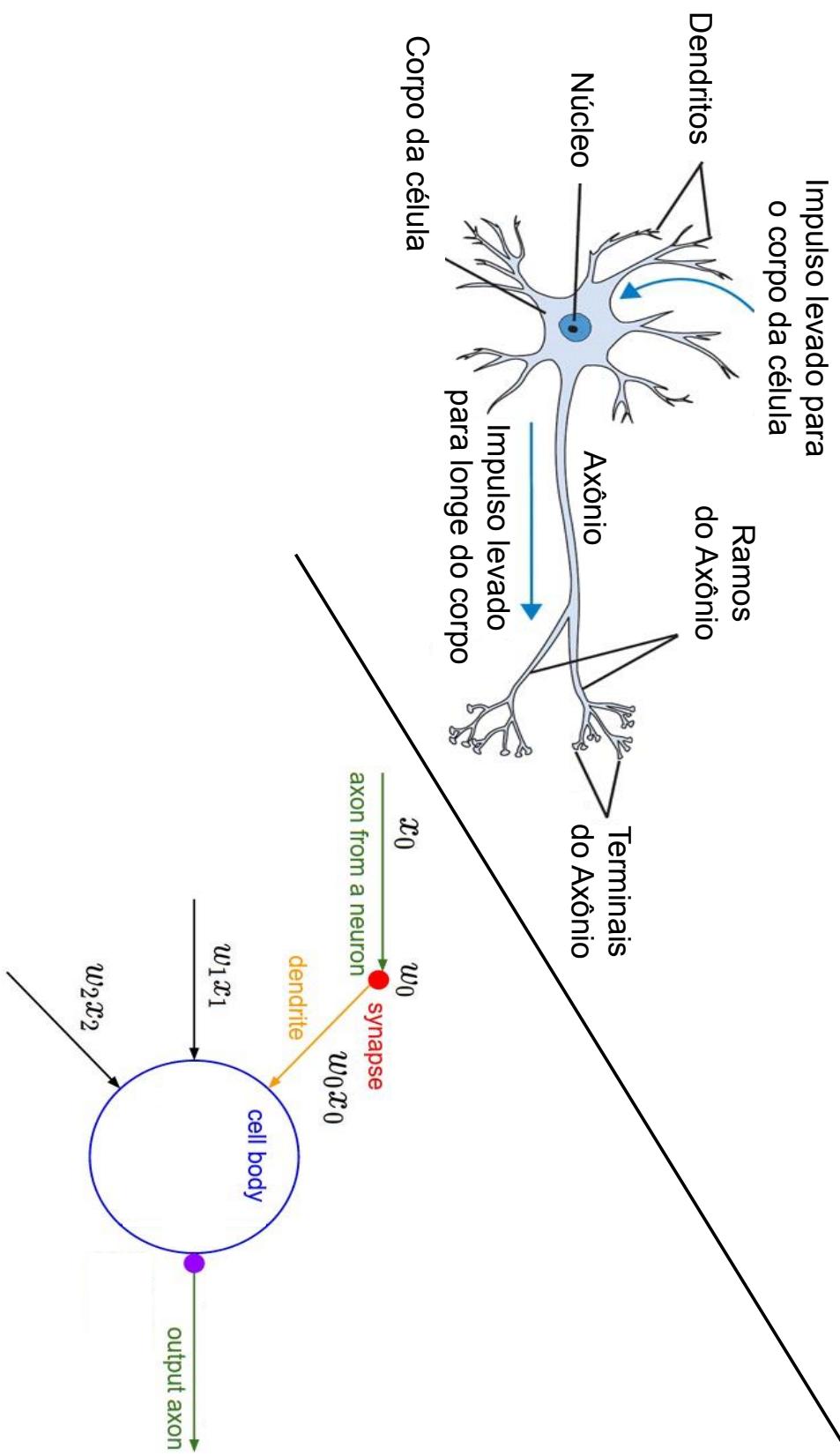
# Inspiração Biológica



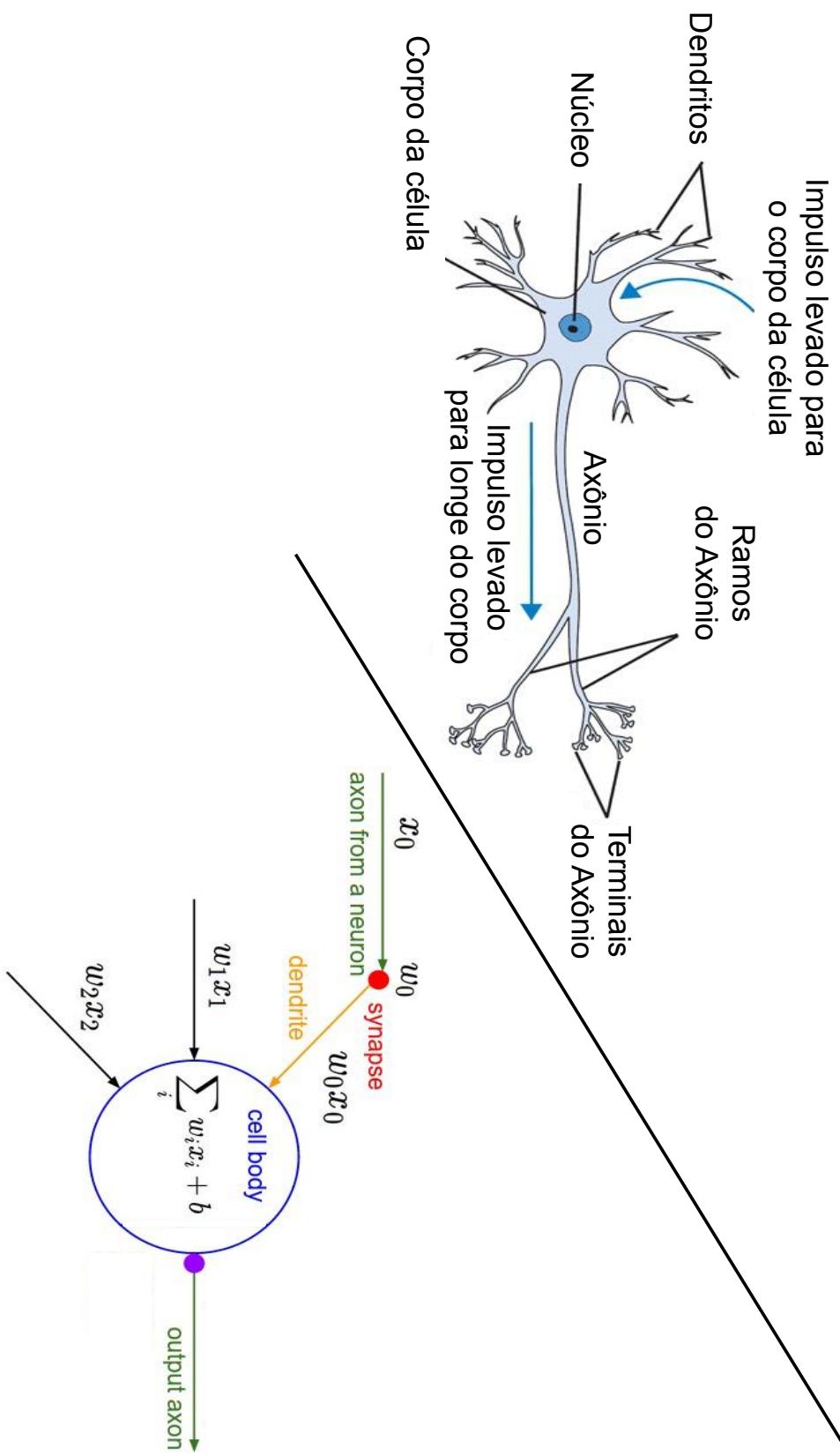
# Inspiração Biológica



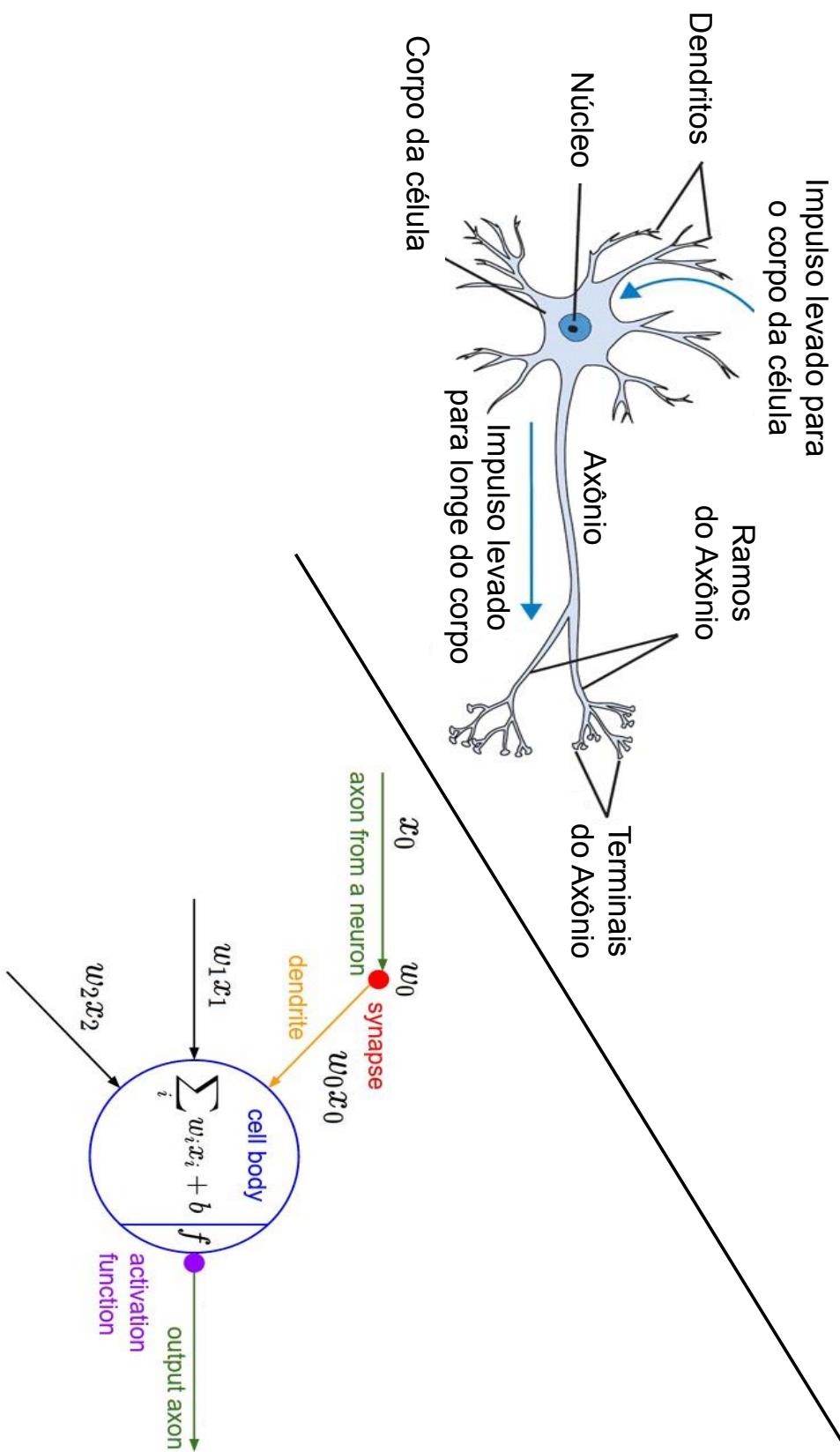
# Inspiração Biológica



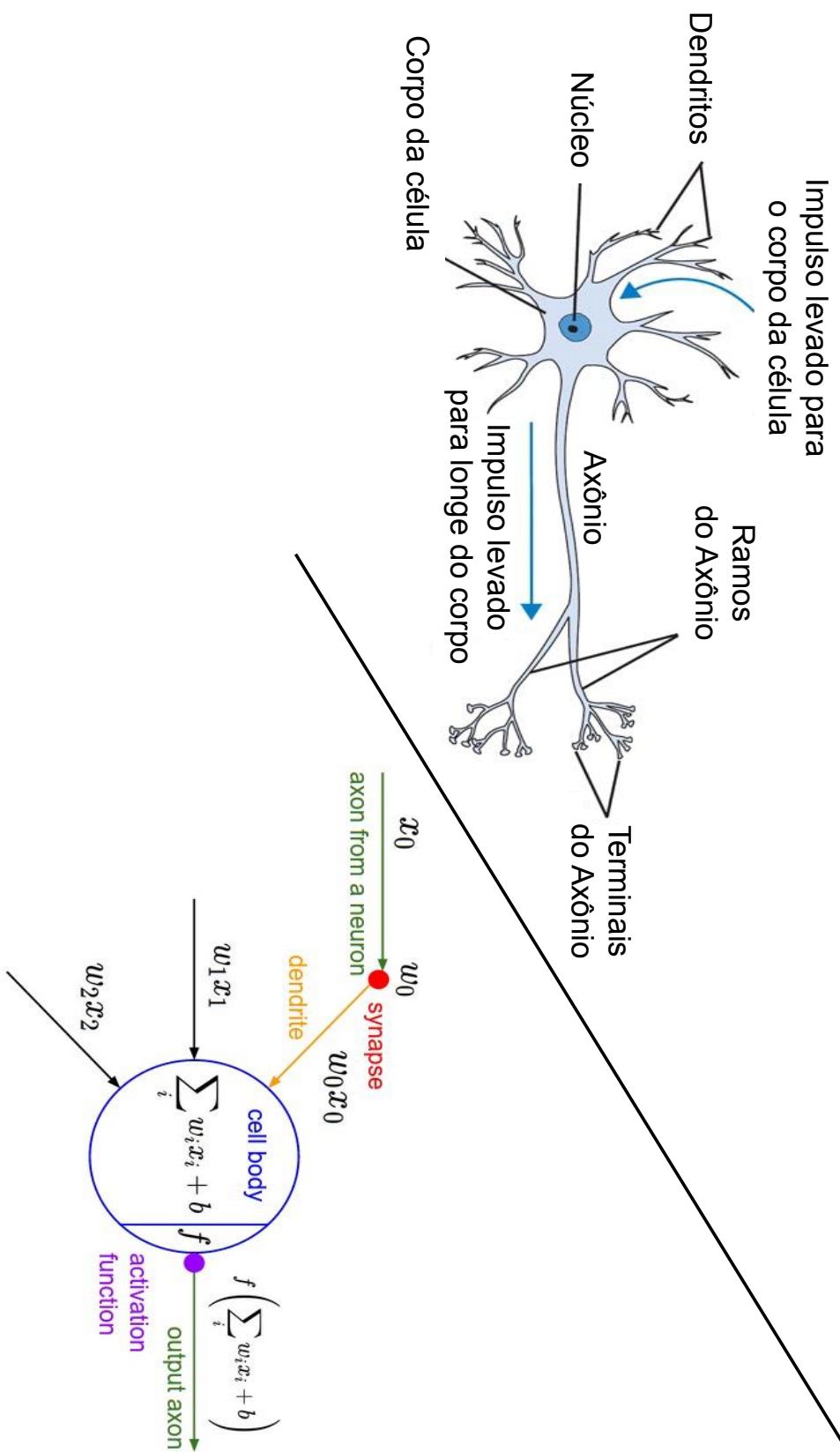
# Inspiração Biológica



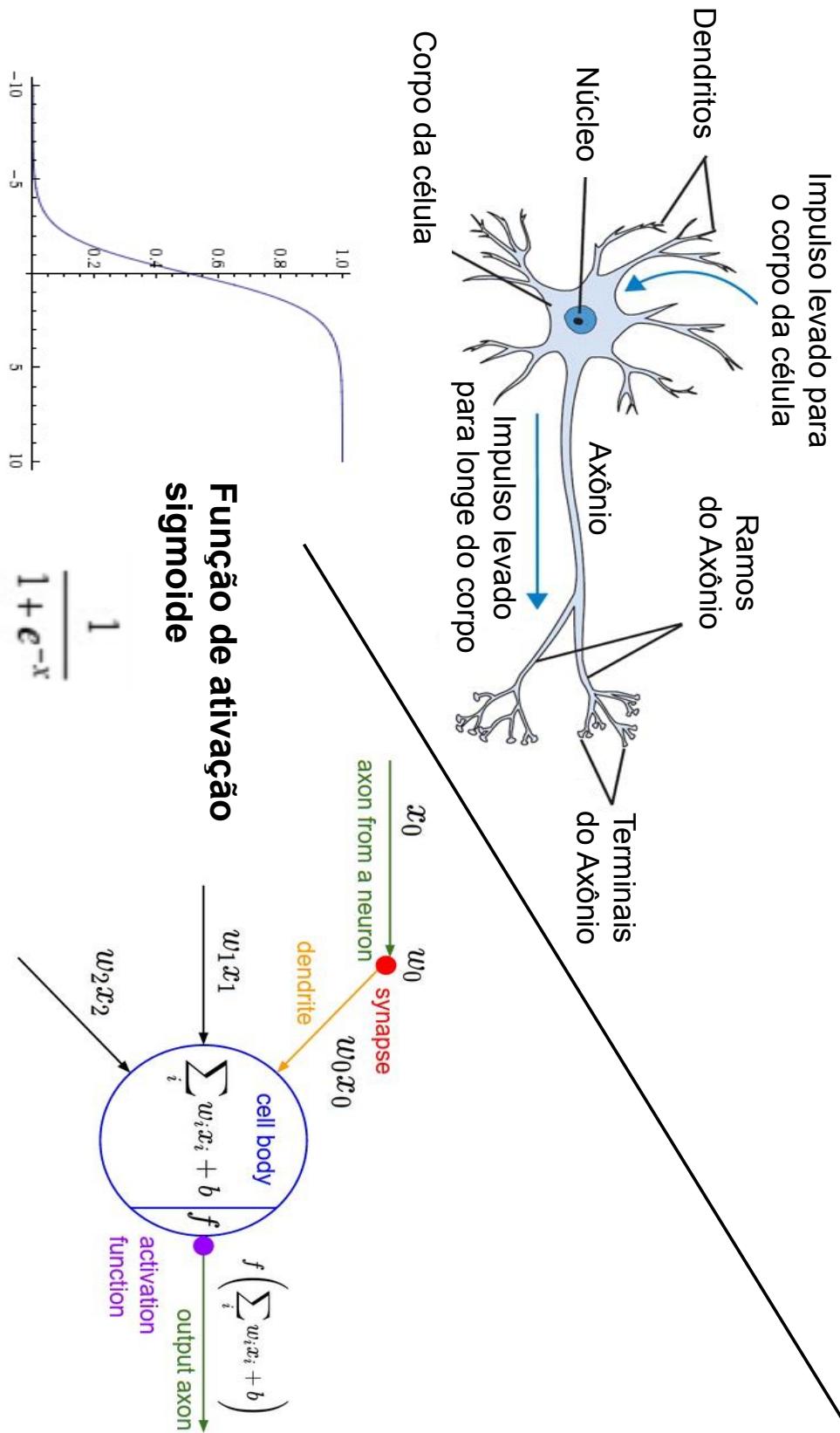
# Inspiração Biológica



# Inspiração Biológica

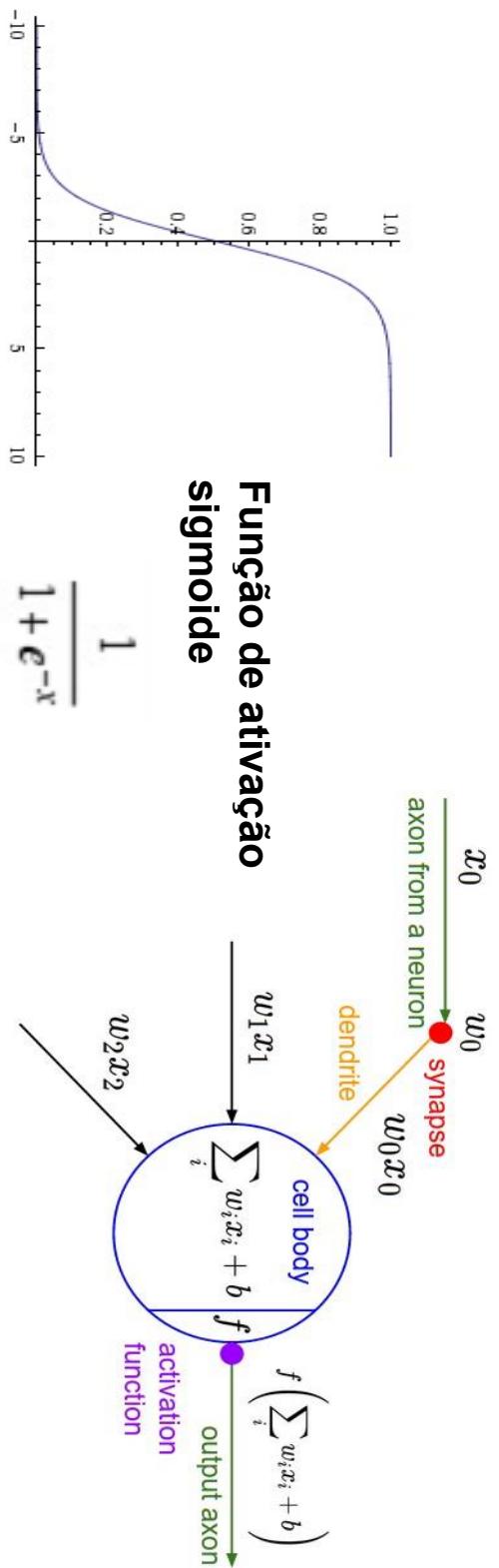


# Inspiração Biológica



# Inspiração Biológica

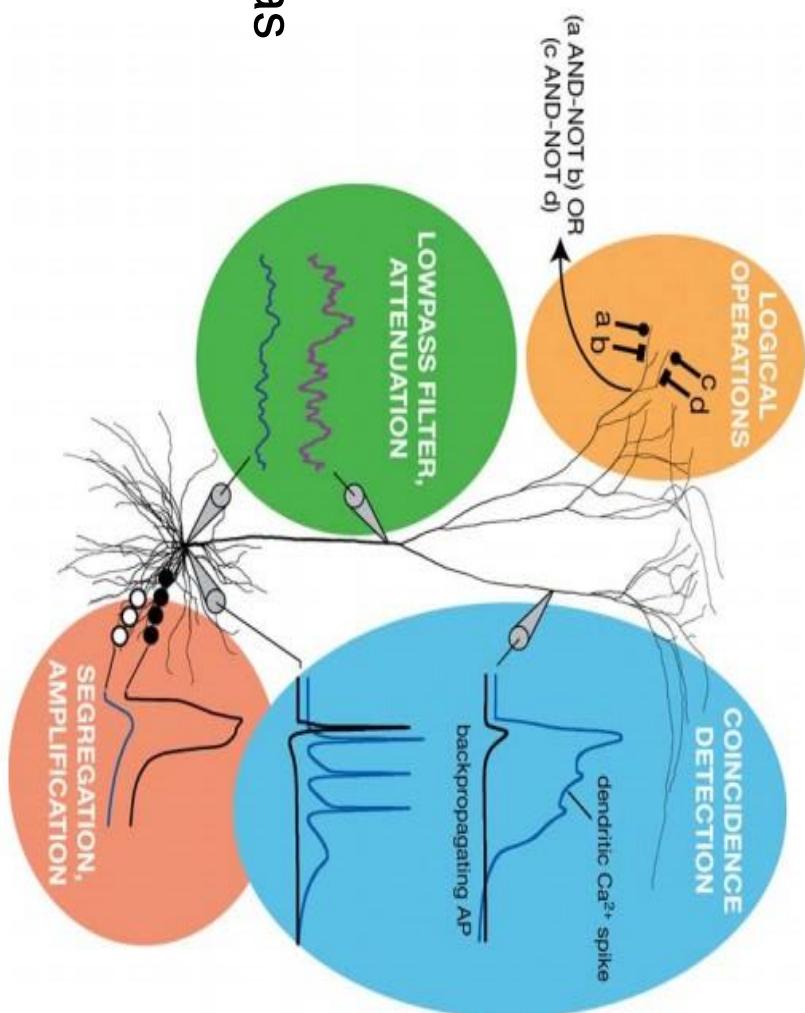
```
class Neuron:  
    # ...  
  
    def neuron_tick(inputs):  
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """  
        cell_body_sum = np.sum(inputs * self.weights) + self.bias  
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function  
        return firing_rate
```



# Cuidados com Analogias

## Neurônio biológicos:

- Vários tipos diferentes
- Dendritos pode realizar computações não-lineares
- Sinapses não representam apenas um “simples peso” mas sim um complexo sistema dinâmico não-linear

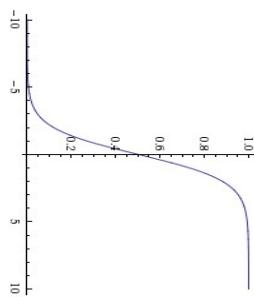


London, M., & Häusser, M. Dendritic computation. *Annual Review of Neuroscience*, 28: 503-532, (2005).

# Algumas Funções de Ativação

## Sigmoid

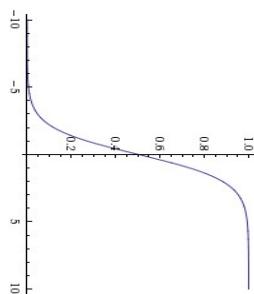
$$\sigma(x) = 1/(1 + e^{-x})$$



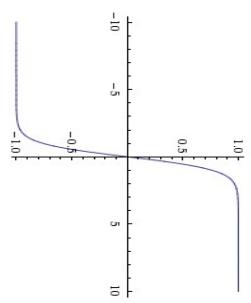
# Algumas Funções de Ativação

## Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



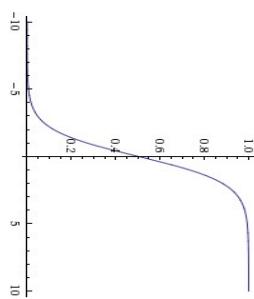
## Tanh tanh(x)



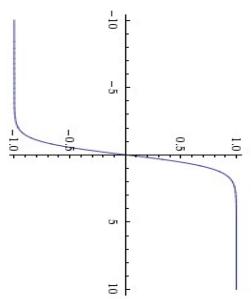
# Algumas Funções de Ativação

## Sigmoid

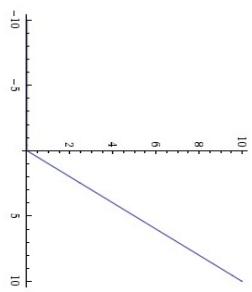
$$\sigma(x) = 1/(1 + e^{-x})$$



## Tanh

$$\tanh(x)$$


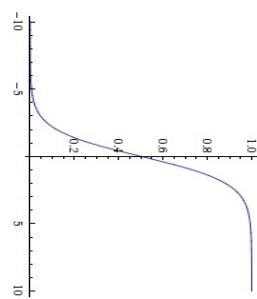
## ReLU

$$\max(0, x)$$


# Algumas Funções de Ativação

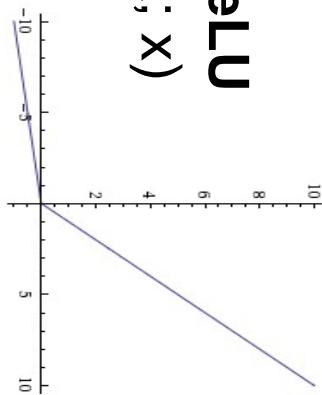
## Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



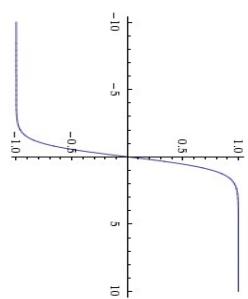
## Leaky ReLU

$$\max(0, \alpha x; x)$$



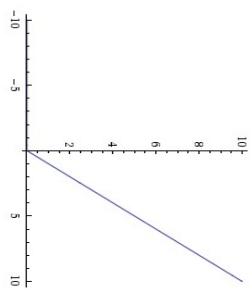
## Tanh

$$\tanh(x)$$



## ReLU

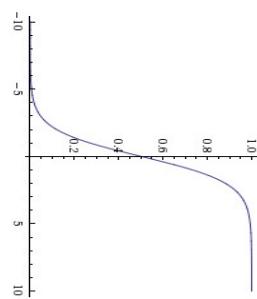
$$\max(0, x)$$



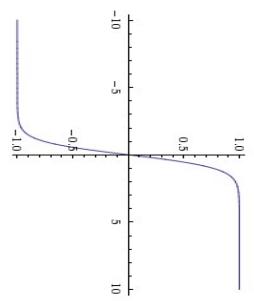
# Algumas Funções de Ativação

## Sigmoid

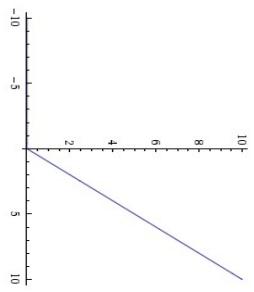
$$\sigma(x) = 1/(1 + e^{-x})$$



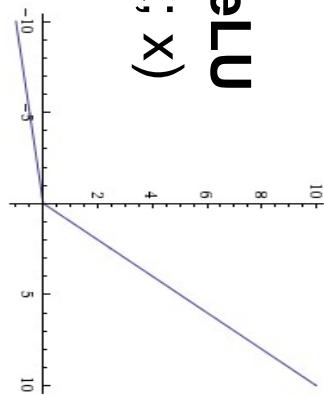
## Tanh tanh(x)



## ReLU max(0,x; x)

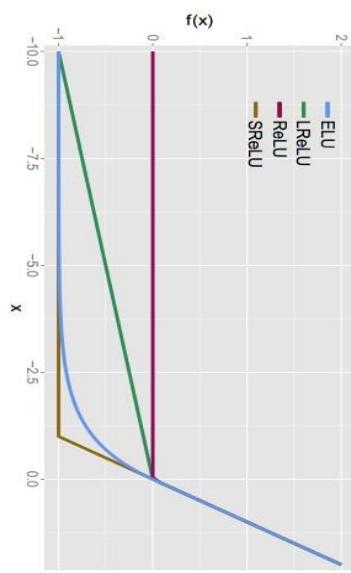


## Leaky ReLU $\max(0, \alpha x; x)$



## ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



## ReLU

## max(0,x)

## ReLU

## max(0,x)

# Redes Neurais e Aprendizagem Profunda

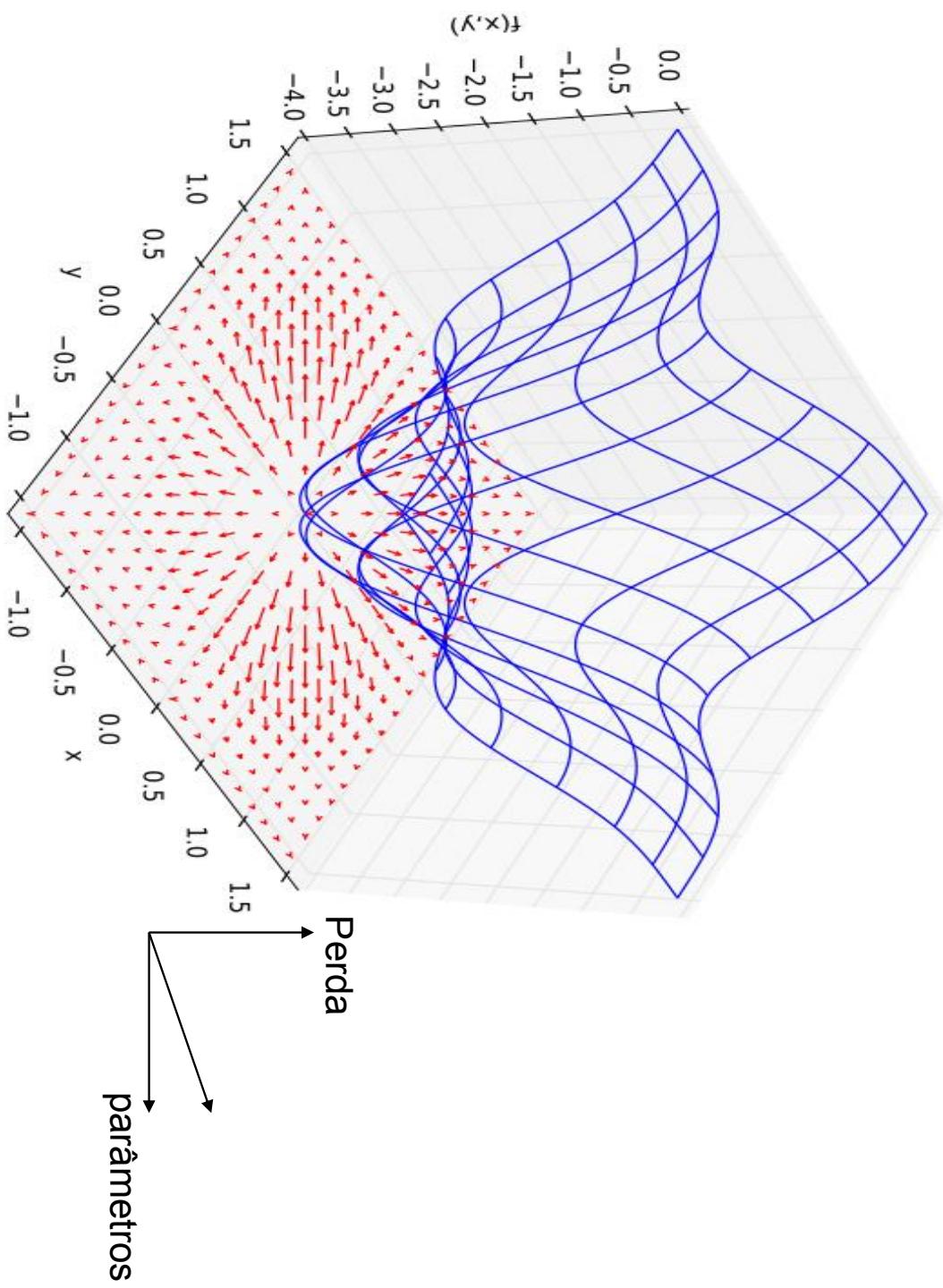
## REDES NEURAIS ARTIFICIAIS OTIMIZAÇÃO DA FUNÇÃO DE PERDA

---

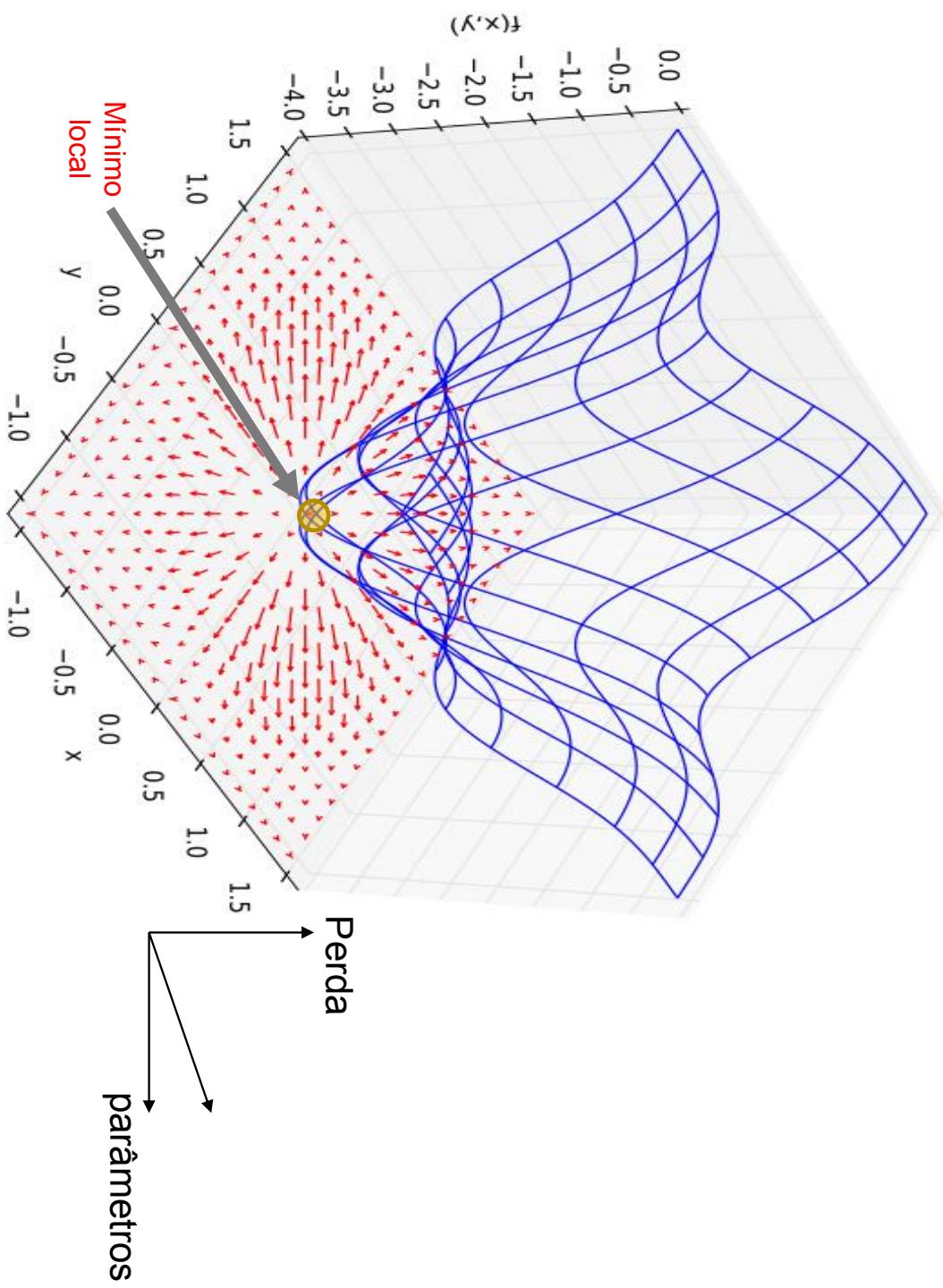
Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

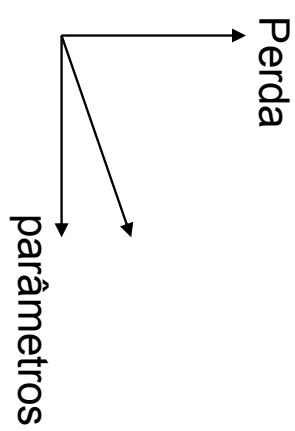
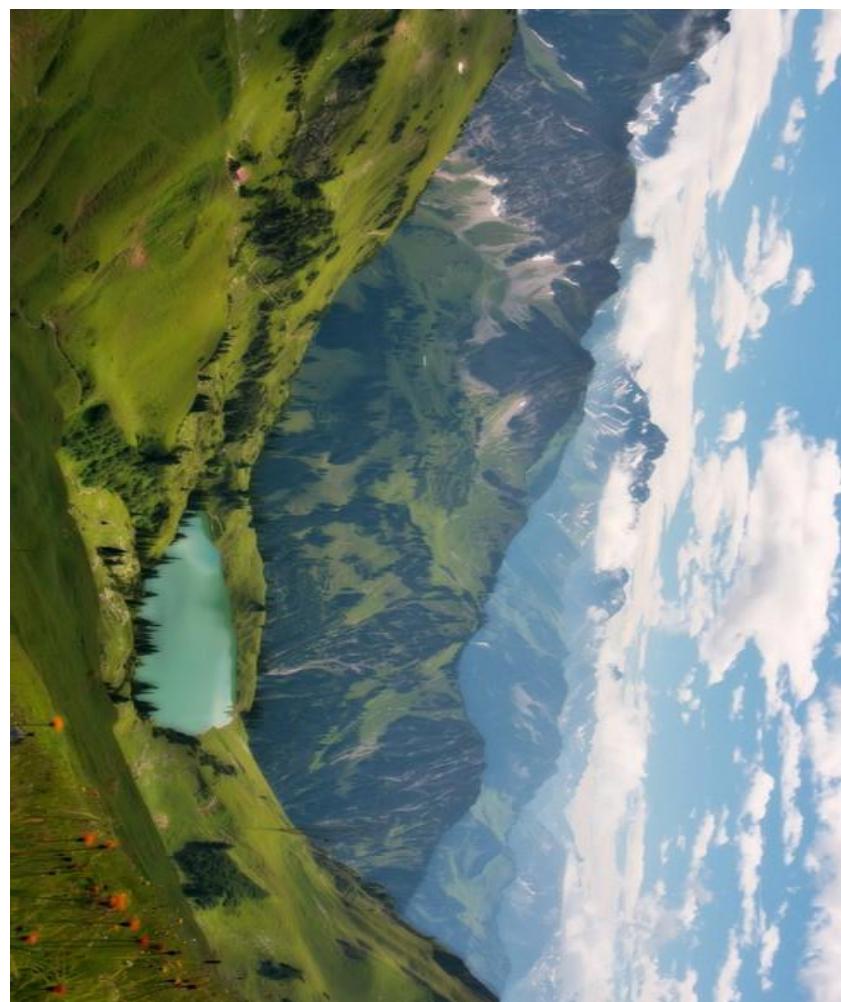
# Otimização da Função de Perda



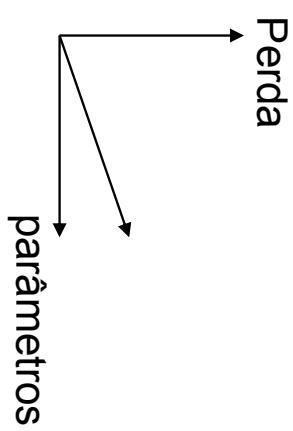
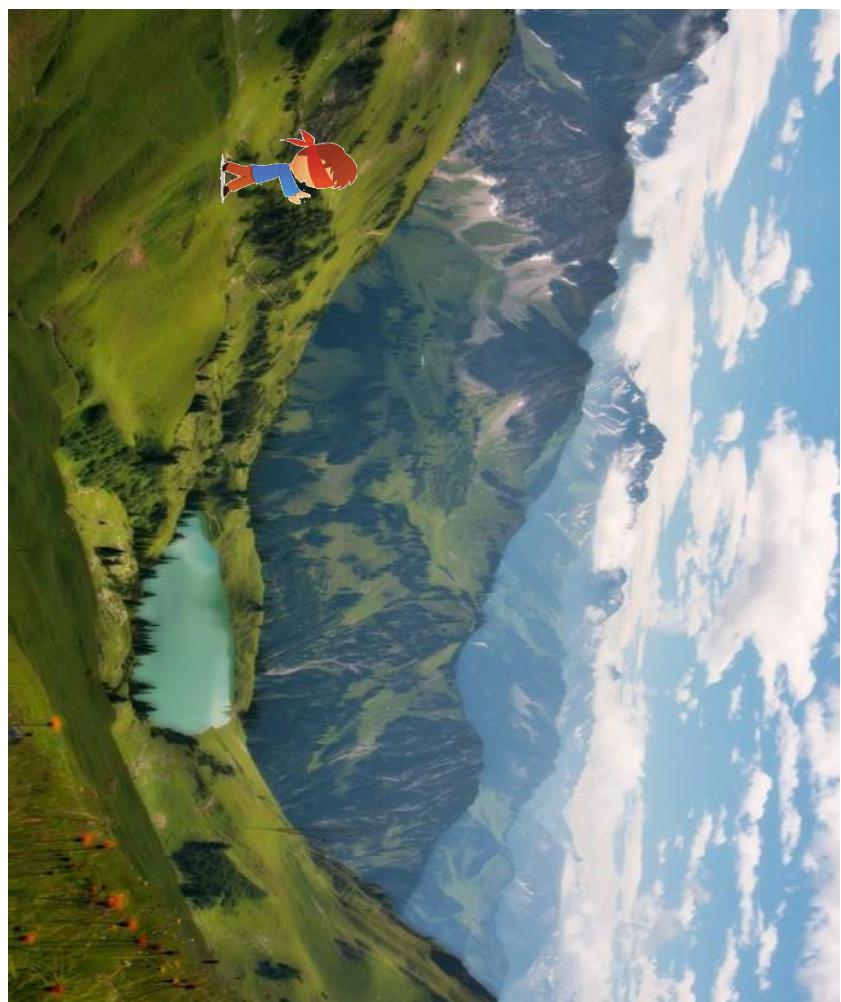
# Otimização da Função de Perda



# Otimização da Função de Perda



# Método 1 – Busca Randômica



# Método 1 – Busca Randômica

Talvez seja uma ideia muito ruim... mas é simples!

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
    W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
    loss = L(X_train, Y_train, W) # get the loss over the entire training set
    if loss < bestloss: # keep track of the best solution
        bestloss = loss
        bestW = W

print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

# Método 1 – Busca Randômica

Quão bem se comporta essa abordagem sobre o conjunto de teste...

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15,5% acurácia! Não é tão ruim!

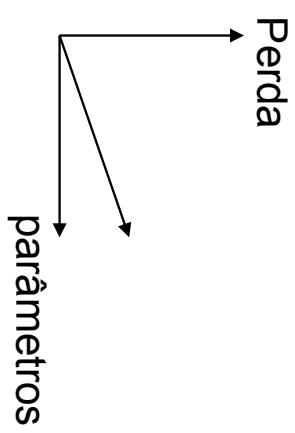
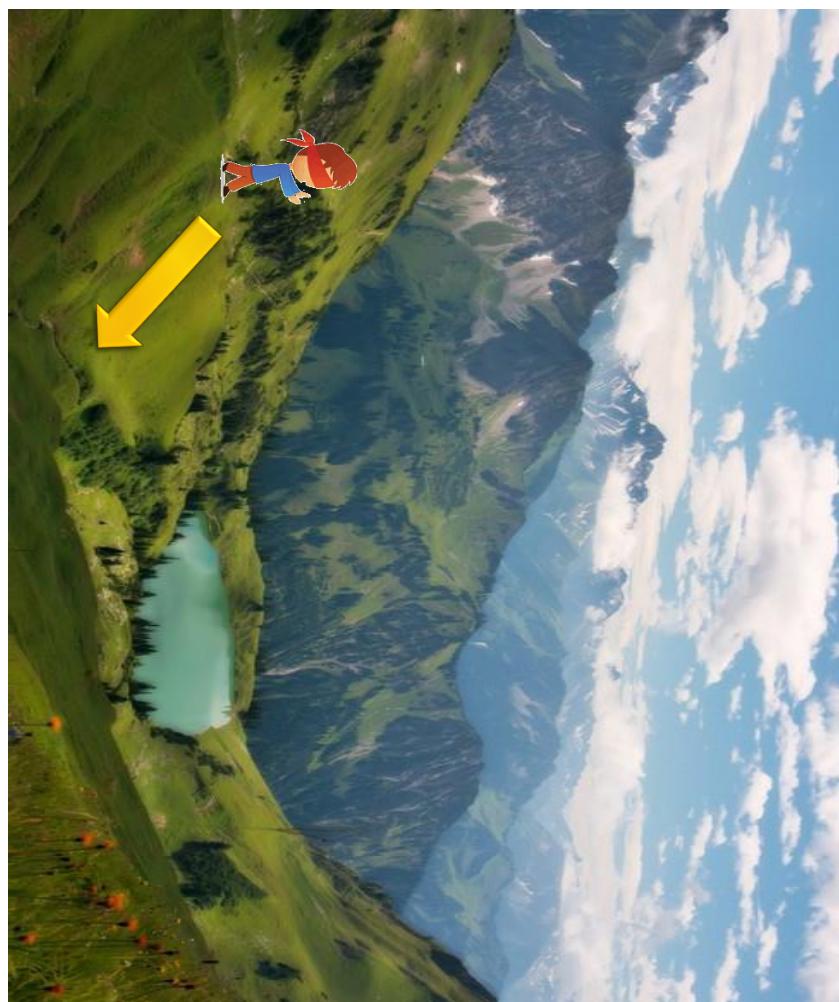
Porém, estado da arte é ~95%

# Método 1 – Busca Randômica

Toma-se vários passos aleatórios e mede-se a perda. Em seguida, direciona-se para o valor mais baixo encontrado

- Não é totalmente “sem sentido”. Esse é um tipo de otimização “sem gradiente”. Faz sentido se você não puder calcular gradientes por algum motivo
  - Uma versão um pouco mais inteligente calcula a média dos pontos aleatórios ponderados pela perda (pontos de menor perda ganham maior peso). Isso se aproxima do gradiente
- Mas é menos eficiente que o método do gradiente quando os gradientes estão disponíveis

# Método 2 – Seguindo a Inclinação



## Método 2 – Seguindo a Inclinação

Em uma dimensão, a derivada de uma função é dada por

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Em múltiplas dimensões, o **gradiente** representa o vetor de derivadas parciais

## Método 2 – Seguindo a Inclinação

Em uma dimensão, a derivada de uma função é dada por

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Em múltiplas dimensões, o **gradiente** representa o vetor de derivadas parciais

## Método 2 – Seguindo a Inclinação

Em uma dimensão, a derivada de uma função é dada por

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Em múltiplas dimensões, o **gradiente** representa o vetor de derivadas parciais

## Método 2 – Seguindo a Inclinação

Em uma dimensão, a derivada de uma função é dada por

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Em múltiplas dimensões, o **gradiente** representa o vetor de derivadas parciais

# Método 2 – Seguindo a Inclinação

**W corrente:**

```
[0,34;  
-1,11;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]
```

**perda 1,25347**

**gradiente dW:**

```
[?;  
?;  
?;  
?;  
?;  
?;  
?;  
?;  
?;...]
```

# Método 2 – Seguindo a Inclinação

**W corrente:**

[0,34;  
-1,11;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

**W + h (1ª dim):**

[0,34 + 0,0001;  
-1,11;  
0,78;  
0,12;  
0,55;

2,81;  
-3,1;  
-1,5;  
0,33;...]

**gradiente dW:**

[?;  
?;  
?;  
?;  
?;  
?;  
?;  
?;...]

**perda 1,25347**

**perda 1,25322**

# Método 2 – Seguindo a Inclinação

W corrente:

[0,34;  
-1,11;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

**perda 1,25347**

W + h (1ª dim):

[0,34 + 0,0001;  
-1,11;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

gradiente dW:

[-2,5;  
?;  
?;

?;  
?;

$$(1,25322 - 1,25347)/0,0001$$

$$= -2,5$$

$$\boxed{\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}}$$

?;  
?;...]

**perda 1,25322**

# Método 2 – Seguindo a Inclinação

**W corrente:**

[0,34;  
-1,11;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

**W + h (2<sup>a</sup> dim):**

[0,34;  
**-1,11 + 0,0001;**  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

**gradiente dW:**

[-2,5;  
?;  
?;  
?;  
?;  
?;  
?;  
?;  
?;...]

**perda 1,25347**

**perda 1,25353**

# Método 2 – Seguindo a Inclinação

W corrente:

[0,34;  
-1,11;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

W + h (2<sup>a</sup> dim):

[0,34;  
-1,11 + 0,0001;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

gradiente dW:

[-2,5;  
**0,6;** ↗  
?;  
?;

$$(\textcolor{red}{1,25353} - \textcolor{red}{1,25347})/0,0001 = 0,6$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

?;...]

**perda 1,25347**

**perda 1,25353**

# Método 2 – Seguindo a Inclinação

**W corrente:**

[0,34;  
-1,11;  
0,78;

0,12;  
0,55;  
2,81;

-3,1;  
-1,5;  
0,33;...]

**perda 1,25347**

**W + h (3<sup>a</sup> dim):**

[0,34;  
-1,11;  
0,78 + 0,0001;

0,12;  
0,55;  
2,81;

-3,1;  
-1,5;  
0,33;...]

**perda 1,25347**

**gradiente dW:**

[-2,5;  
0,6;  
?;  
?;  
?;  
?;

?;  
?;  
?;

?;  
?;

?;  
?;

?;

# Método 2 – Seguindo a Inclinação

W corrente:

[0,34;  
-1,11;  
0,78;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

W + h (3<sup>a</sup> dim):

[0,34;  
-1,11;  
0,78 + 0,0001;  
0,12;  
0,55;  
2,81;  
-3,1;  
-1,5;  
0,33;...]

gradiente dW:

[-2,5;  
0,6;  
**0**;  
?;  
?; 

$$(1,25347 - 1,25347)/0,0001 = 0$$

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

**perda 1,25347**

**perda 1,25347**

# Método 2 – Seguindo a Inclinação

Avaliação numérica do gradiente

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

```
def eval_numerical_gradient(f, x):
    """
    a naive implementation of numerical gradient of f at x
    - f should be a function that takes a single argument
    - x is the point (numpy array) to evaluate the gradient at
    """

    fx = f(x) # evaluate function value at original point
    grad = np.zeros(x.shape)
    h = 0.00001

    # iterate over all indexes in x
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:

        # evaluate function at x+h
        ix = it.multi_index
        old_value = x[ix]
        x[ix] = old_value + h # increment by h
        fxh = f(x) # evaluate f(x + h)
        x[ix] = old_value # restore to previous value (very important!)

        # compute the partial derivative
        grad[ix] = (fxh - fx) / h # the slope
        it.iternext() # step to next dimension

    return grad
```

# Método 2 – Seguindo a Inclinação

Avaliação numérica do gradiente

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

```
def eval_numerical_gradient(f, x):
    """
    a naive implementation of numerical gradient of f at x
    - f should be a function that takes a single argument
    - x is the point (numpy array) to evaluate the gradient at
    """

    fx = f(x) # evaluate function value at original point
    grad = np.zeros(x.shape)
    h = 0.00001

    # iterate over all indexes in x
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
```

- Aproximada
- Muito lenta

```
# evaluate function at x+h
ix = it.multi_index
old_value = x[ix]
x[ix] = old_value + h # increment by h
fxh = f(x) # evaluate f(x + h)
x[ix] = old_value # restore to previous value (very important!)

# compute the partial derivative
grad[ix] = (fxh - fx) / h # the slope
it.iternext() # step to next dimension

return grad
```

# Método 2 – Seguindo a Inclinação

Uma abordagem mais promissora!

A perda é apenas uma função de  $W$ , por exemplo:

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

Deseja-se

$$\nabla_W L = \dots$$

# Método 2 – Seguindo a Inclinação

Uma abordagem mais promissora

A perda é apenas uma função de W, por exemplo:

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

Deseja-se

$\nabla_W L = \dots$  ~~Solução Diferencial~~

Cálculo Diferencial



Newton



Leibniz

# Método 2 – Seguindo a Inclinação

**W corrente:**

[0,34;  
-1,11;  
0,78;

0,12;  
0,55;  
2,81;

-3,1;  
-1,5;  
0,33;...]

**perda 1,25347**

**gradiente dW:**

$dW = \dots$

(alguma função dos  
dados e de W)



[-2,5;  
0,6;  
0;

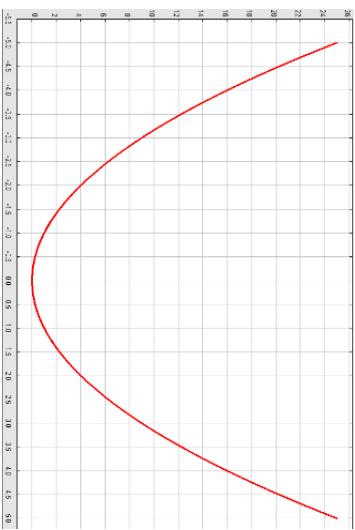
0,2;  
0,7;  
-0,5;

1,1;  
1,3;  
-2,1;...]

# Funções de Perda

## Perda Quadrática

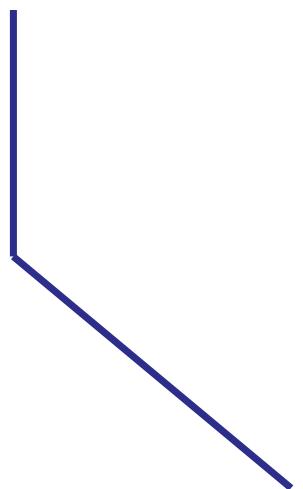
$$L = (y_i - f(x_i))^2$$



## Perda de Articulação

$$y_i \in \{-1,1\}$$

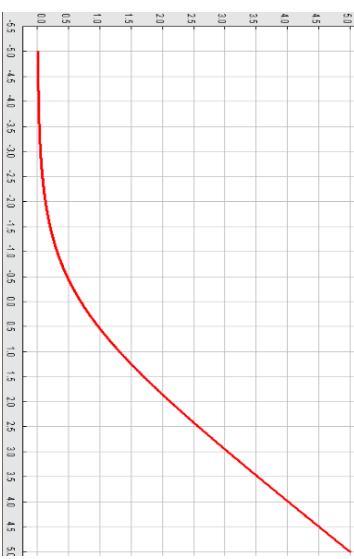
$$L = \max(0, 1 - y_i f(x_i))$$



## Perda de Entropia Cruzada

$$y_i \in \{-1,1\}$$

$$L = \log(1 + \exp(-y_i f(x_i)))$$



Todas essas três funções de perda possuem derivadas “bem comportadas”

Como  $f(x) = w^T x$  é uma função linear dos pesos  $W$ , pode-se derivar a perda em relação aos pesos

## Método 2 – Seguindo a Inclinação

- Gradiente Numérico: aproximado, lento, fácil de codificar
- Gradiente Analítico : exato, rápido, a prova de erros

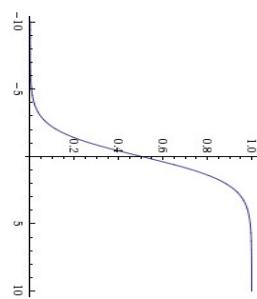
**Na prática:** Sempre se usa gradiente analítico, mas ocasionalmente verifica-se a implementação por meio de gradiente numérico.

⇒ Isto é chamado de **verificação de gradiente**

# Algumas Funções de Ativação

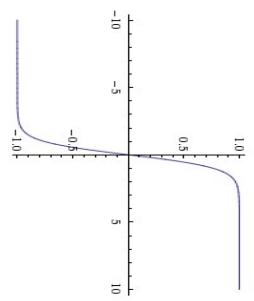
## Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



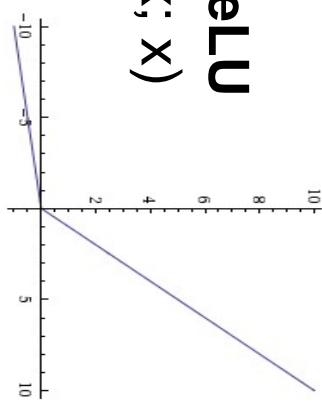
## Tanh

$$\tanh(x)$$



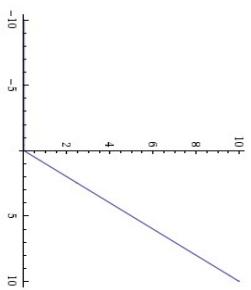
## ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



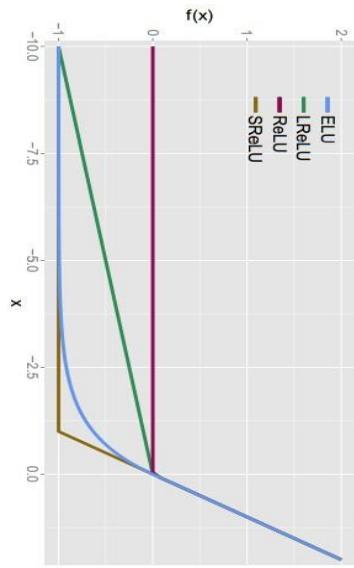
## ReLU

$$\max(0, x)$$



## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$



# Redes Neurais e Aprendizagem Profunda

## REDES NEURAIS ARTIFICIAIS MÉTODO DO GRADIENTE

---

Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

# Gradiente da Função de Perda

O gradiente  $\nabla_W L(W)$  representa o vetor de derivadas parciais

$$\nabla_W L(W) = \left[ \frac{\partial L}{\partial W_{11}}, \frac{\partial L}{\partial W_{12}}, \dots, \frac{\partial L}{\partial W_{21}}, \frac{\partial L}{\partial W_{22}}, \dots \right]^T$$

# Gradiente da Função de Perda

O gradiente  $\nabla_W L(W)$  representa o vetor de derivadas parciais

$$\nabla_W L(W) = \left[ \frac{\partial L}{\partial W_{11}}, \frac{\partial L}{\partial W_{12}}, \dots, \frac{\partial L}{\partial W_{21}}, \frac{\partial L}{\partial W_{22}}, \dots \right]^T$$

em que, por exemplo,  $\frac{\partial L}{\partial W_{11}}$  mede o quanto rápido varia a perda  $L$  em relação a uma variação do coeficiente  $W_{11}$  da matriz  $W$

# Gradiente da Função de Perda

O gradiente  $\nabla_W L(W)$  representa o vetor de derivadas parciais

$$\nabla_W L(W) = \left[ \frac{\partial L}{\partial W_{11}}, \frac{\partial L}{\partial W_{12}}, \dots, \frac{\partial L}{\partial W_{21}}, \frac{\partial L}{\partial W_{22}}, \dots \right]^T$$

em que, por exemplo,  $\frac{\partial L}{\partial W_{11}}$  mede o quanto rápido varia a perda  $L$  em relação a uma variação do coeficiente  $W_{11}$  da matriz  $W$

Neste caso, para se minimizar o valor do risco empírico é necessário se igualar a zero o gradiente de  $L$  em relação à matriz  $W$ , isto é

$$\nabla_W L(W) = 0$$

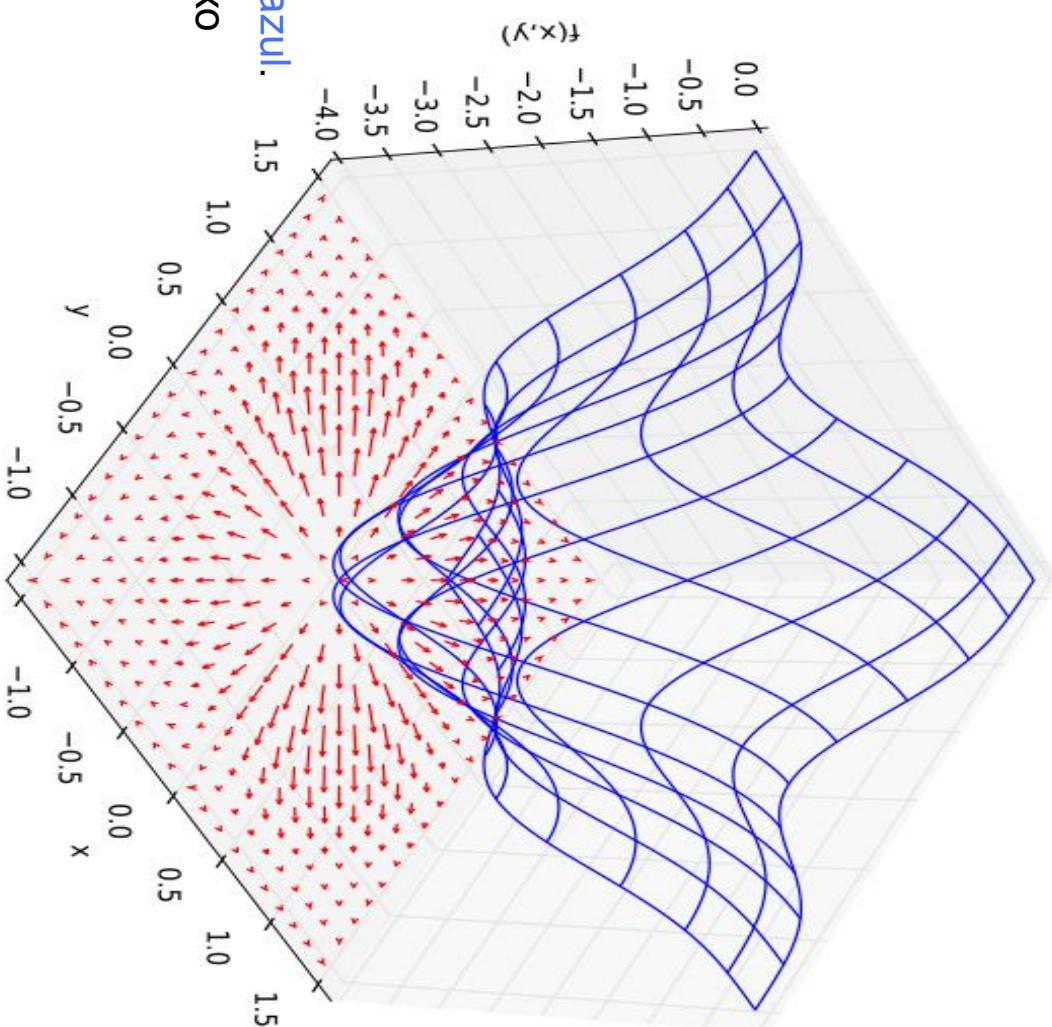
considerando, assim, que  $L$  é uma função da matriz  $W$

# Gradiente da Função de Perda

Quando  $\nabla_W L(W) = 0$ , então todas as derivadas parciais são nulas, ou ainda, a perda não se modifica em nenhuma direção

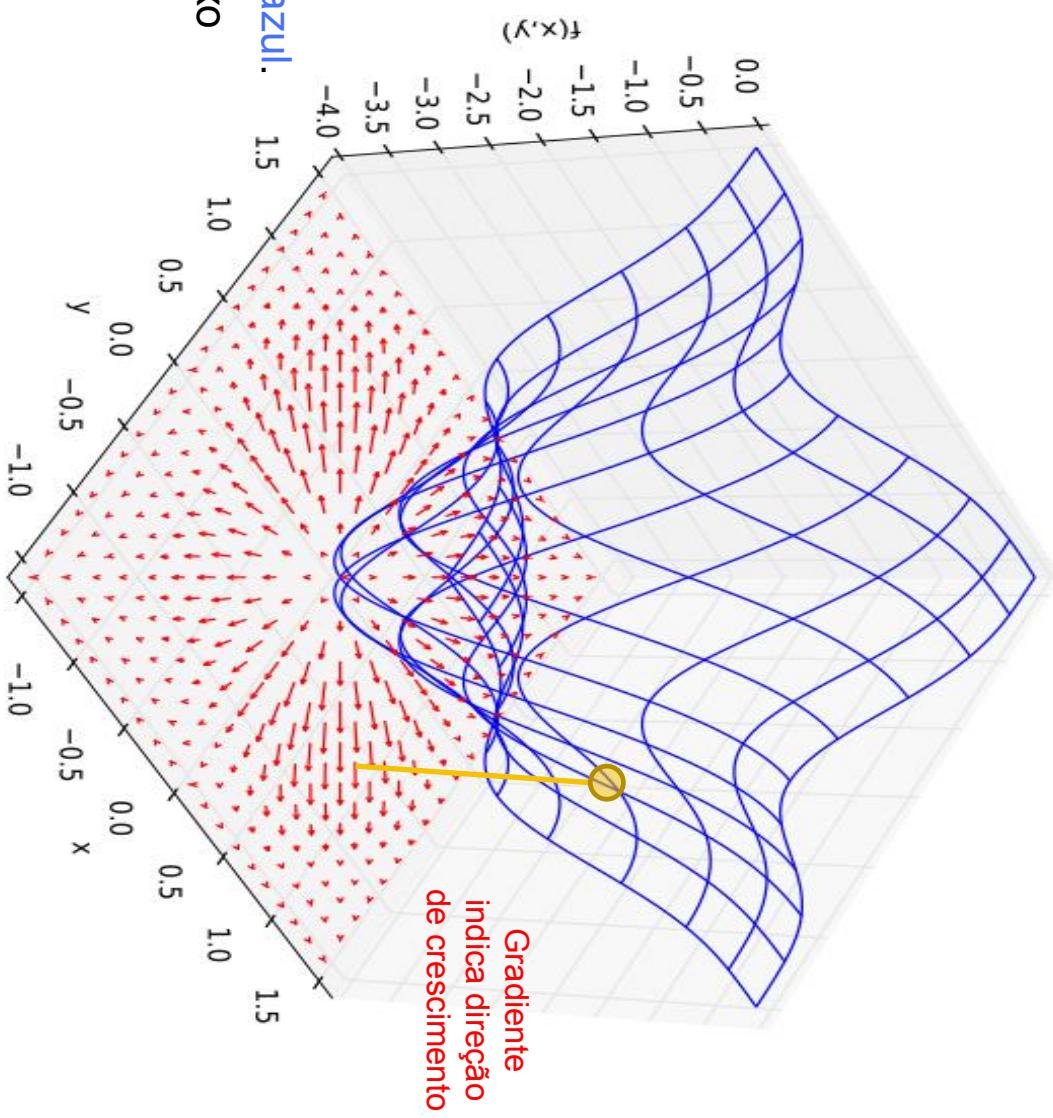
# Gradiente da Função de Perda

Quando  $\nabla_W L(W) = 0$ , então todas as derivadas parciais são nulas, ou ainda, a perda não se modifica em nenhuma direção



# Gradiente da Função de Perda

Quando  $\nabla_W L(W) = 0$ , então todas as derivadas parciais são nulas, ou ainda, a perda não se modifica em nenhuma direção

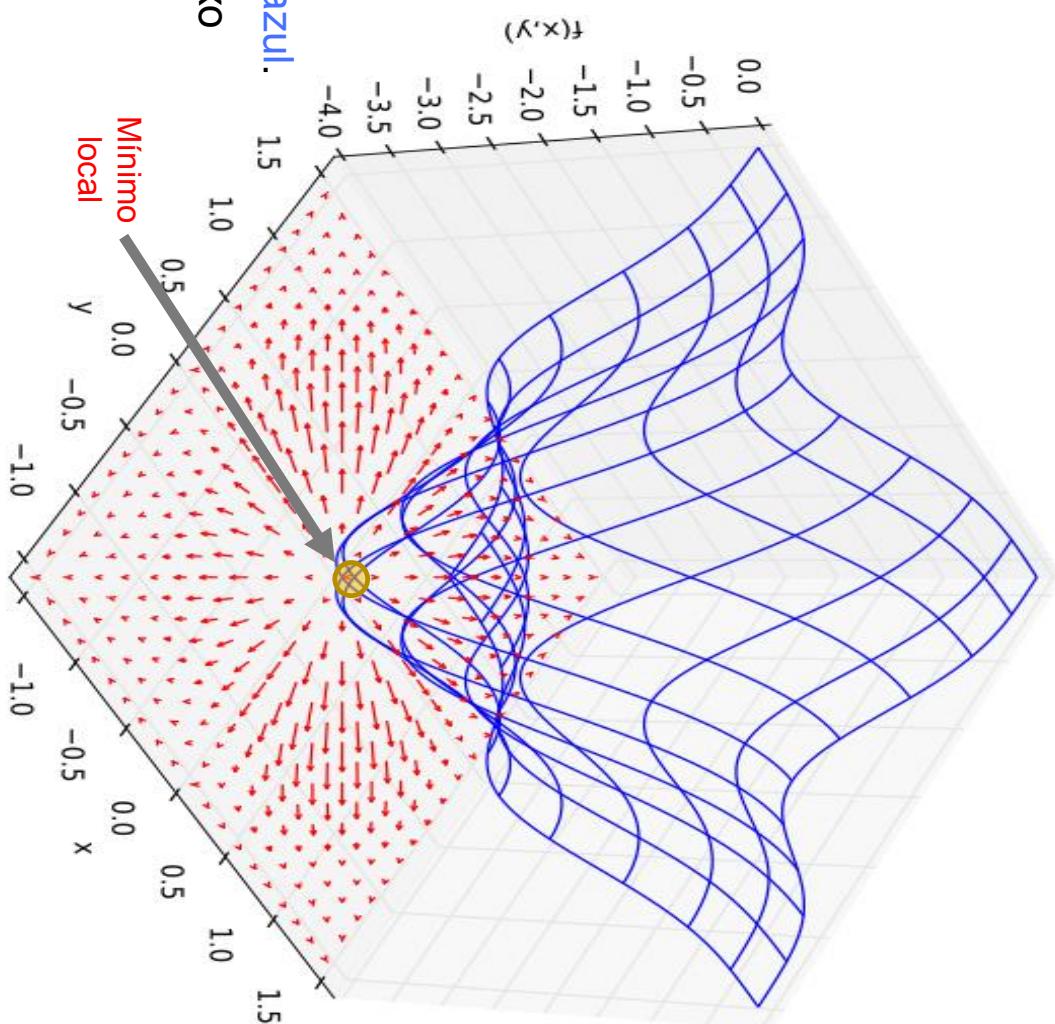


# Gradiente da Função de Perda

Quando  $\nabla_W L(W) = 0$ , então todas as derivadas parciais são nulas, ou ainda, a perda não se modifica em nenhuma direção

Portanto, obteve-se um ótimo local

- A superfície da perda aparece em azul.
- Os gradientes são mostrados abaixo como setas vermelhas.
- O gradiente é zero no mínimo.

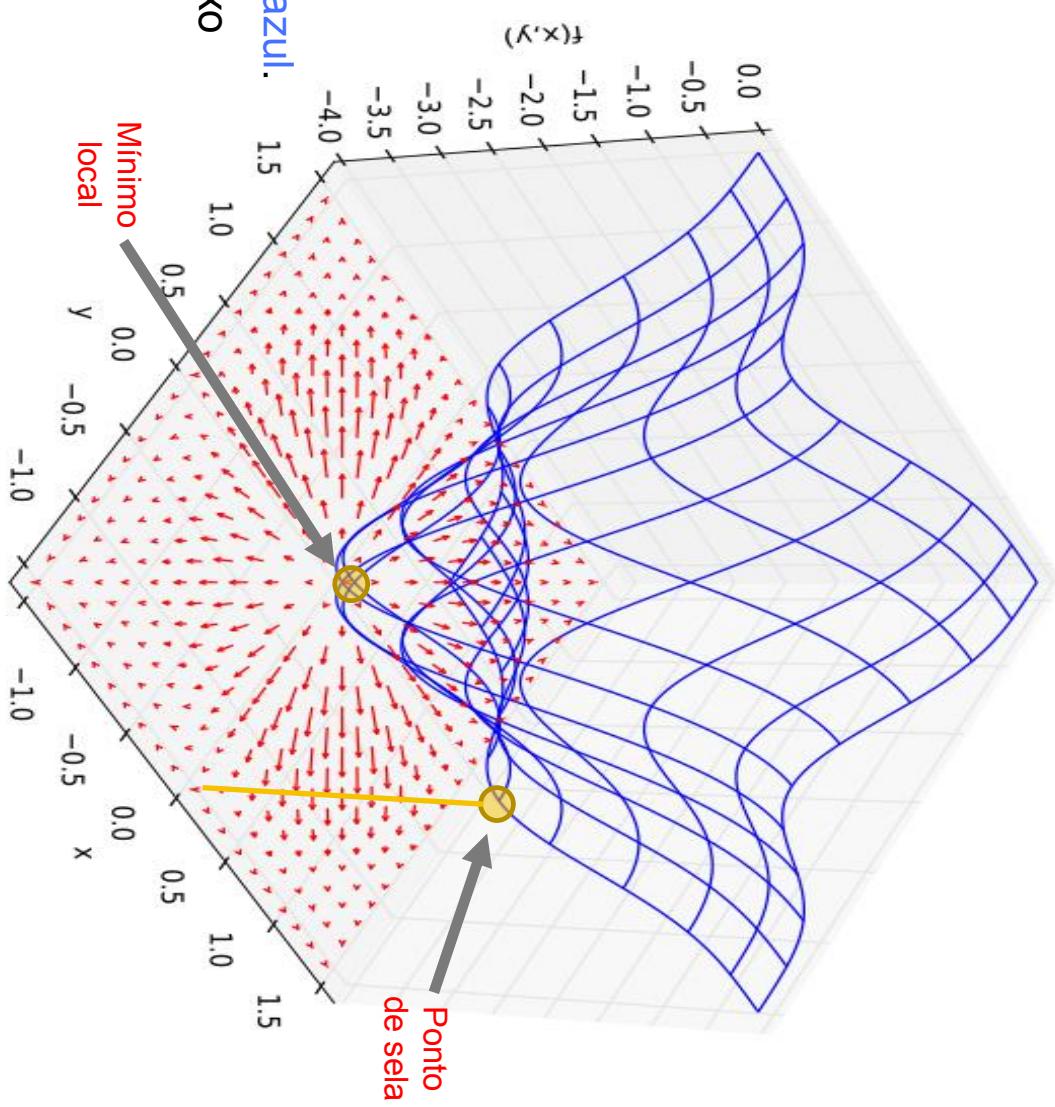


# Gradiente da Função de Perda

Quando  $\nabla_W L(W) = 0$ , então todas as derivadas parciais são nulas, ou ainda, a perda não se modifica em nenhuma direção

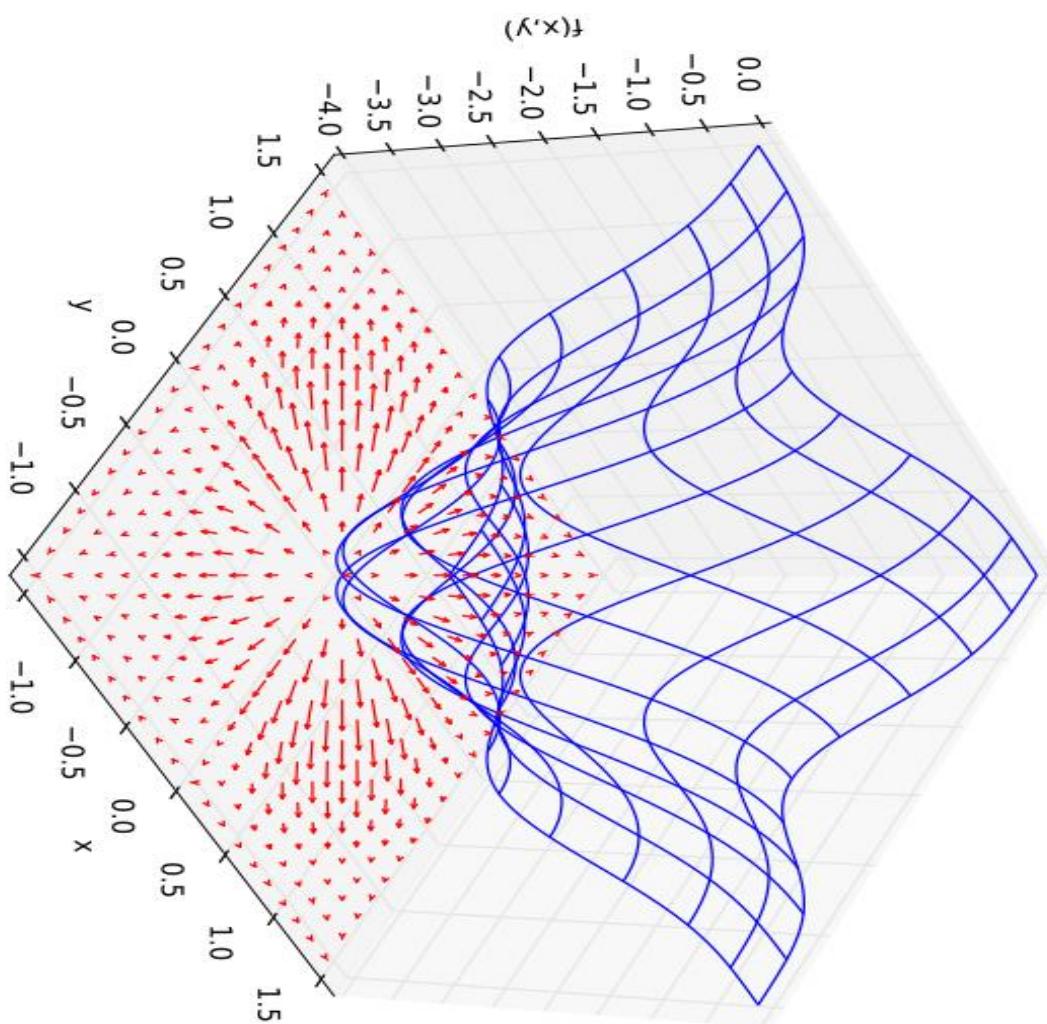
Portanto, obteve-se um ótimo local (ou, pelo menos, um ponto de sela)

- A superfície da perda aparece em azul.
- Os gradientes são mostrados abaixo como setas vermelhas.
- O gradiente é zero no mínimo.



# Método do Gradiente (ou Descida Mais Íngreme)

Para se alcançar um ponto de mínimo deve-se seguir a direção contrária ao gradiente,

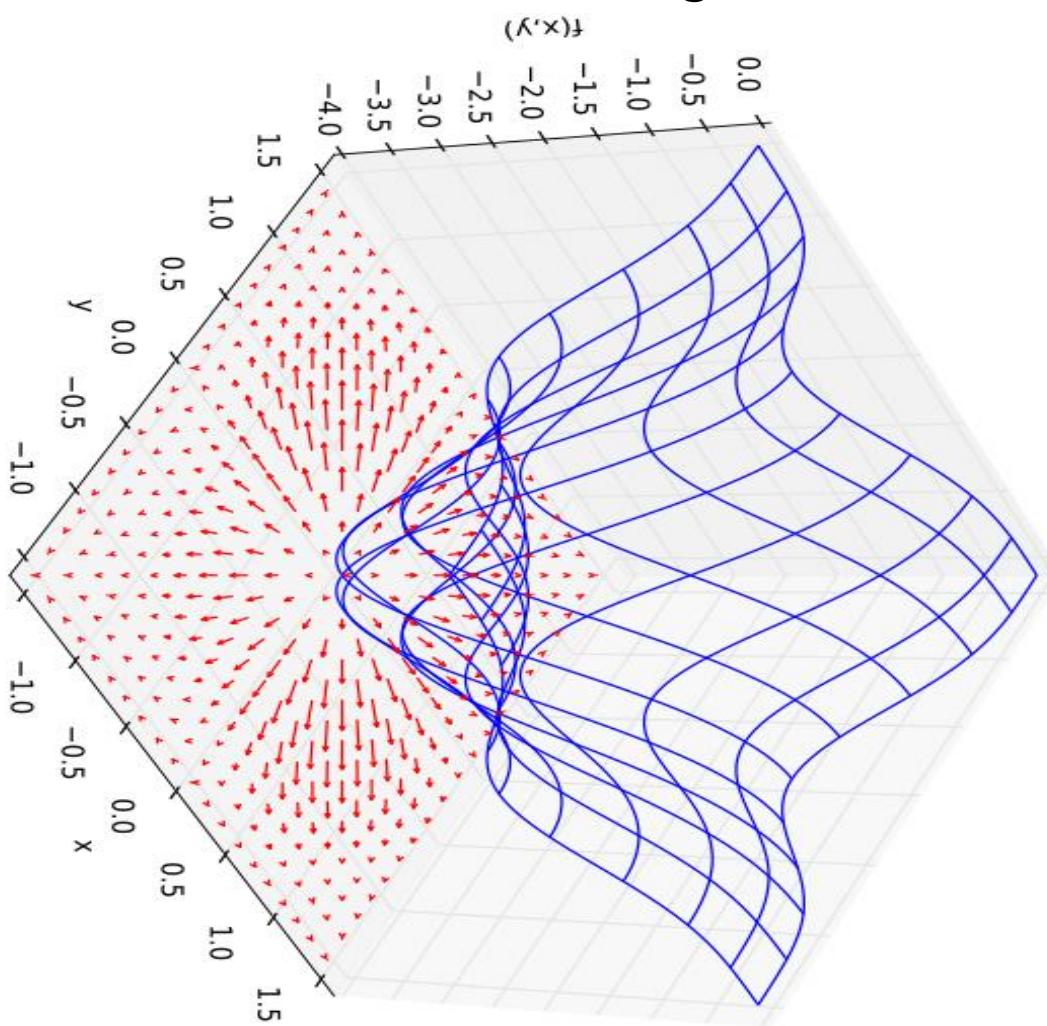


# Método do Gradiente (ou Descida Mais Íngreme)

Para se alcançar um ponto de mínimo deve-se seguir a direção contrária ao gradiente,

isto é, deve-se dar passos na direção

$$-\nabla_W L(W)$$



# Método do Gradiente (ou Descida Mais Íngreme)

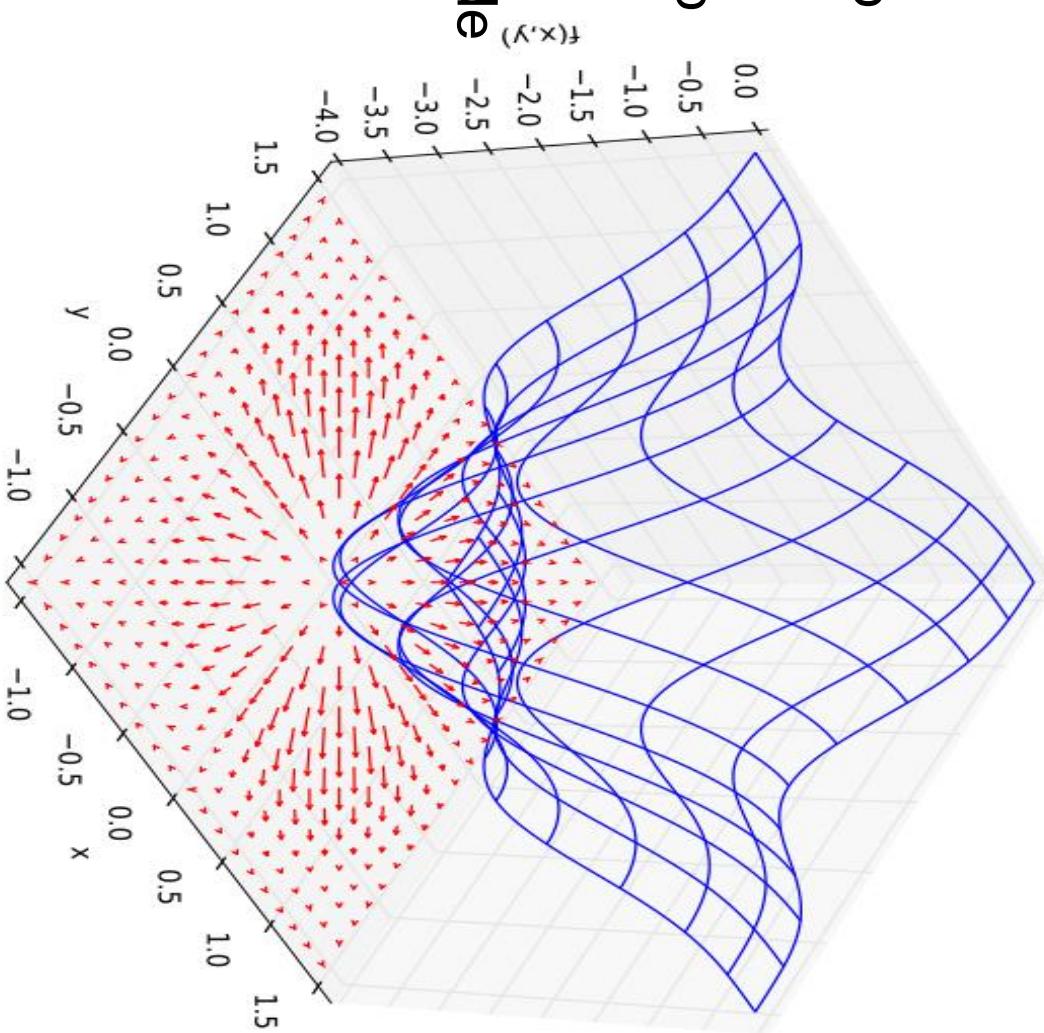
Para se alcançar um ponto de mínimo deve-se seguir a direção contrária ao gradiente,

isto é, deve-se dar passos na direção

$$-\nabla_W L(W)$$

Mais precisamente, seja  $W^t$  a matriz de pesos no passo  $t$  então

$$W^{t+1} = W^t - \alpha \nabla_W L(W)$$



# Método do Gradiente (ou Descida Mais Íngreme)

Para se alcançar um ponto de mínimo deve-se seguir a direção contrária ao gradiente,

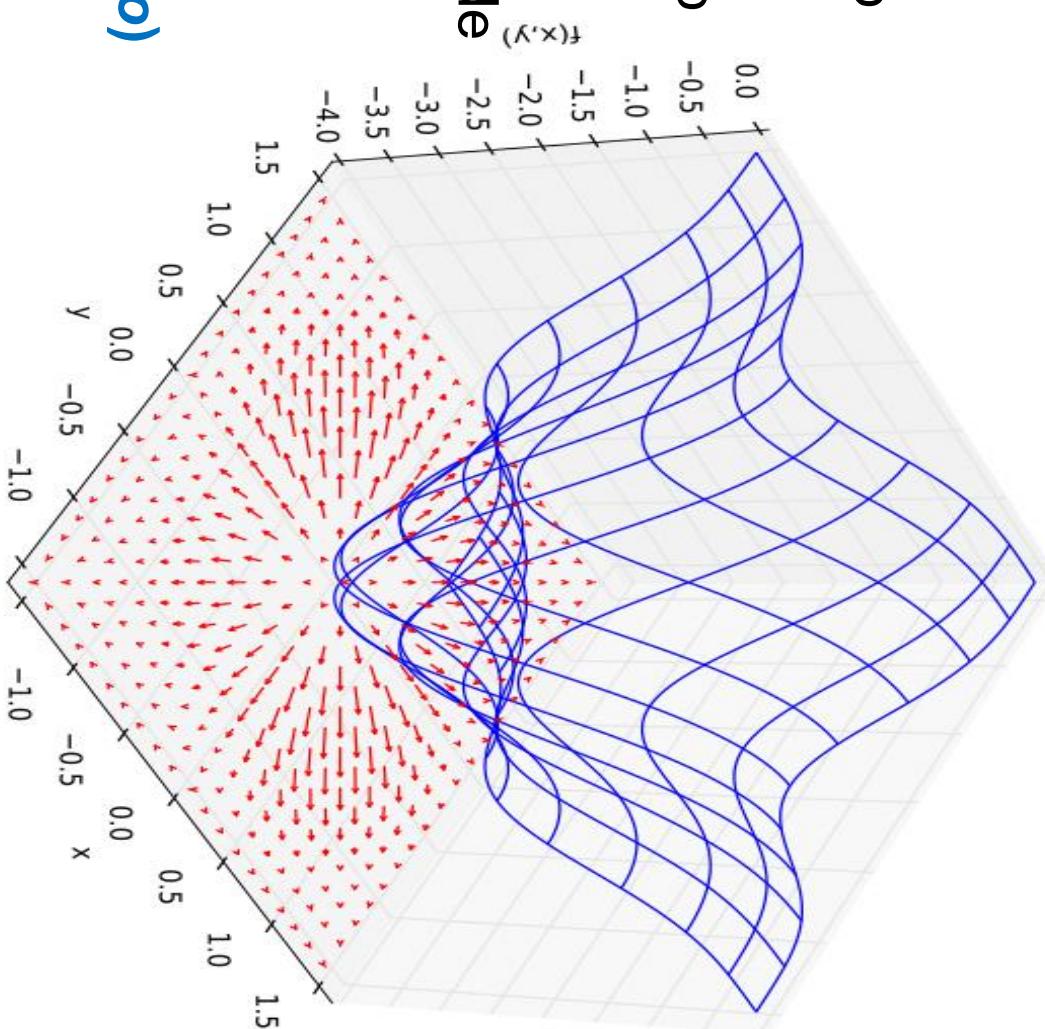
isto é, deve-se dar passos na direção

$$-\nabla_W L(W)$$

Mais precisamente, seja  $W^t$  a matriz de pesos no passo  $t$  então

$$W^{t+1} = W^t - \alpha \nabla_W L(W)$$

em que  $\alpha$  é chamado de **taxa de aprendizado (ou tamanho do passo)**



# Método do Gradiente (ou Descida Mais Íngreme)

Para se alcançar um ponto de mínimo deve-se seguir a direção contrária ao gradiente,

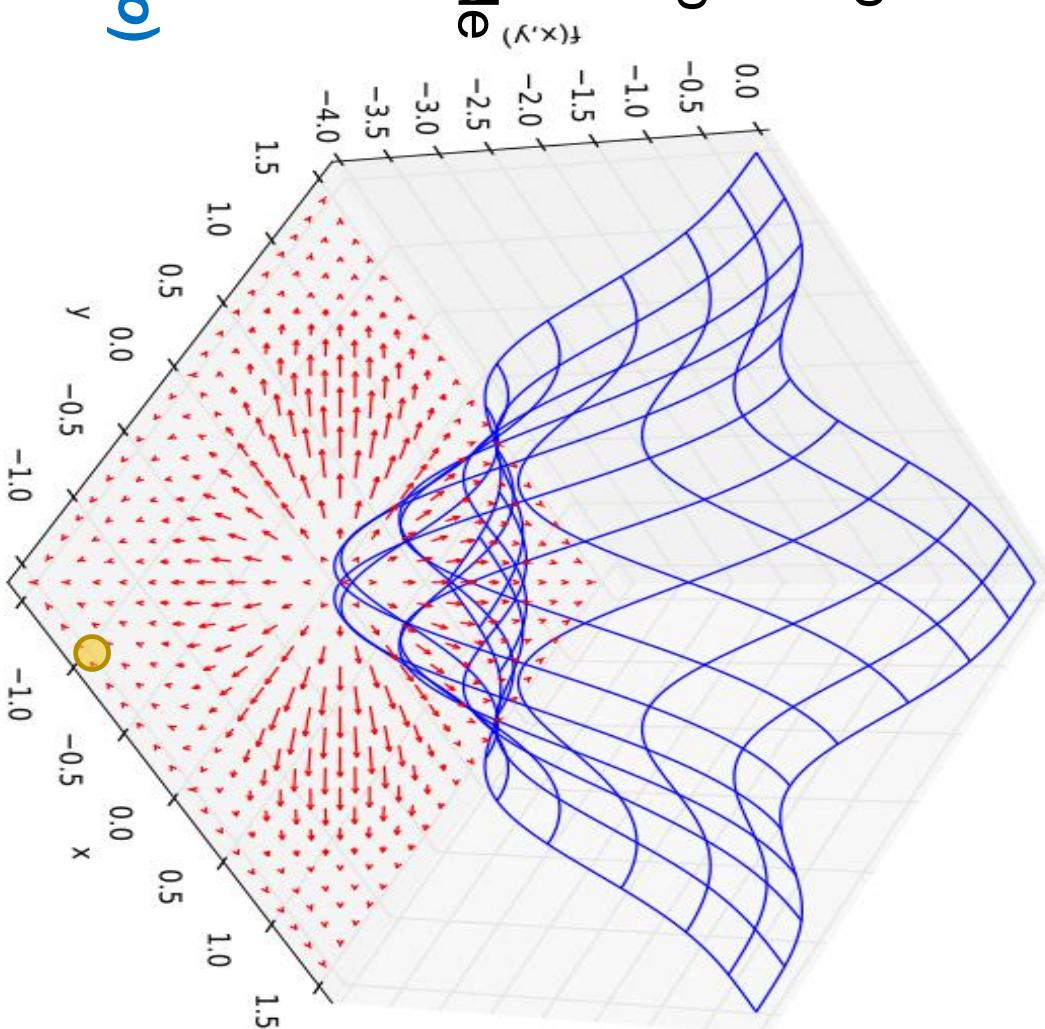
isto é, deve-se dar passos na direção

$$-\nabla_W L(W)$$

Mais precisamente, seja  $W^t$  a matriz de pesos no passo  $t$  então

$$W^{t+1} = W^t - \alpha \nabla_W L(W)$$

em que  $\alpha$  é chamado de **taxa de aprendizado (ou tamanho do passo)**



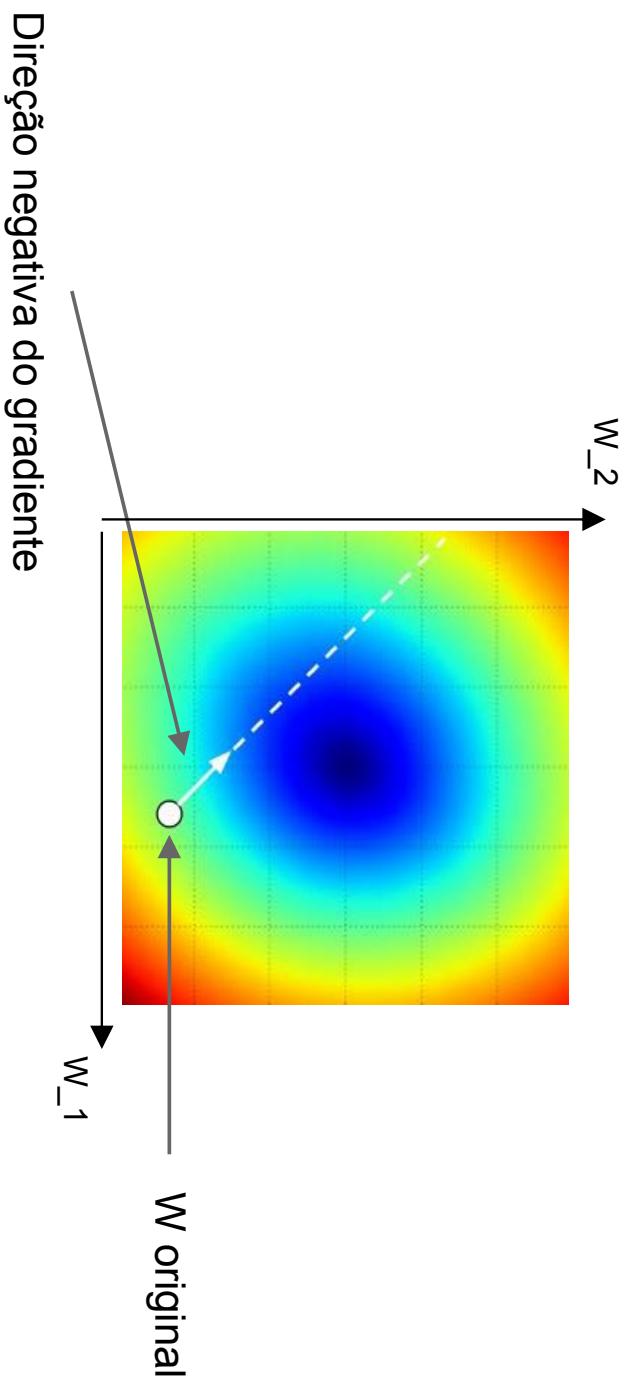
# Método do Gradiente (ou Descida Mais Íngreme)

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

# Método do Gradiente (ou Descida Mais Íngreme)

```
# Vanilla Gradient Descent
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```



# Redes Neurais e Aprendizagem Profunda

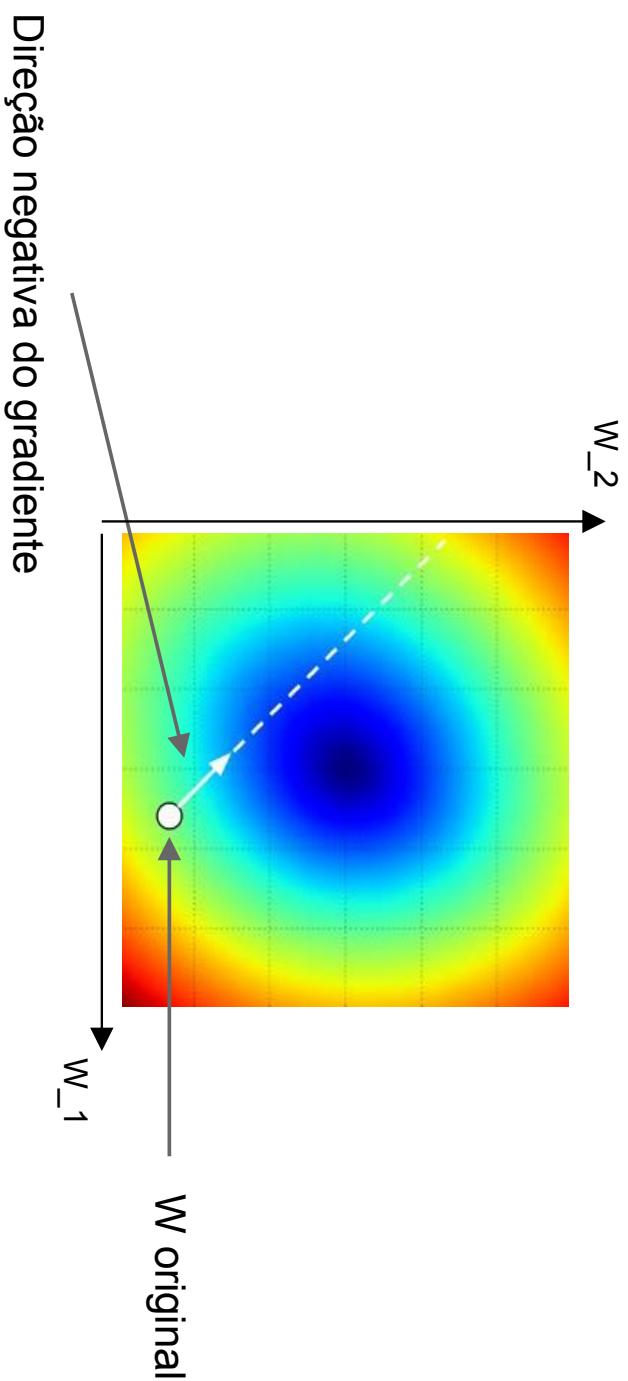
## REDES NEURAIS ARTIFICIAIS GRADIENTE DESCENDENTE ESTOCÁSTICO (SGD)

Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

# Método do Gradiente (ou Descida Mais Íngreme)

```
# Vanilla Gradient Descent
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```



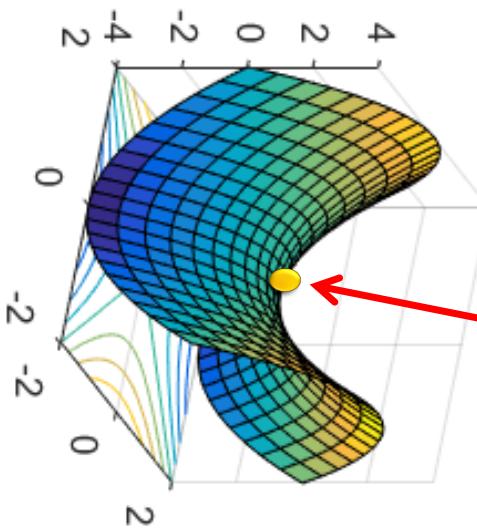
# Ressalva: Pontos de Sela

*Seguir a direção contrária ao gradiente deve levar a uma perda mínima*

# Ressalva: Pontos de Sela

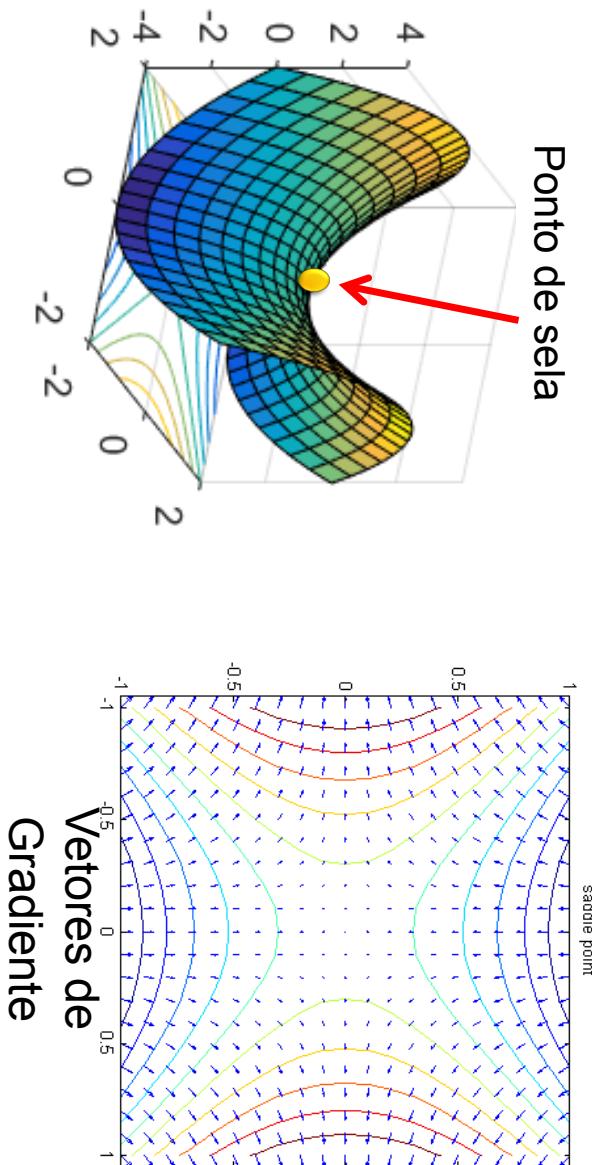
Seguir a direção contrária ao gradiente deve levar a uma perda mínima  
Mas pode demorar muito tempo. Uma razão é a presença de pontos de sela,  
em que o gradiente também desaparece

Ponto de sela



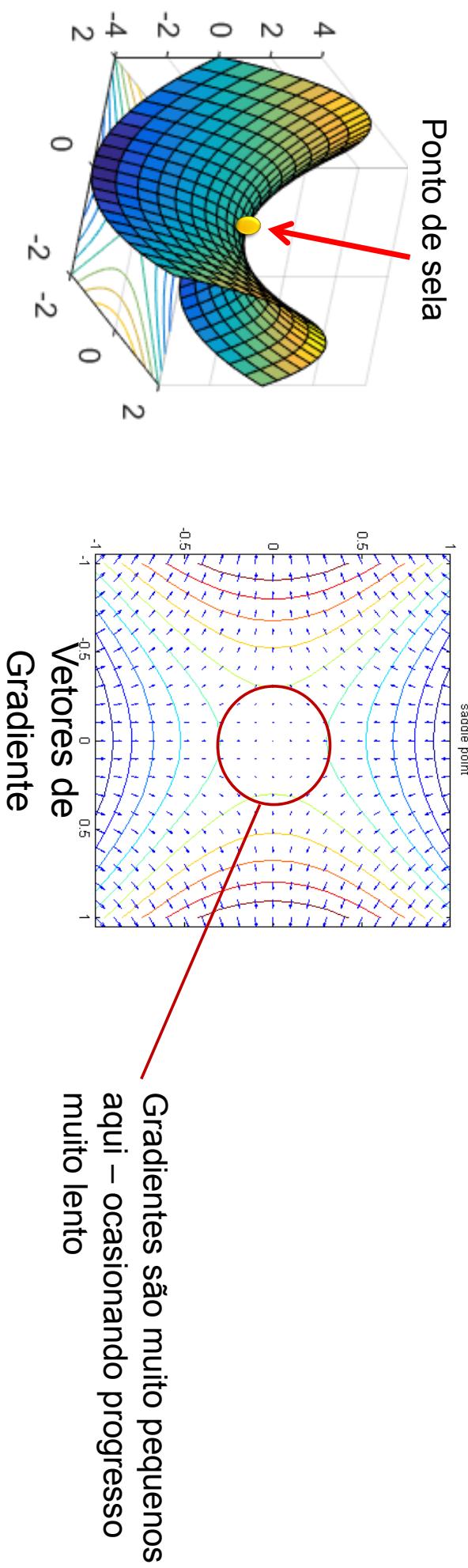
# Ressalva: Pontos de Sela

Seguir a direção contrária ao gradiente deve levar a uma perda mínima  
Mas pode demorar muito tempo. Uma razão é a presença de pontos de sela,  
em que o gradiente também desaparece



# Ressalva: Pontos de Sela

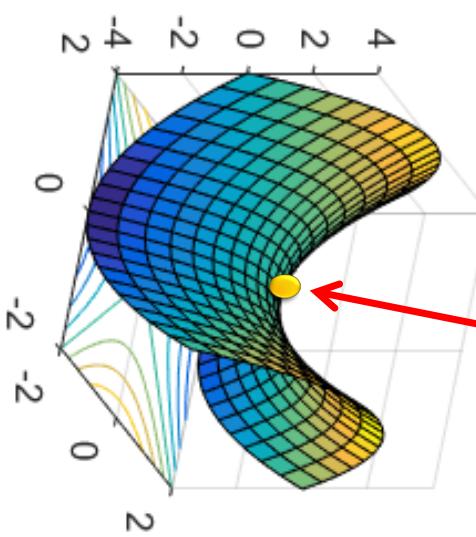
Seguir a direção contrária ao gradiente deve levar a uma perda mínima  
Mas pode demorar muito tempo. Uma razão é a presença de pontos de sela,  
em que o gradiente também desaparece



# Ressalva: Pontos de Sela

Existem várias evidências de que a maioria dos extremos da função de perda (ou zeros do gradiente da função de perda  $\nabla_W L$ ) para redes neurais profundas são de fato pontos de sela

Ponto de sela



Veja por exemplo

Dauphin et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”  
arXiv 1406.2572, 2014.

## Ressalva : Eficiência

A perda total  $L$  é dada pela soma das perdas para todos os itens de dados

$$L = \sum_{i=1}^N L(x_i, y_i, W)$$

## Ressalva : Eficiência

A perda total  $L$  é dada pela soma das perdas para todos os itens de dados

$$L = \sum_{i=1}^N L(x_i, y_i, W)$$

assim o gradiente da perda total é dado por

$$\frac{dL}{dW} = \sum_{i=1}^N \frac{dL}{dW}(x_i, y_i, W)$$

# Ressalva : Eficiência

A perda total  $L$  é dada pela soma das perdas para todos os itens de dados

$$L = \sum_{i=1}^N L(x_i, y_i, W)$$

assim o gradiente da perda total é dado por

$$\frac{dL}{dW} = \sum_{i=1}^N \frac{dL}{dW}(x_i, y_i, W)$$

Portanto, calcular o gradiente em relação a  $W$  exige uma **passagem completa pelo conjunto de dados**

# Ressalva : Eficiência

A perda total  $L$  é dada pela soma das perdas para todos os itens de dados

$$L = \sum_{i=1}^N L(x_i, y_i, W)$$

assim o gradiente da perda total é dado por

$$\frac{dL}{dW} = \sum_{i=1}^N \frac{dL}{dW}(x_i, y_i, W)$$

Portanto, calcular o gradiente em relação a  $W$  exige uma **passagem completa pelo conjunto de dados**

A obtenção do valor mínimo da função de perda pode exigir milhões de passos de gradiente e, portanto, se torna muito caro

# Processamento em Lotes (*Minibatches*)

Em vez de se calcular um gradiente sobre todo o conjunto de dados, pode-se calculá-lo usando um subconjunto de tamanho fixo de amostras de dados chamado ***minibatch*** (seu tamanho é tipicamente 32, 64, 128, 256,...)

# Processamento em Lotes (*Minibatches*)

Em vez de se calcular um gradiente sobre todo o conjunto de dados, pode-se calculá-lo usando um subconjunto de tamanho fixo de amostras de dados chamado ***minibatch*** (seu tamanho é tipicamente 32, 64, 128, 256,...)

O ***minibatch*** é idealmente uma amostra aleatória de tamanho  $m$  do conjunto de dados (na prática, podem ser apenas  $m$  amostras consecutivas)

# Processamento em Lotes (*Minibatches*)

Em vez de se calcular um gradiente sobre todo o conjunto de dados, pode-se calculá-lo usando um subconjunto de tamanho fixo de amostras de dados chamado ***minibatch*** (seu tamanho é tipicamente 32, 64, 128, 256,...)

O ***minibatch*** é idealmente uma amostra aleatória de tamanho  $m$  do conjunto de dados (na prática, podem ser apenas  $m$  amostras consecutivas)

Então se calcula o gradiente da seguinte forma: ( $N$  o tamanho do conjunto de dados,  $m$  o tamanho do ***minibatch***)

$$g^{(t)} = \frac{1}{m} \sum_{\substack{j=1, \dots, m \\ j \in \{1, \dots, N\}}} \nabla_W L(x_j, y_j, W)$$

# Gradiente Descendente Estocástico ( $SGD$ )

Considerando  $W^{(t)}$  como a matriz de pesos na iteração  $t$ , temos que:

$$W^{(t+1)} = W^{(t)} - \alpha g^{(t)}$$

# Gradiente Descendente Estocástico (SGD)

Considerando  $W^{(t)}$  como a matriz de pesos na iteração  $t$ , temos que:

$$W^{(t+1)} = W^{(t)} - \alpha g^{(t)}$$

Esse método é chamado de **gradiente descendente estocástico** (SGD). O SGD e suas variantes são usadas quase universalmente em treinamento de redes profundas

# Gradiente Descendente Estocástico (SGD)

Considerando  $W^{(t)}$  como a matriz de pesos na iteração  $t$ , temos que:

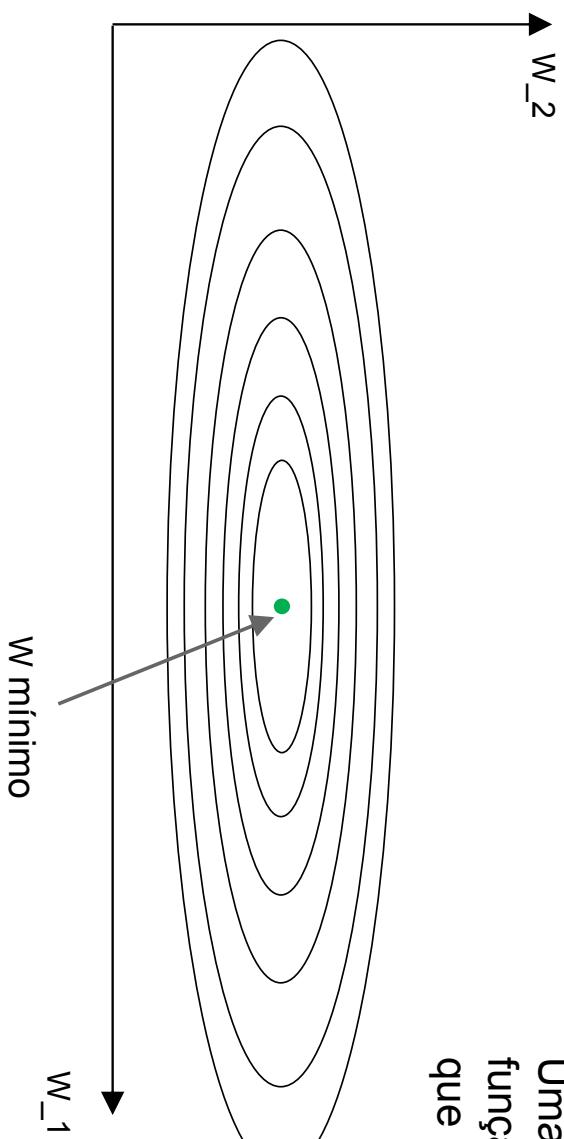
$$W^{(t+1)} = W^{(t)} - \alpha g^{(t)}$$

Esse método é chamado de **gradiente descendente estocástico** (SGD). O SGD e suas variantes são usadas quase universalmente em treinamento de redes profundas

O SGD usa  $g^{(t)}$ , isto é o gradiente de um **minibatch**, no lugar do gradiente sobre o conjunto de dados completo e é dito “estocástico” pois o gradiente de um **minibatch** é calculado sobre uma amostra do conjunto de dados

# Gradiente Descendente Estocástico (*SGD*)

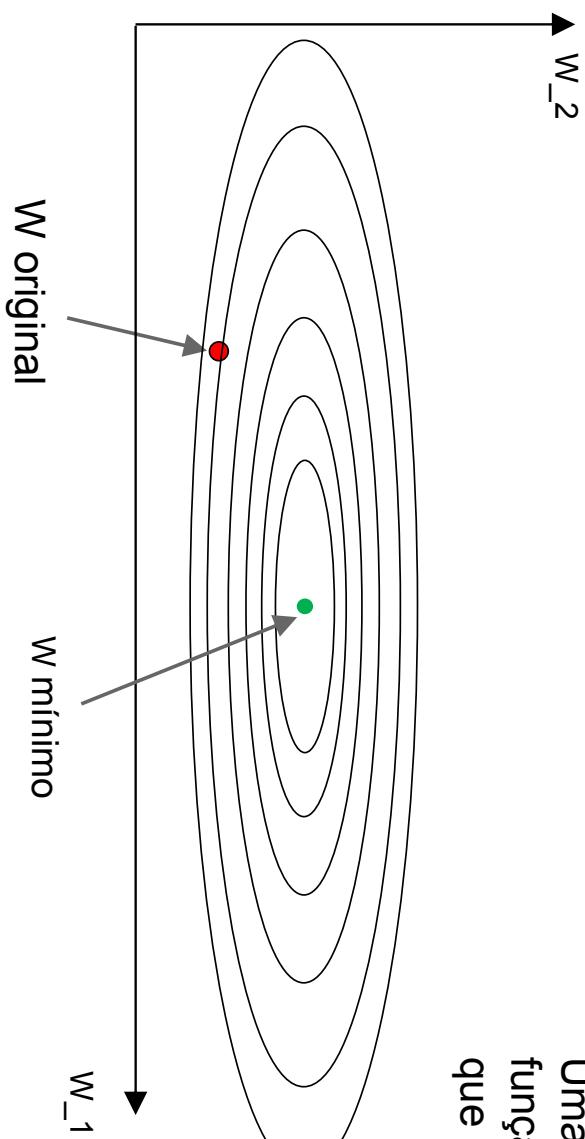
O gradiente da função em um ponto é sempre perpendicular a curva de nível naquele ponto



Uma curva de nível de uma função representa a curva em que o valor  $f(x) = \text{constante}$

# Gradiente Descendente Estocástico (*SGD*)

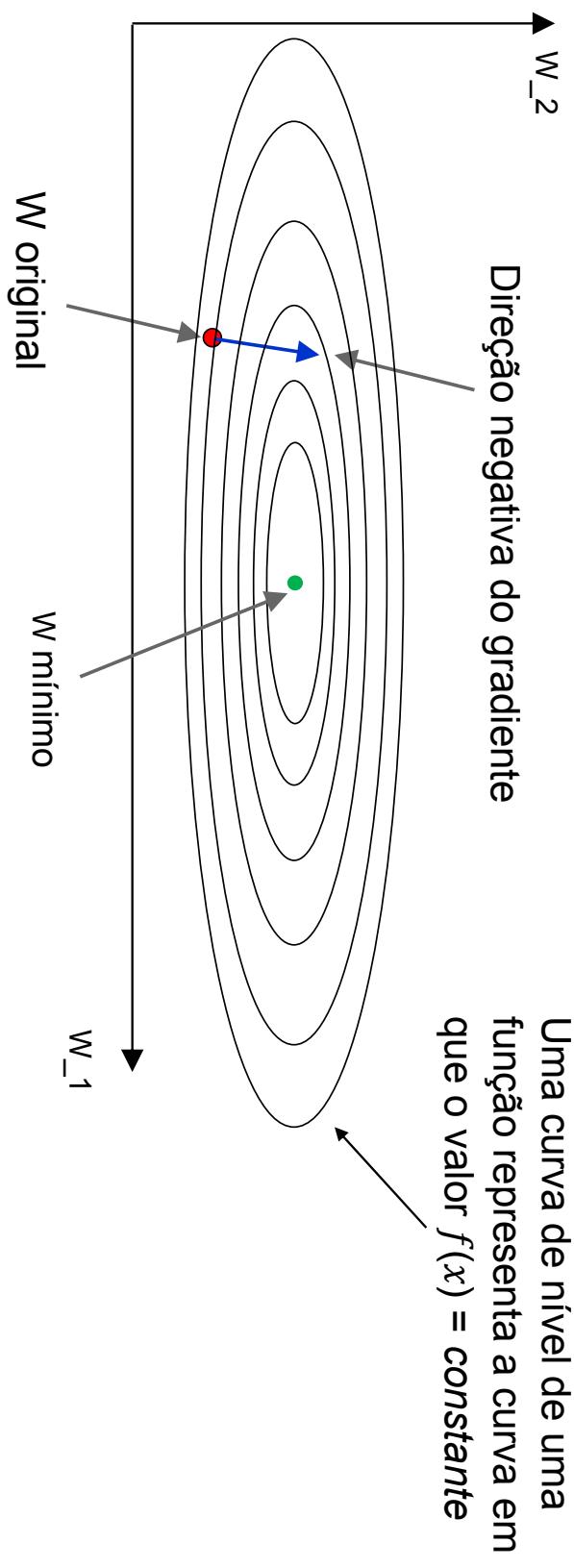
O gradiente da função em um ponto é sempre perpendicular a curva de nível naquele ponto



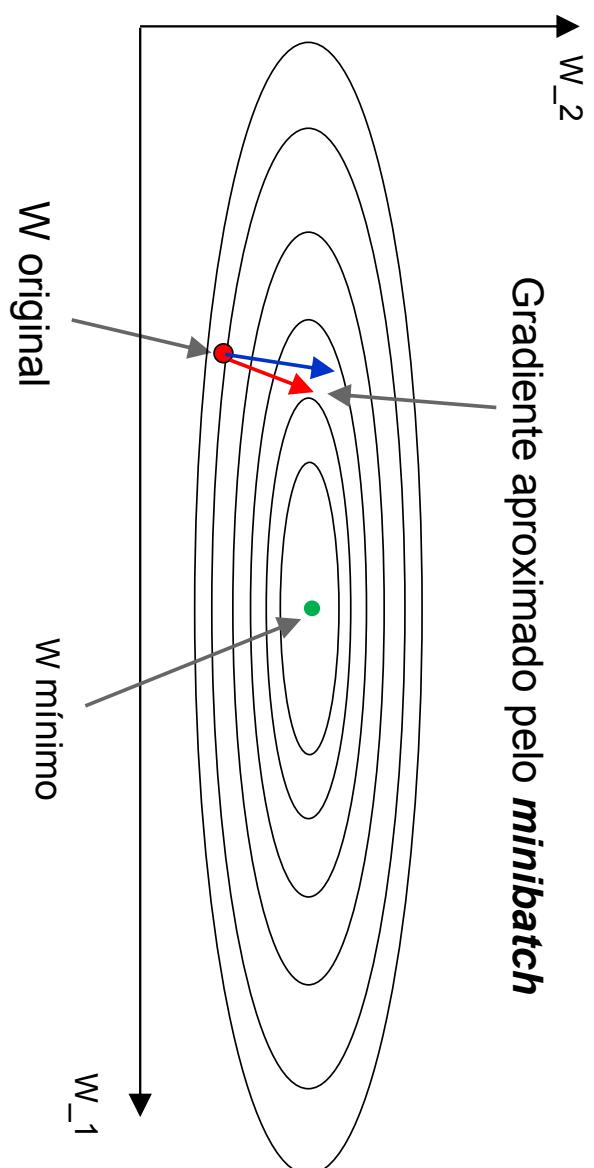
Uma curva de nível de uma função representa a curva em que o valor  $f(x) = \text{constante}$

# Gradiente Descendente Estocástico (SGD)

O gradiente da função em um ponto é sempre perpendicular a curva de nível naquele ponto



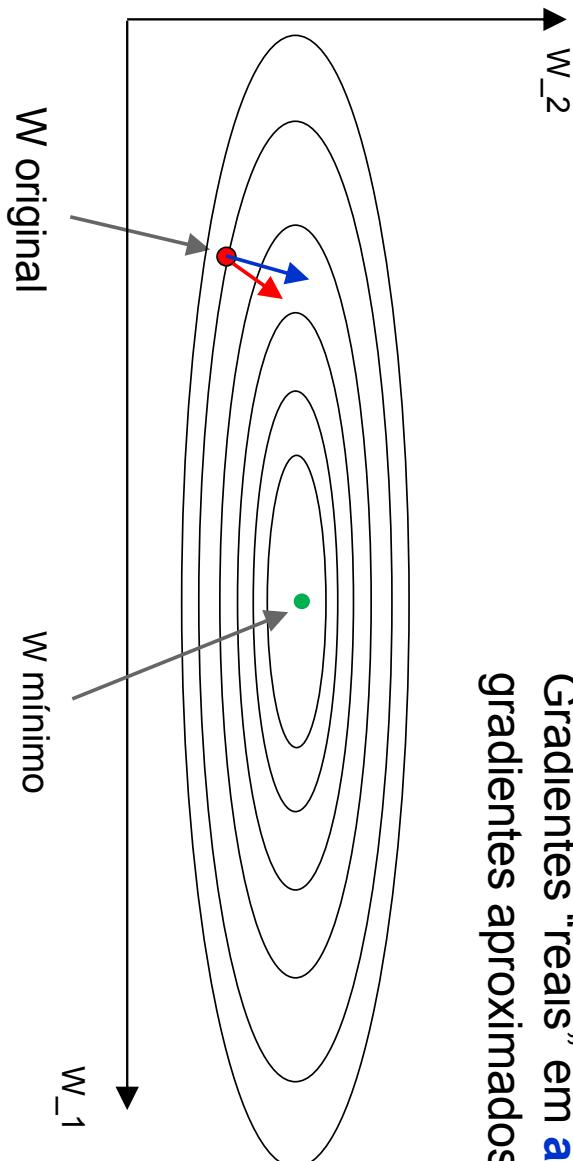
# Gradiente Descendente Estocástico (*SGD*)



# Gradiente Descendente Estocástico (*SGD*)

Gradientes apresentam “ruídos” mas ainda são bons para progredir na média em direção ao mínimo

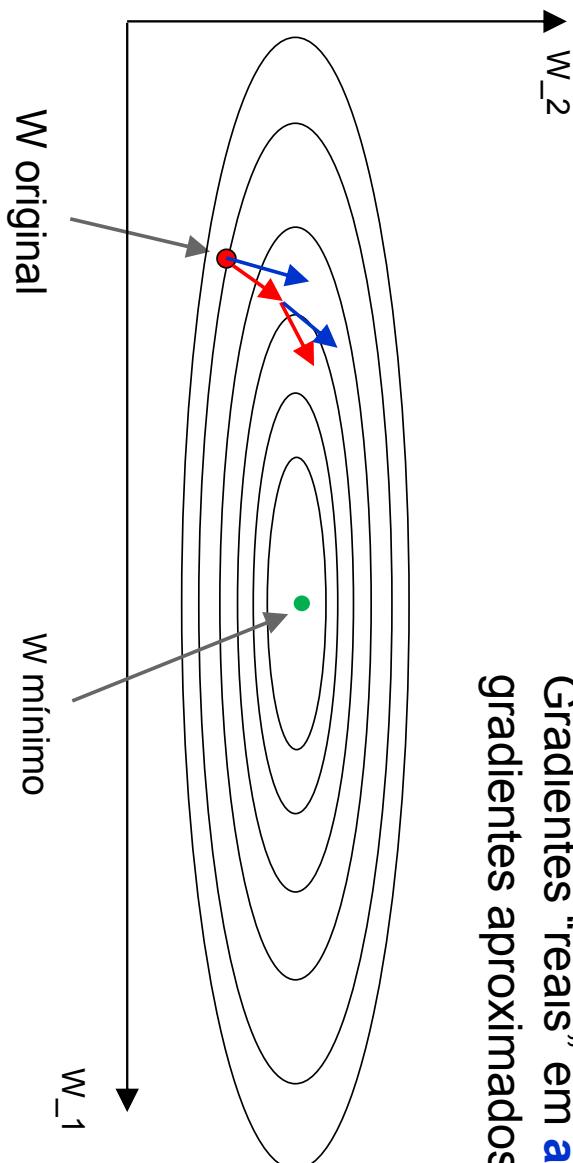
Gradientes “reais” em **azul** e  
gradientes aproximados em **vermelho**



# Gradiente Descendente Estocástico (*SGD*)

Gradientes apresentam “ruídos” mas ainda são bons para progredir na média em direção ao mínimo

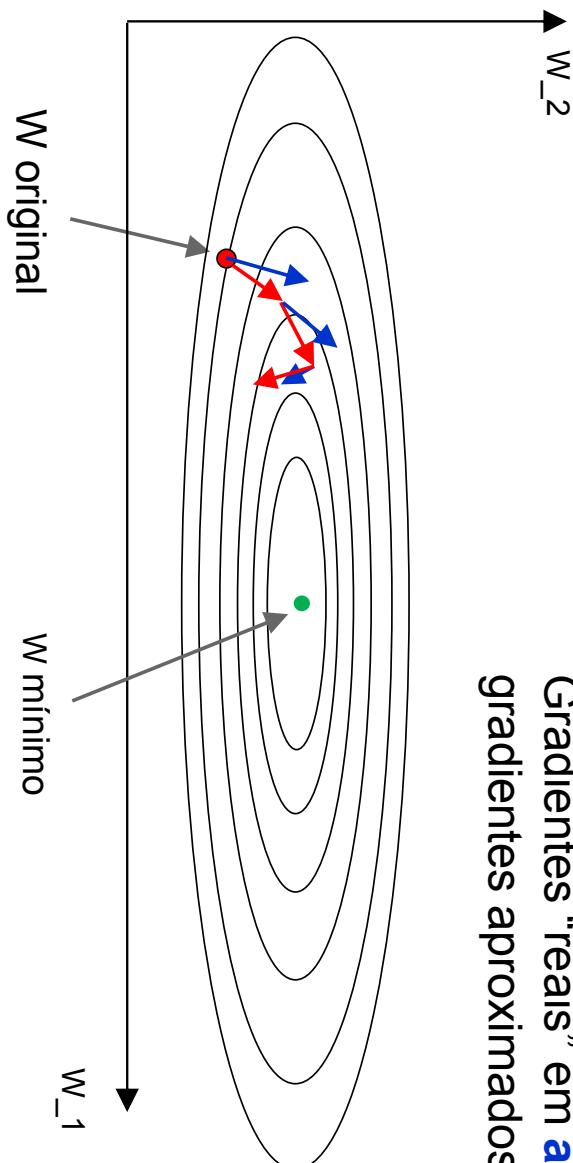
Gradientes “reais” em **azul** e  
gradientes aproximados em **vermelho**



# Gradiente Descendente Estocástico (*SGD*)

Gradientes apresentam “ruídos” mas ainda são bons para progredir na média em direção ao mínimo

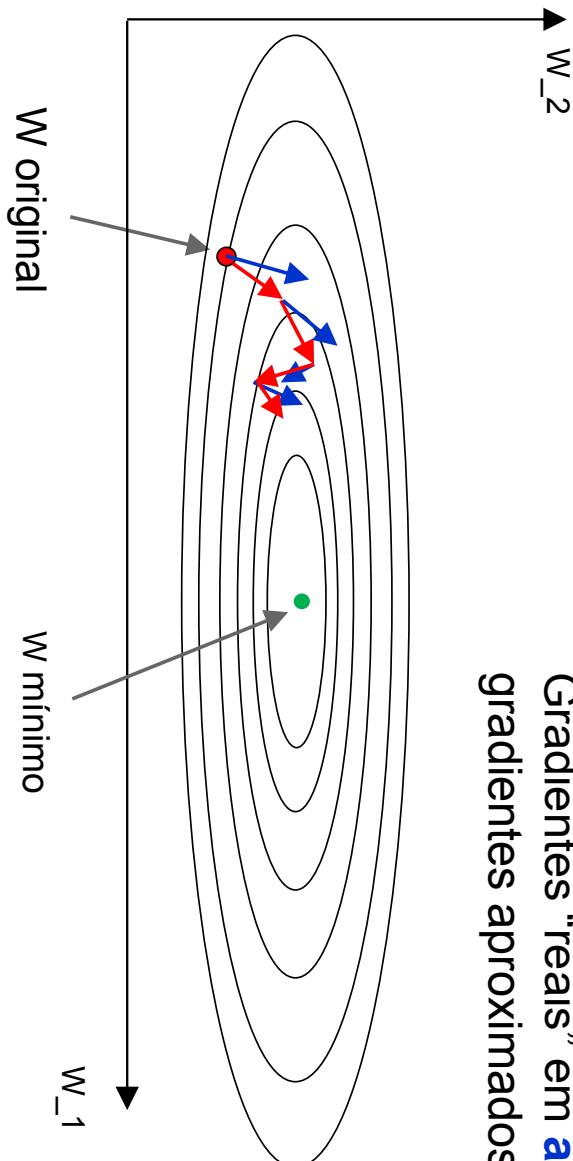
Gradientes “reais” em **azul** e  
gradientes aproximados em **vermelho**



# Gradiente Descendente Estocástico (*SGD*)

Gradientes apresentam “ruídos” mas ainda são bons para progredir na média em direção ao mínimo

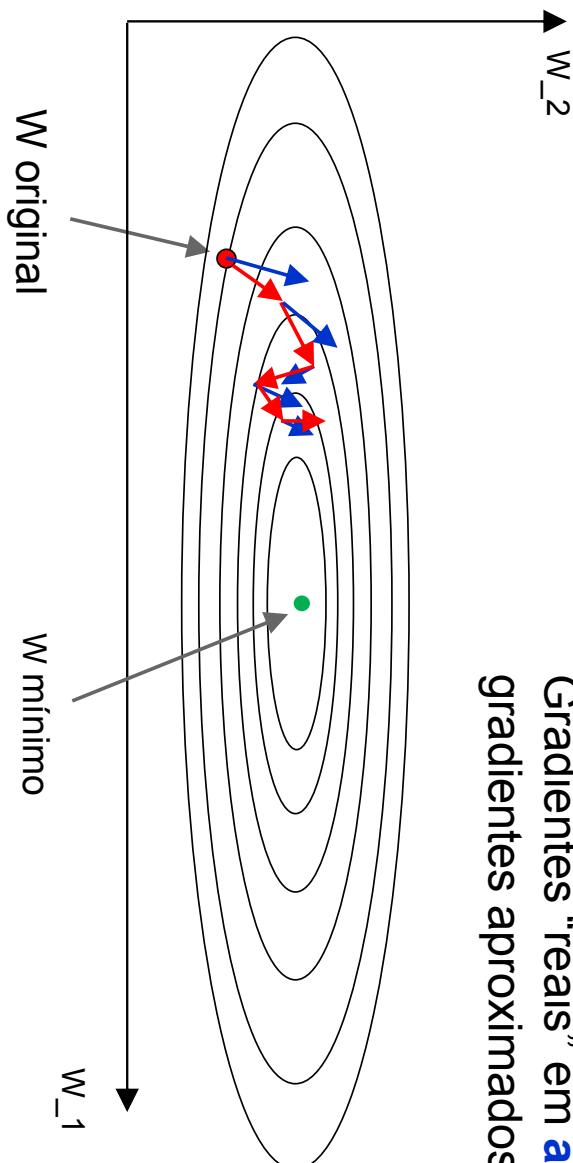
Gradientes “reais” em **azul** e  
gradientes aproximados em **vermelho**



# Gradiente Descendente Estocástico (*SGD*)

Gradientes apresentam “ruídos” mas ainda são bons para progredir na média em direção ao mínimo

Gradientes “reais” em **azul** e  
gradientes aproximados em **vermelho**



# Gradiente Descendente Estocástico ( $SGD$ )

Ideia principal: usar apenas uma pequena amostra do conjunto de treinamento para calcular o gradiente em cada passo

# Gradiente Descendente Estocástico (*SGD*)

Ideia principal: usar apenas uma pequena amostra do conjunto de treinamento para calcular o gradiente em cada passo

```
# Vanilla Minibatch Gradient Descent
```

```
while True:
```

```
    data_batch = sample_training_data(data, 256) # sample 256 examples
```

```
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```

# Gradiente Descendente Estocástico (*SGD*)

Ideia principal: usar apenas uma pequena amostra do conjunto de treinamento para calcular o gradiente em cada passo

```
# Vanilla Minibatch Gradient Descent
```

```
while True:
```

```
    data_batch = sample_training_data(data, 256) # sample 256 examples
```

```
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```

Tamanhos comuns de *minibatches* são amostras de 32, 64, 128, 256, ...

Por exemplo, na AlexNet utilizou-se lotes com 256 amostras

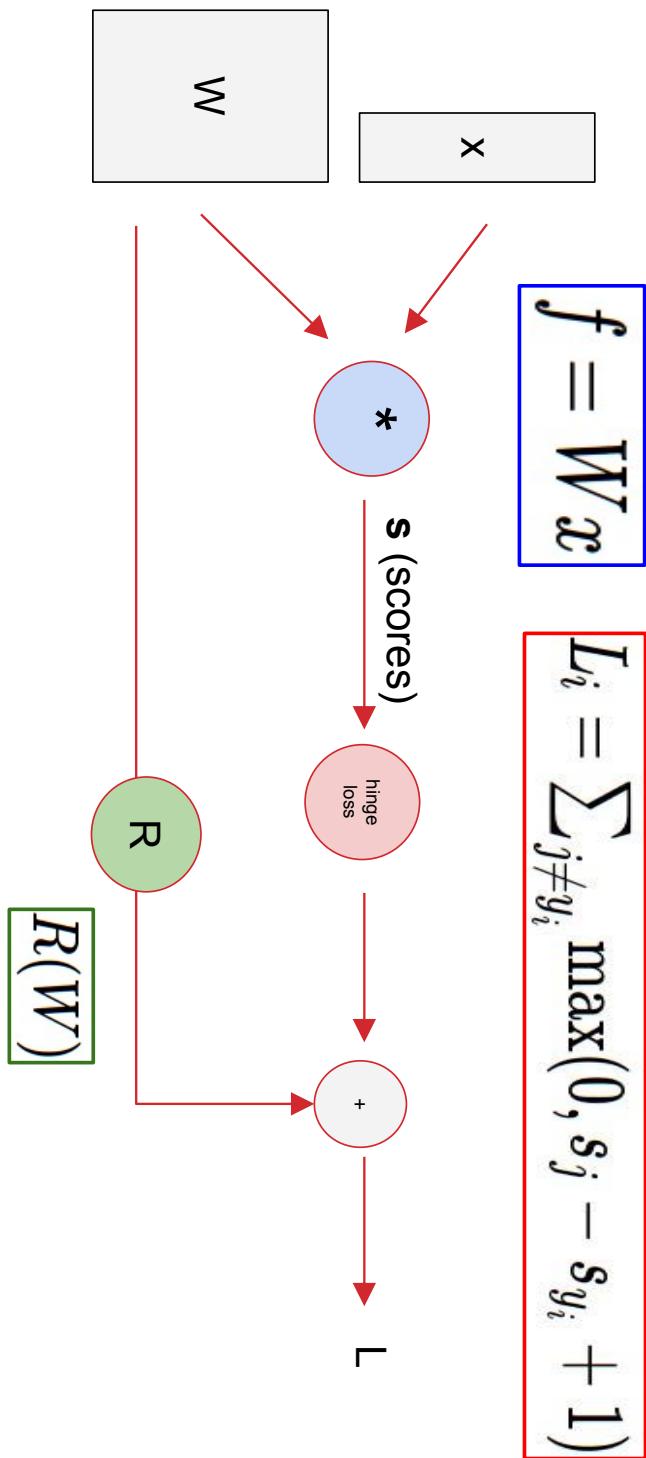
# Redes Neurais e Aprendizagem Profunda

## REDES NEURAIS ARTIFICIAIS PROPAGAÇÃO RETRÓGRADA (I)

Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

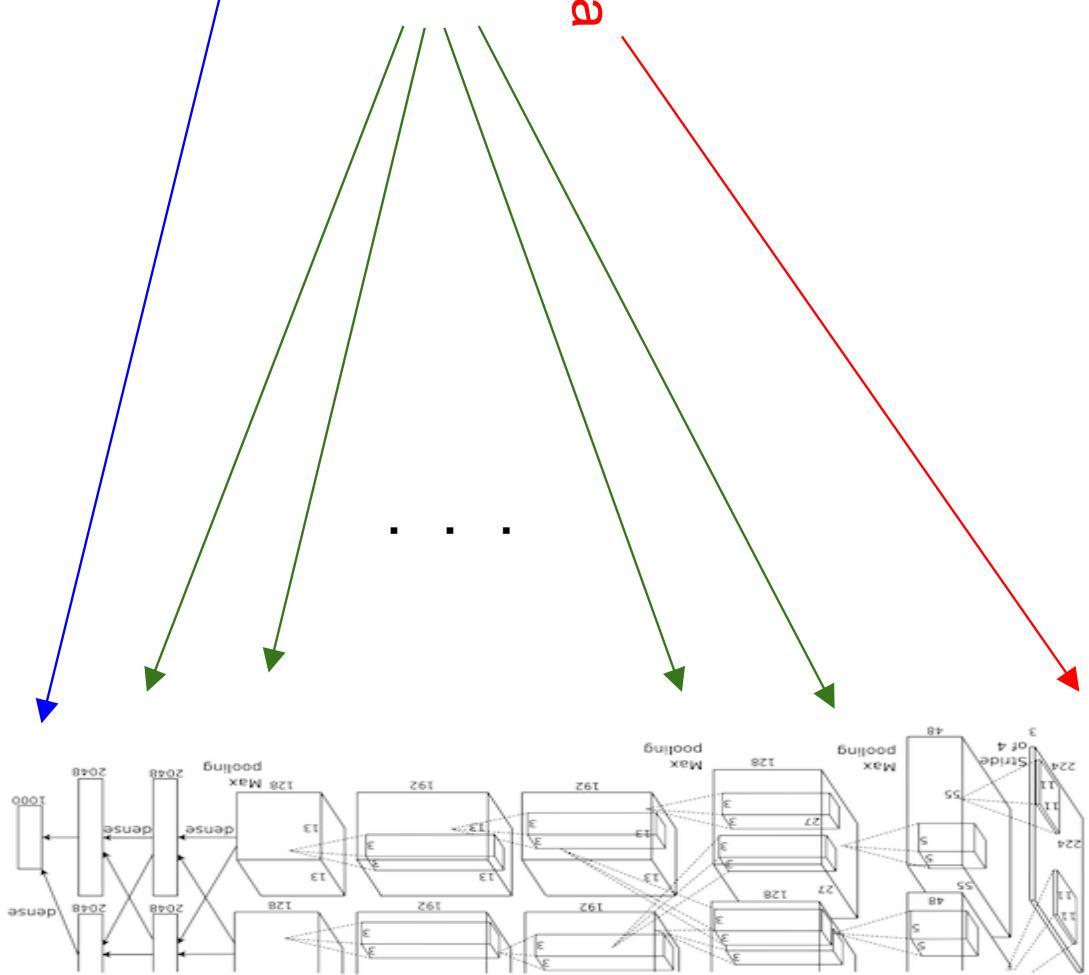
# Grafo de Computação da Função de Perda



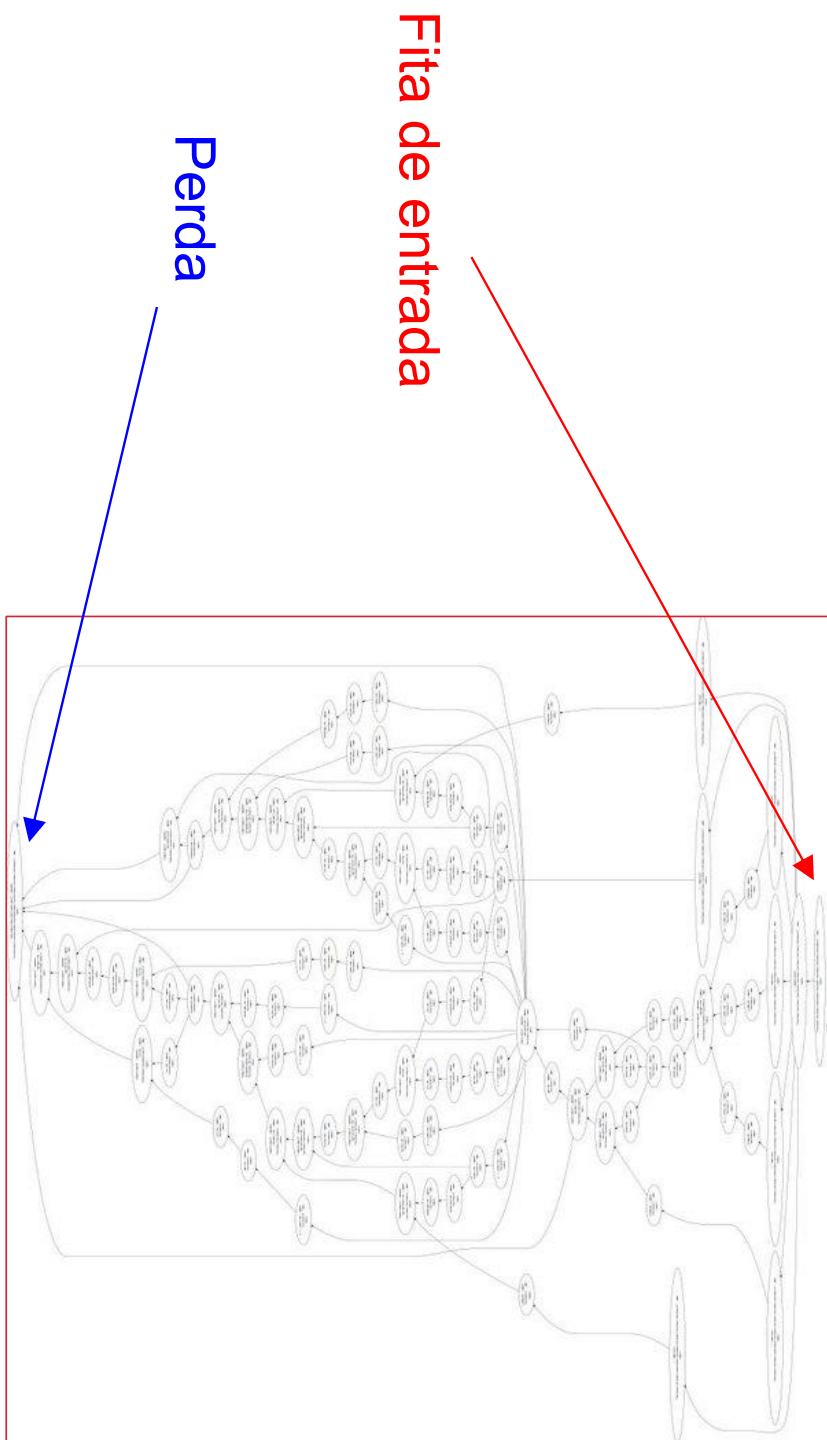
# Rede Convolucional (AlexNet)

Imagen de entrada

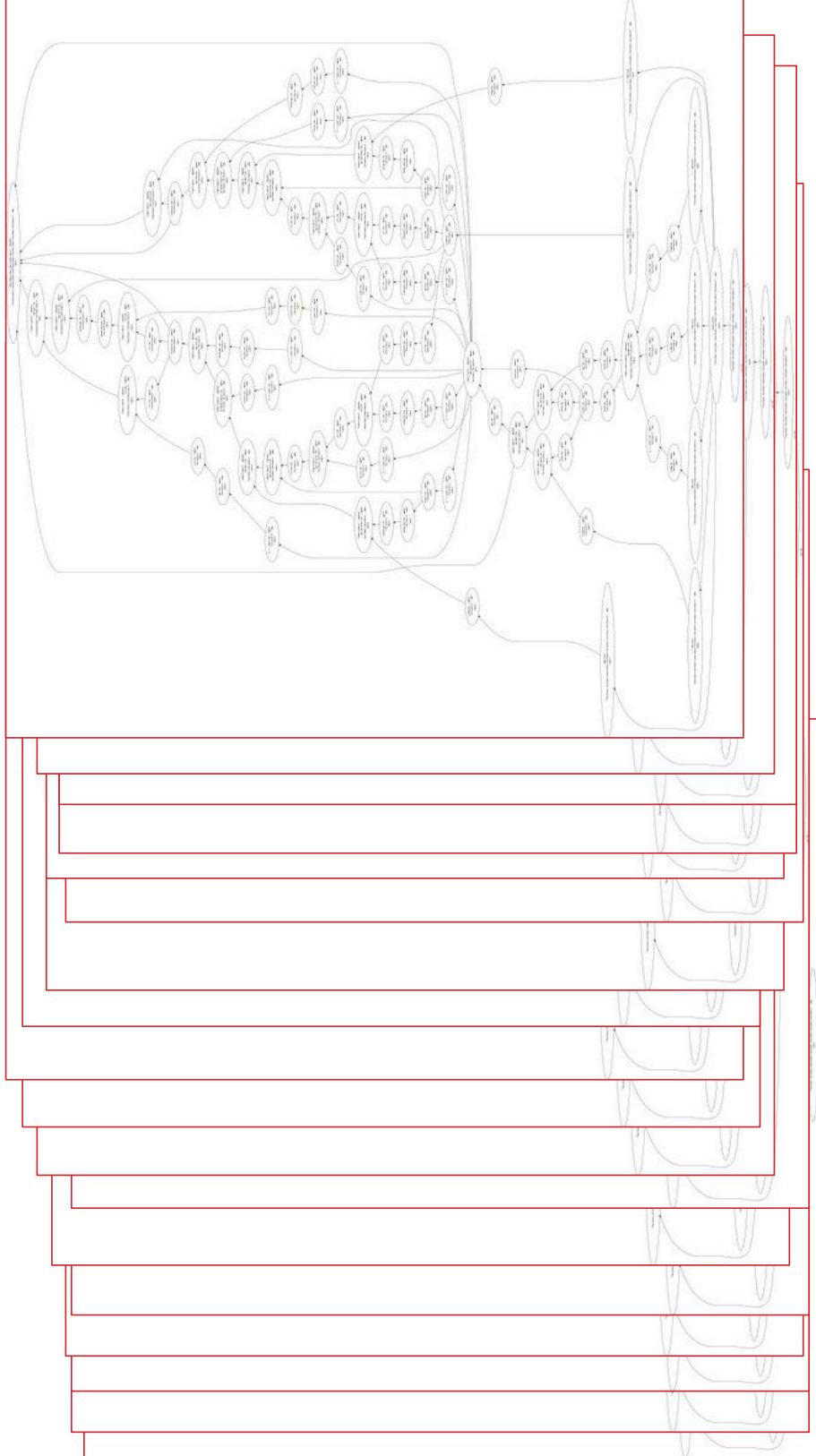
Pesos  
Perda



# Máquina Neural de Turing

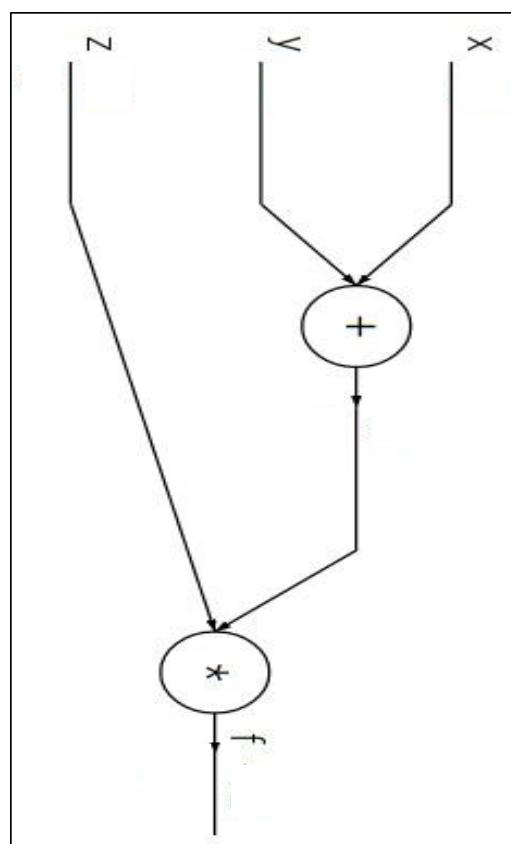


# Neural Turing Machine



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

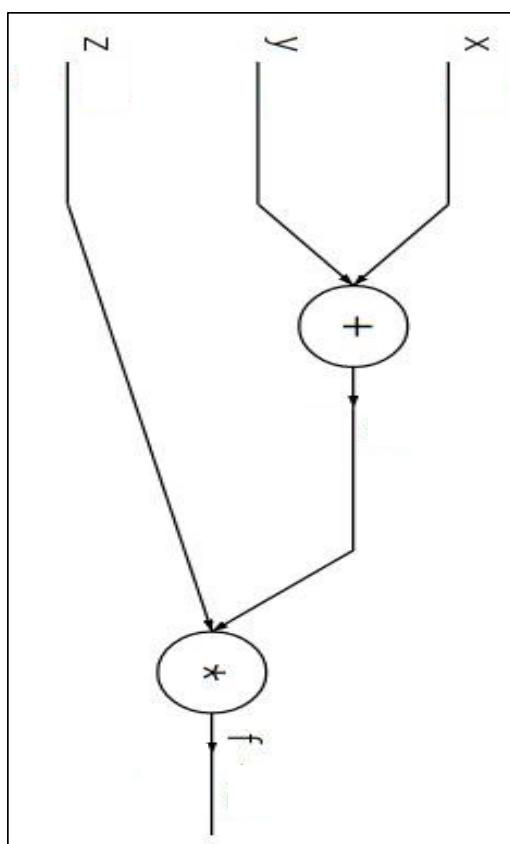


# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Então, pode-se dizer que

$$f(x, y, z) = g(h(x, y), z)$$



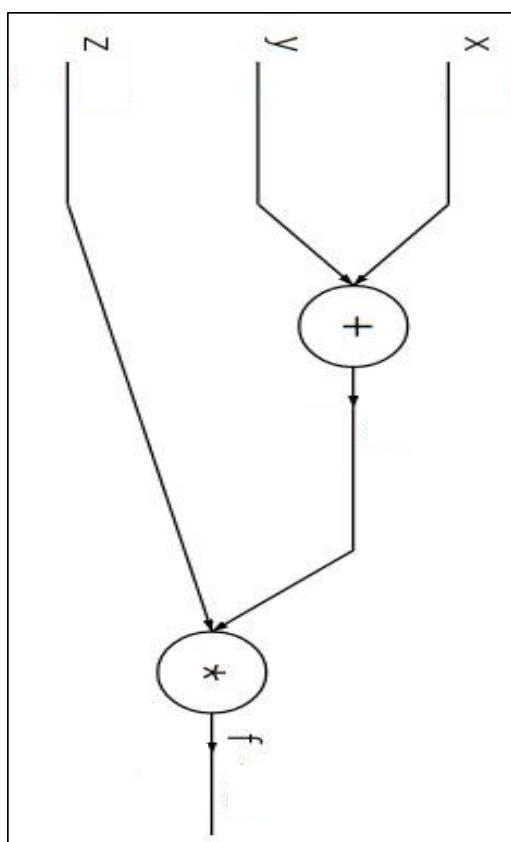
# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Então, pode-se dizer que

$$f(x, y, z) = g(h(x, y), z)$$

em  $h(x, y) = x + y$



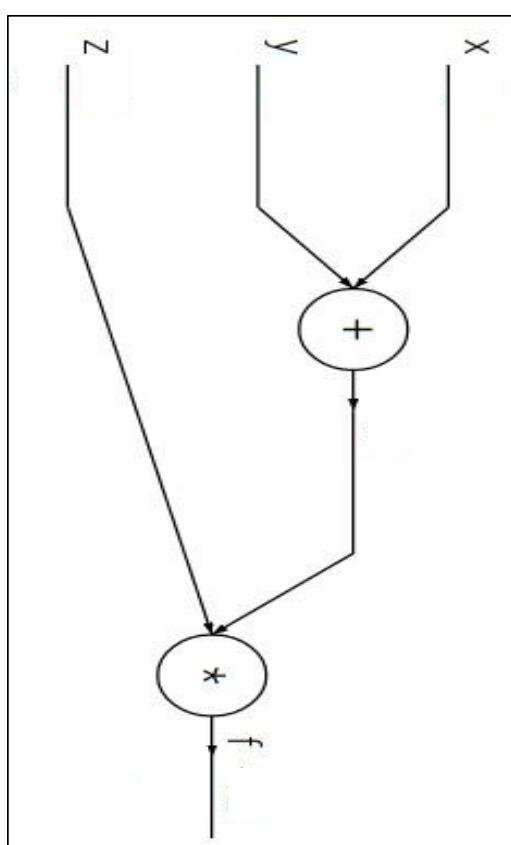
# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Então, pode-se dizer que

$$f(x, y, z) = g(h(x, y), z)$$

em  $h(x, y) = x + y$  e  $g(a, b) = a \times b$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

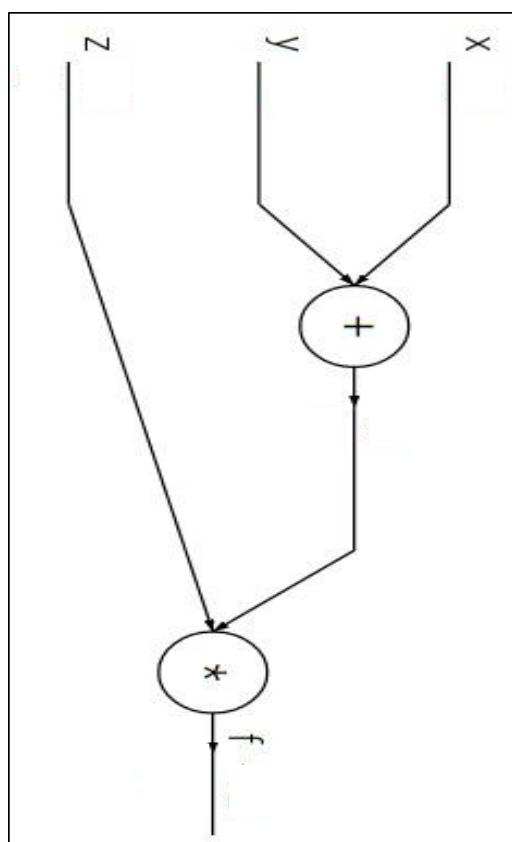
Então, pode-se dizer que

$$f(x, y, z) = g(h(x, y), z)$$

em  $h(x, y) = x + y$  e  $g(a, b) = a \times b$

Pela **regra da cadeia**, sabe-se que

$$\frac{df}{dx} = \frac{dg}{dh} \times \frac{dh}{dx}$$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Então, pode-se dizer que

$$f(x, y, z) = g(h(x, y), z)$$

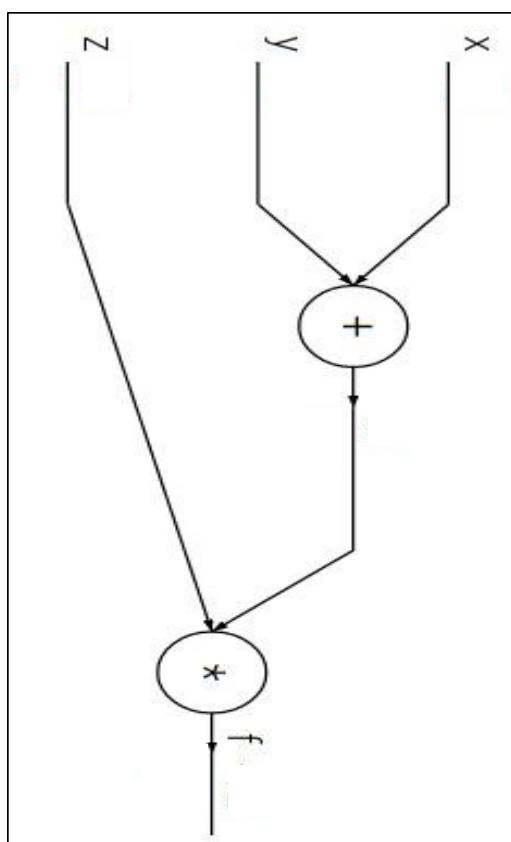
em  $h(x, y) = x + y$  e  $g(a, b) = a \times b$

Pela **regra da cadeia**, sabe-se que

$$\frac{df}{dx} = \frac{dg}{dh} \times \frac{dh}{dx}$$

e

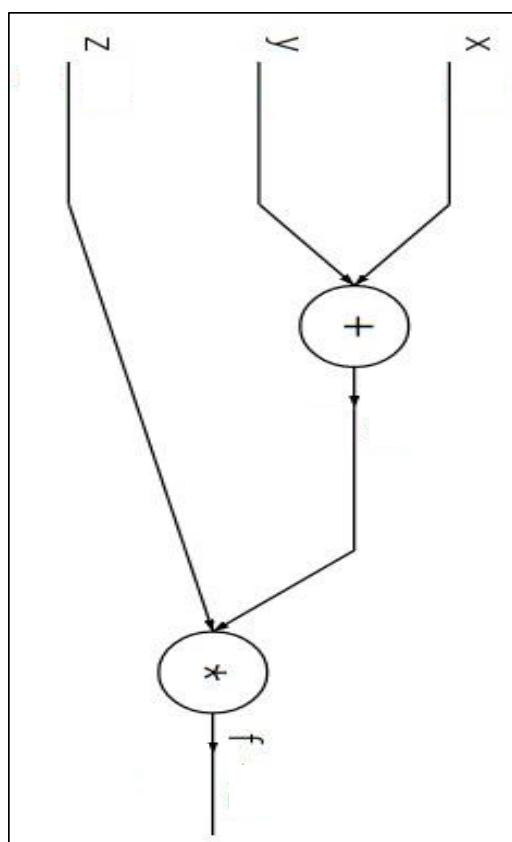
$$\frac{df}{dy} = \frac{dg}{dh} \times \frac{dh}{dy}$$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

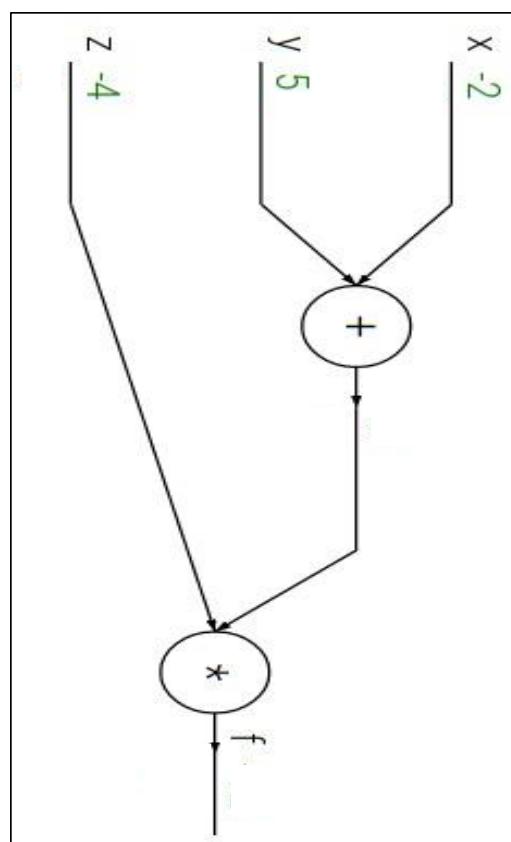
Por exemplo:  $x = -2, y = 5, z = -4$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

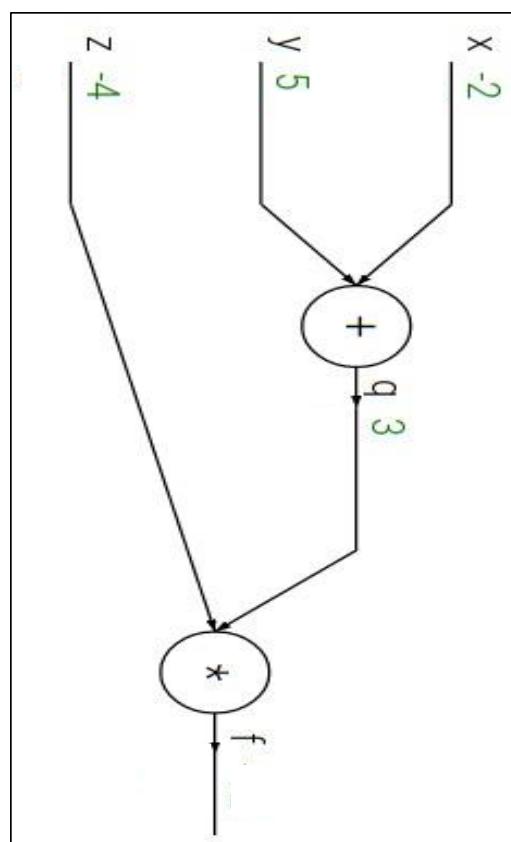


# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y$$



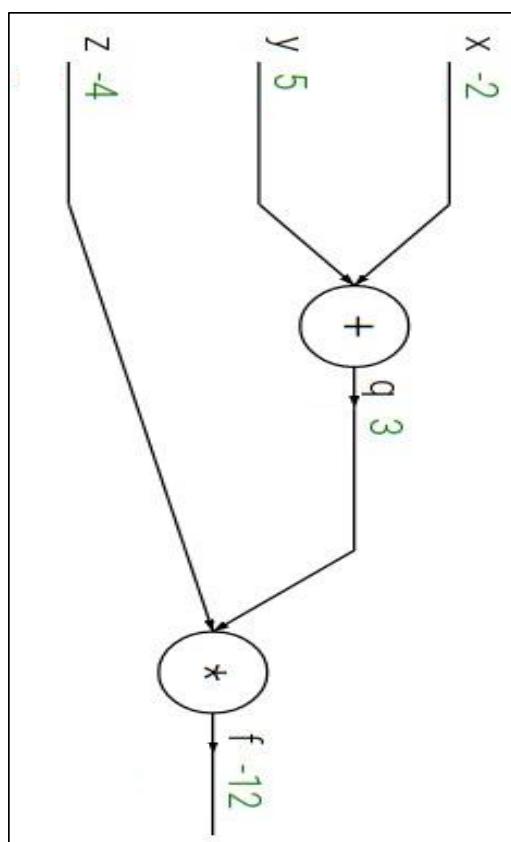
# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y$$

$$f = qz$$



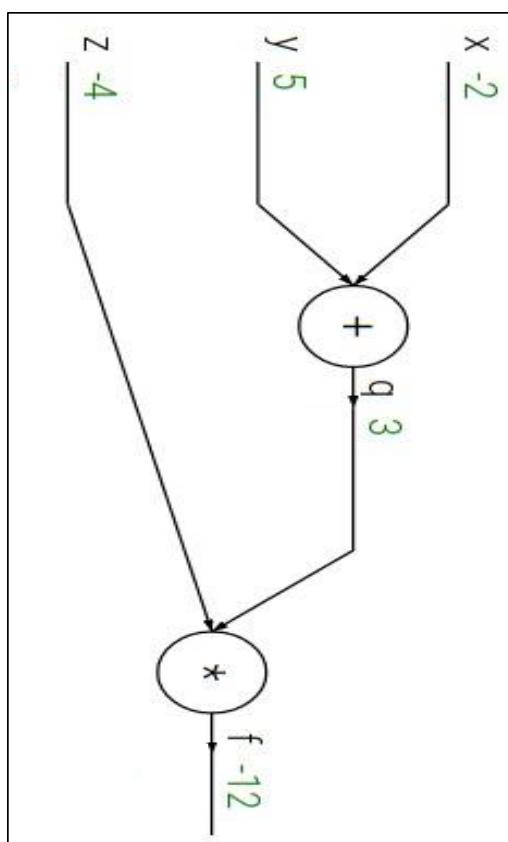
# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz$$



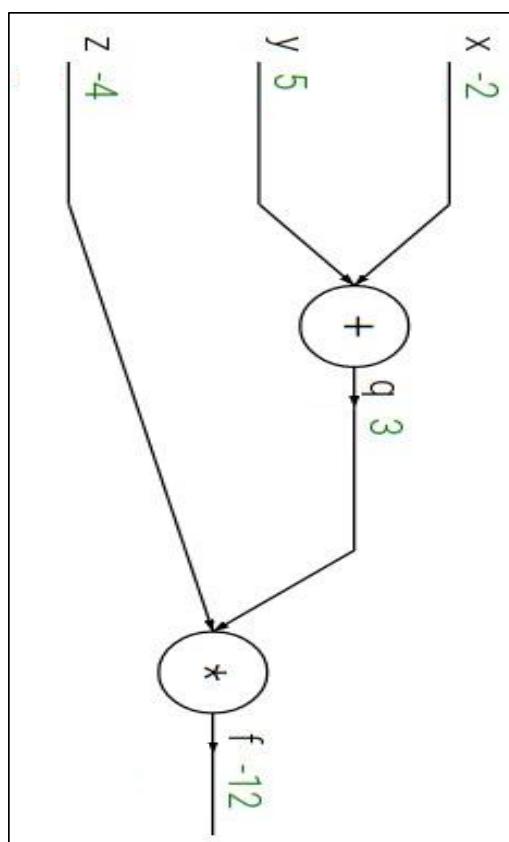
# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



# Diferenciação de um Grafo de Computação

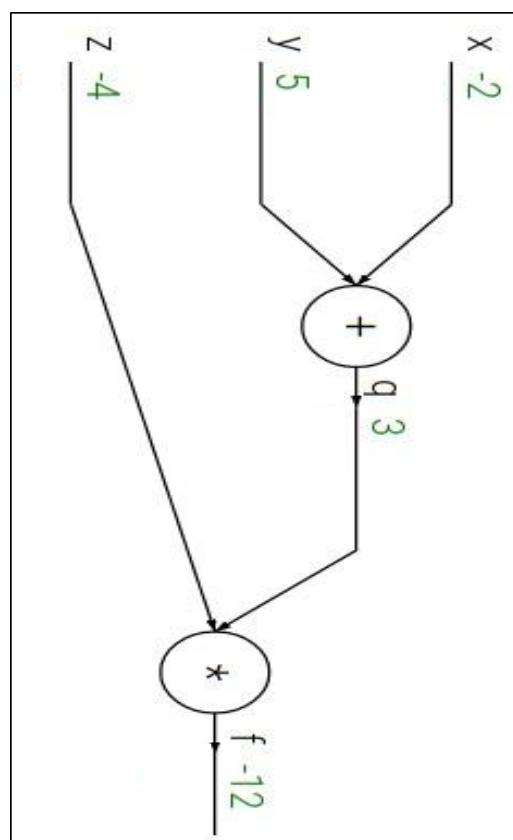
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

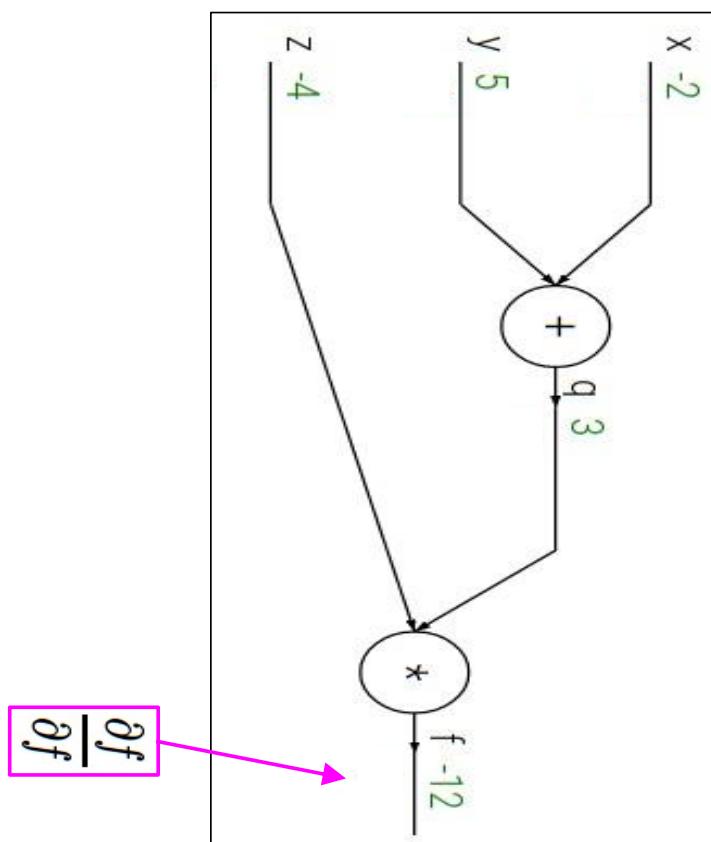
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

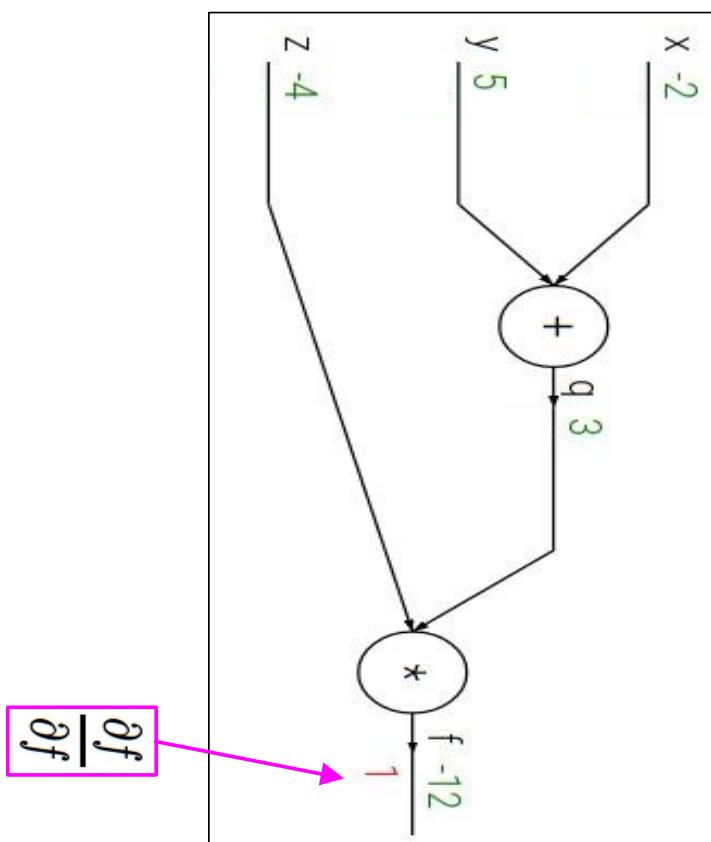
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

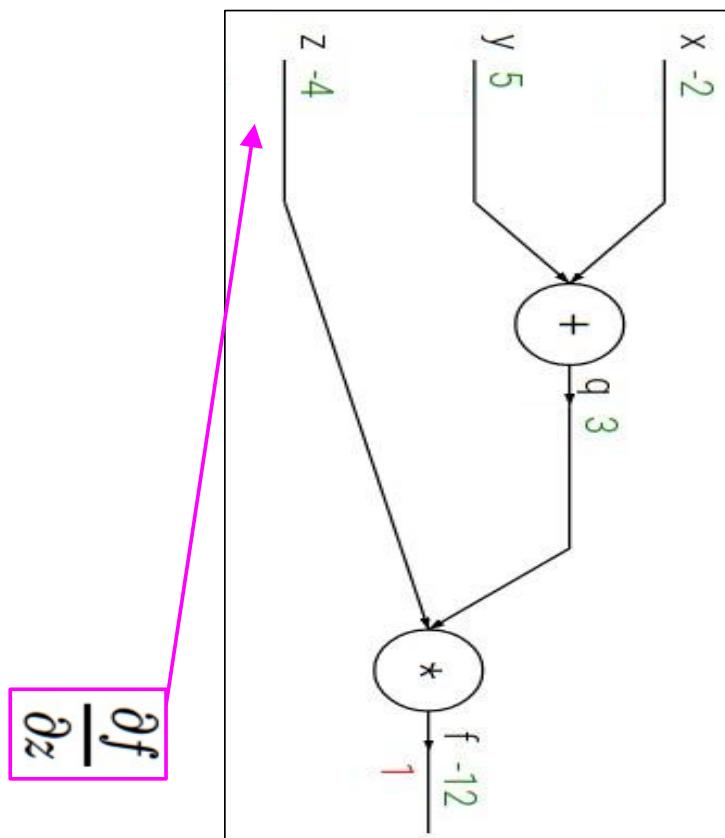
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

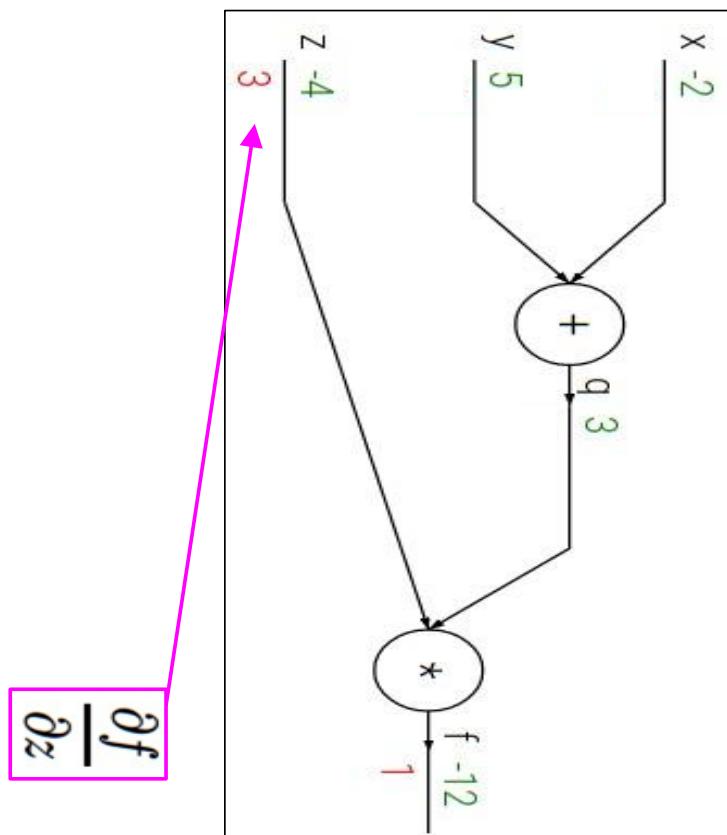
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

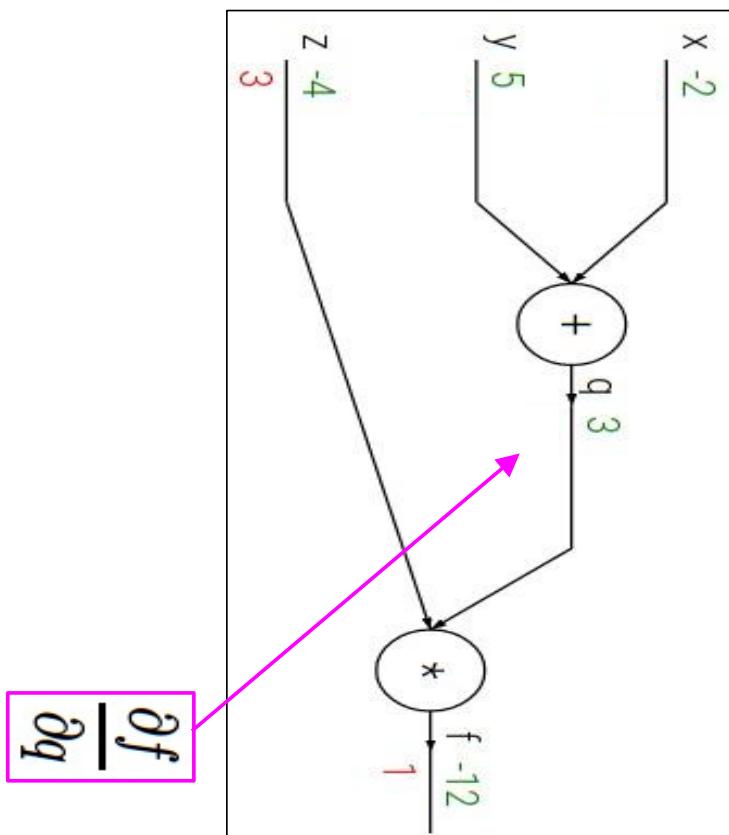
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

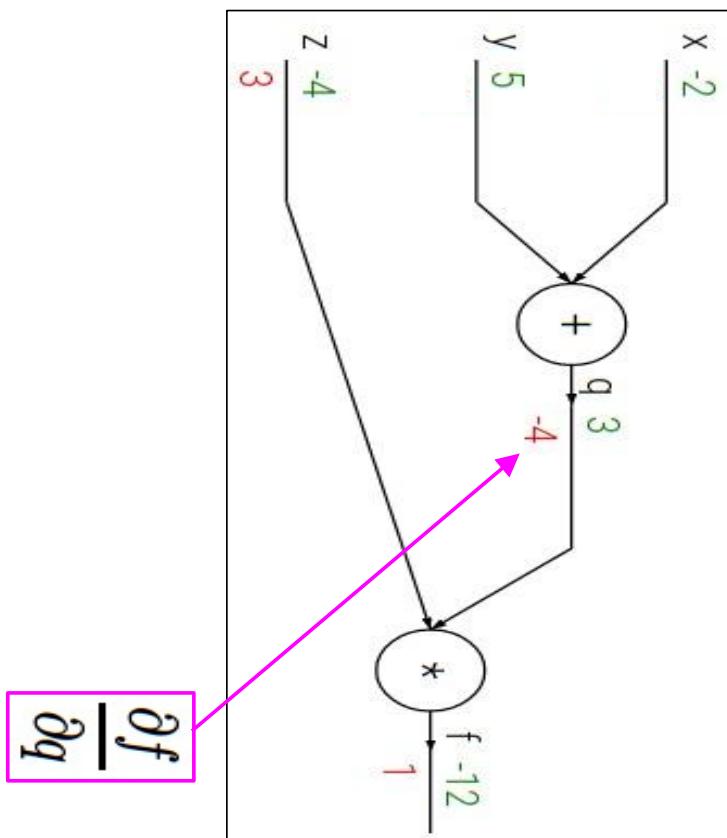
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

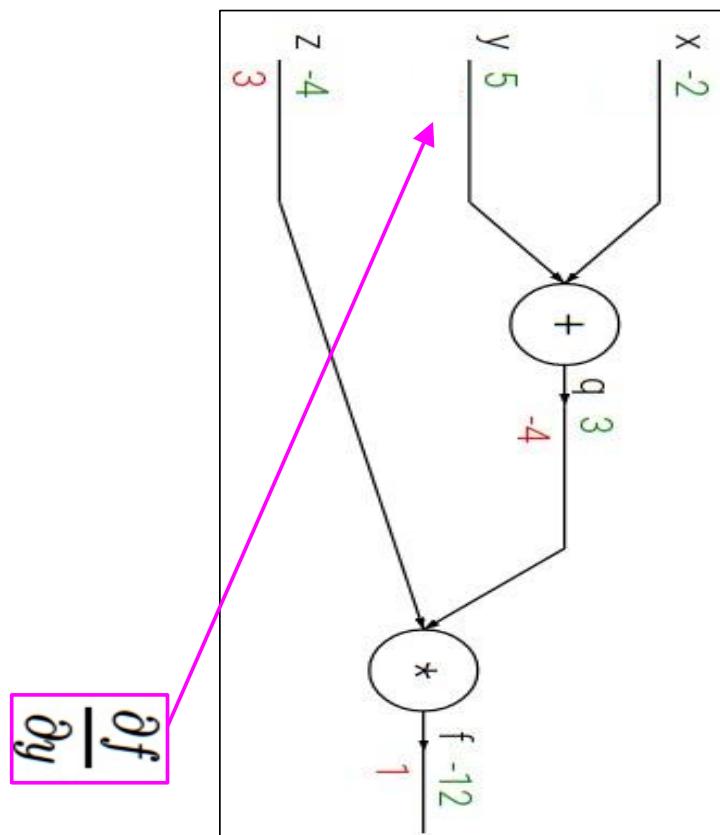
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

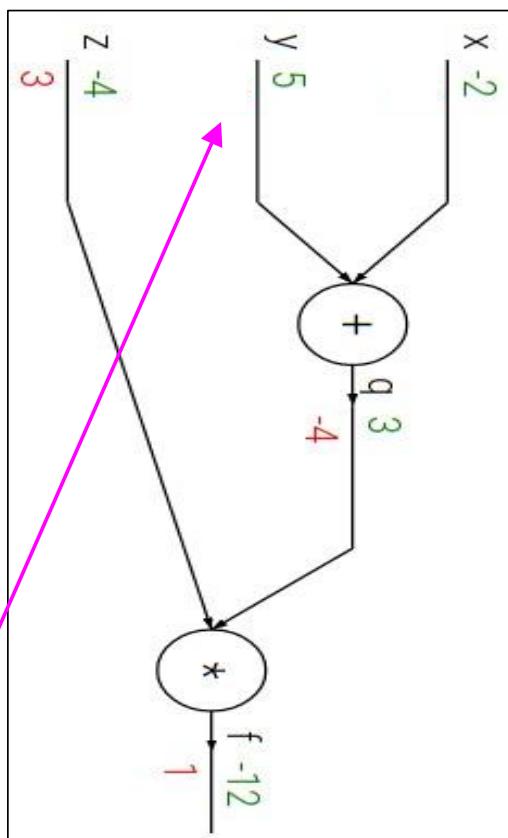
$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Regra da Cadeia:

$$\frac{\partial f}{\partial y}$$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

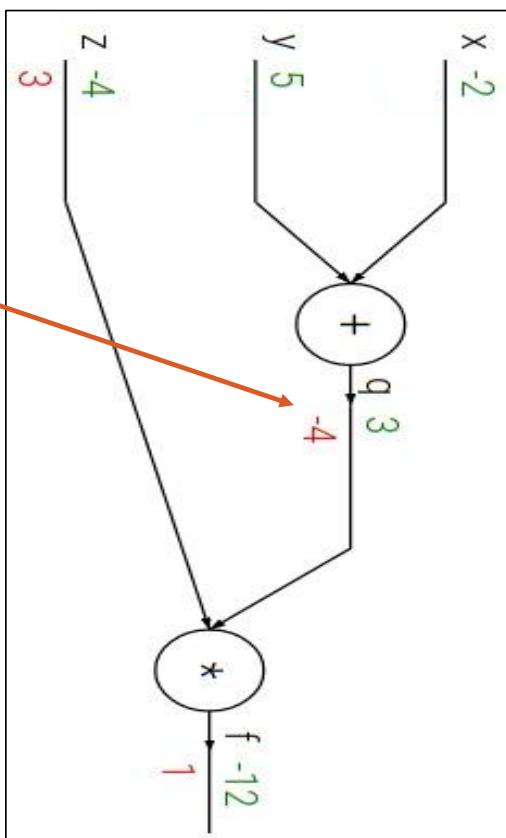
$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Regra da Cadeia:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

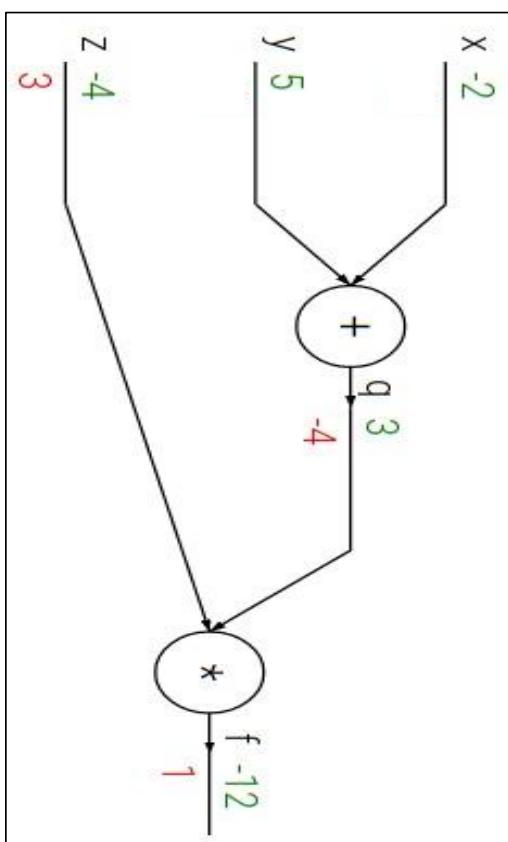
$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Regra da Cadeia:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

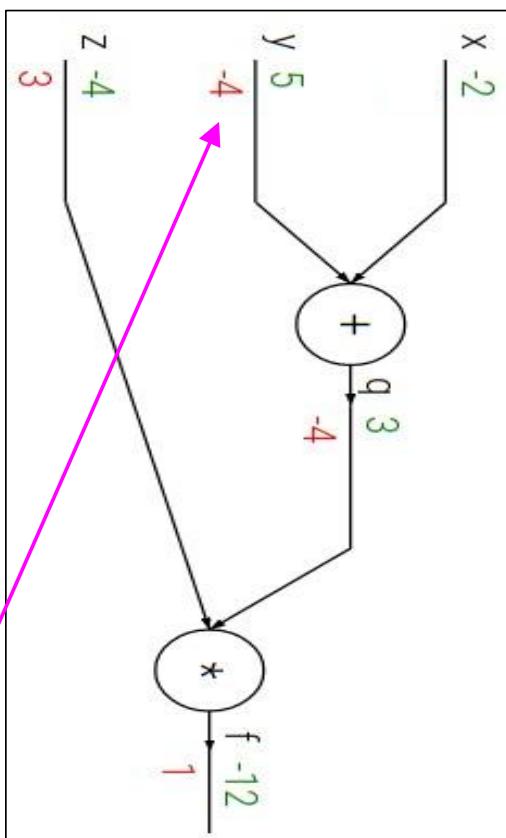
$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Regra da Cadeia:

$$\frac{\partial f}{\partial y}$$



# Diferenciação de um Grafo de Computação

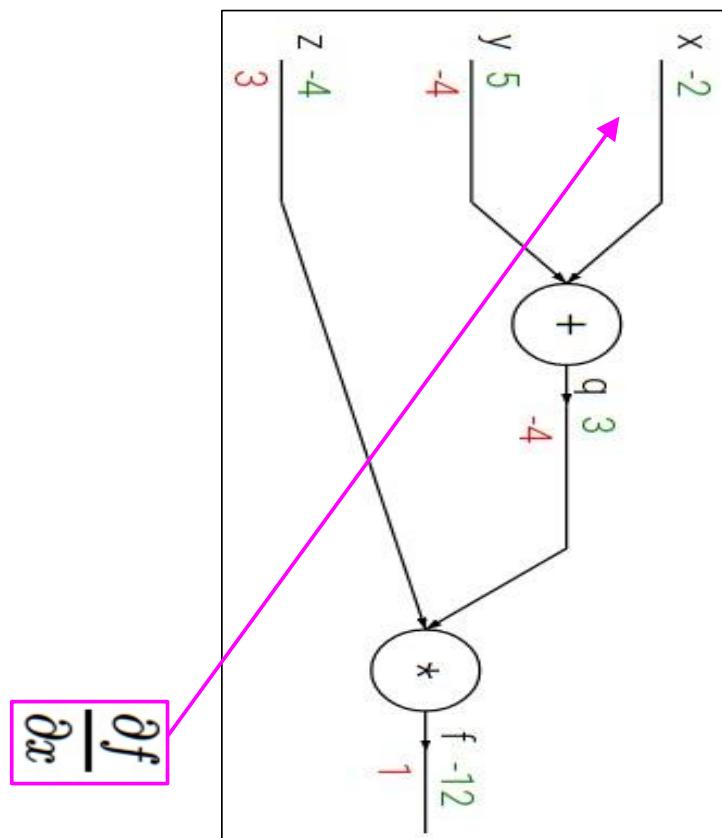
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

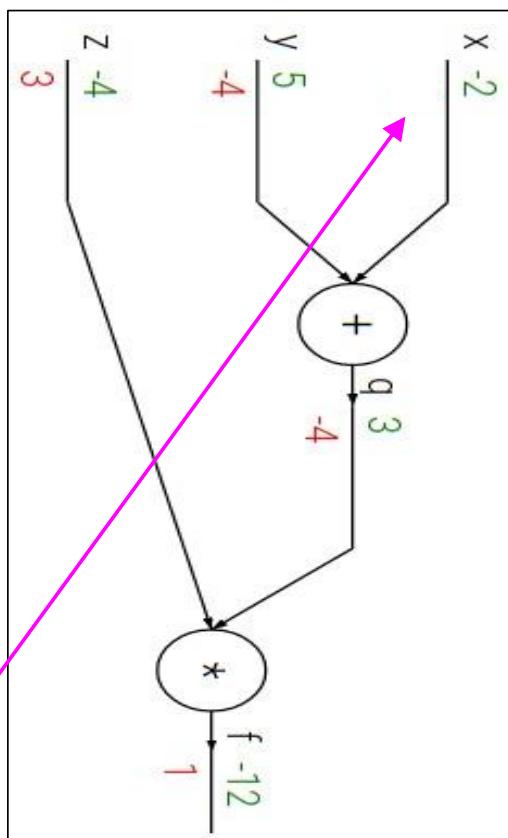
$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Regra da Cadeia:

$$\frac{\partial f}{\partial x}$$



# Diferenciação de um Grafo de Computação

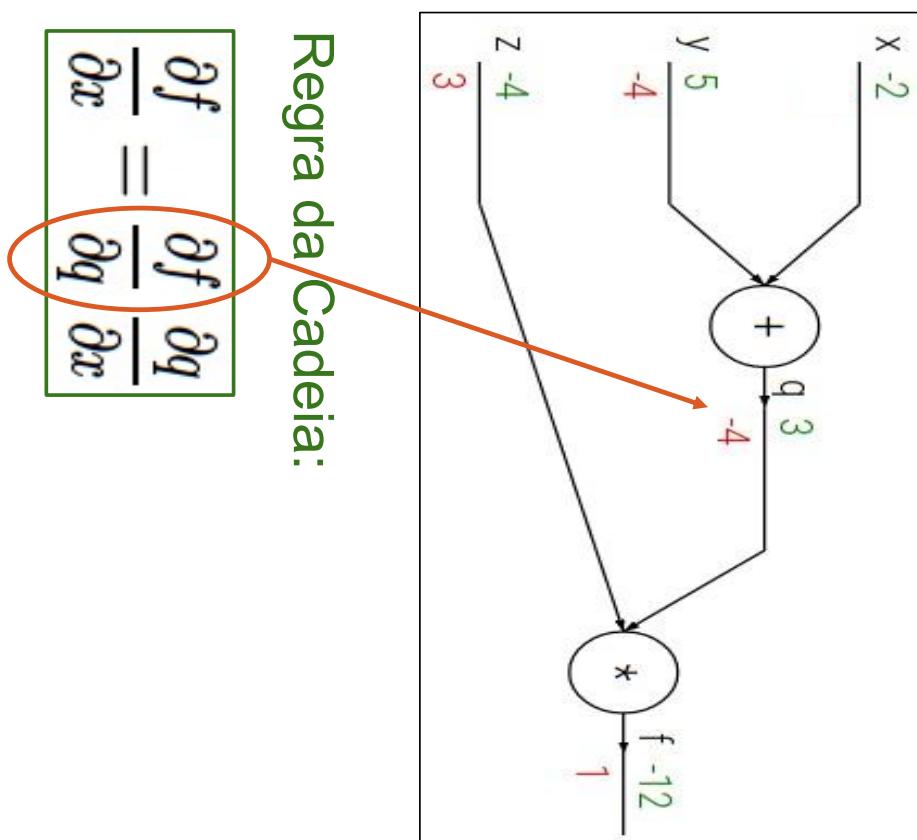
Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Regra da Cadeia:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

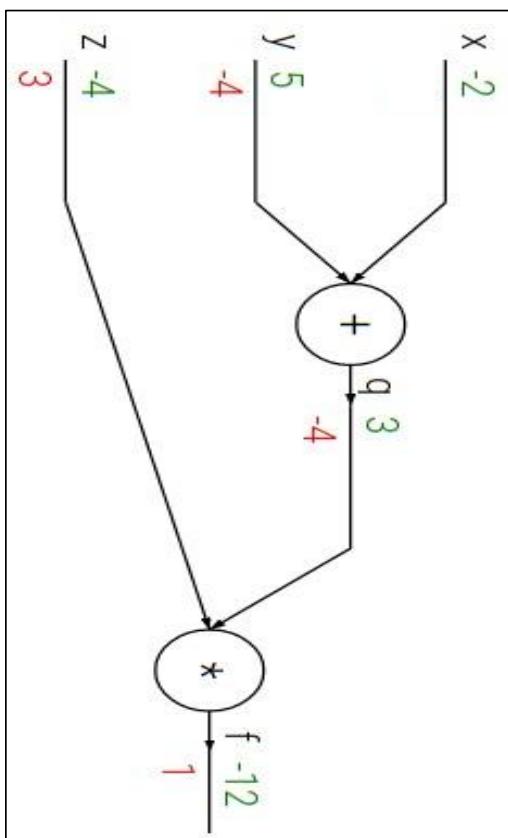
$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Regra da Cadeia:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



# Diferenciação de um Grafo de Computação

Seja  $f(x, y, z) = (x + y) \times z$

Por exemplo:  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

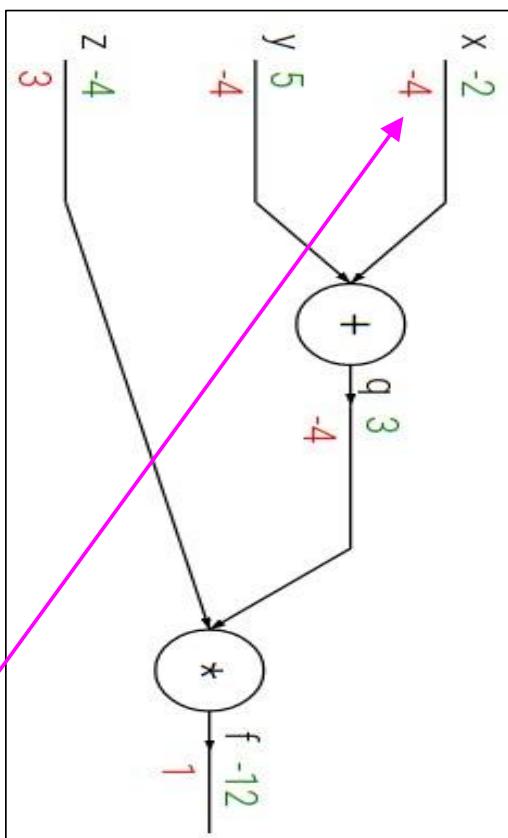
$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Deseja-se:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

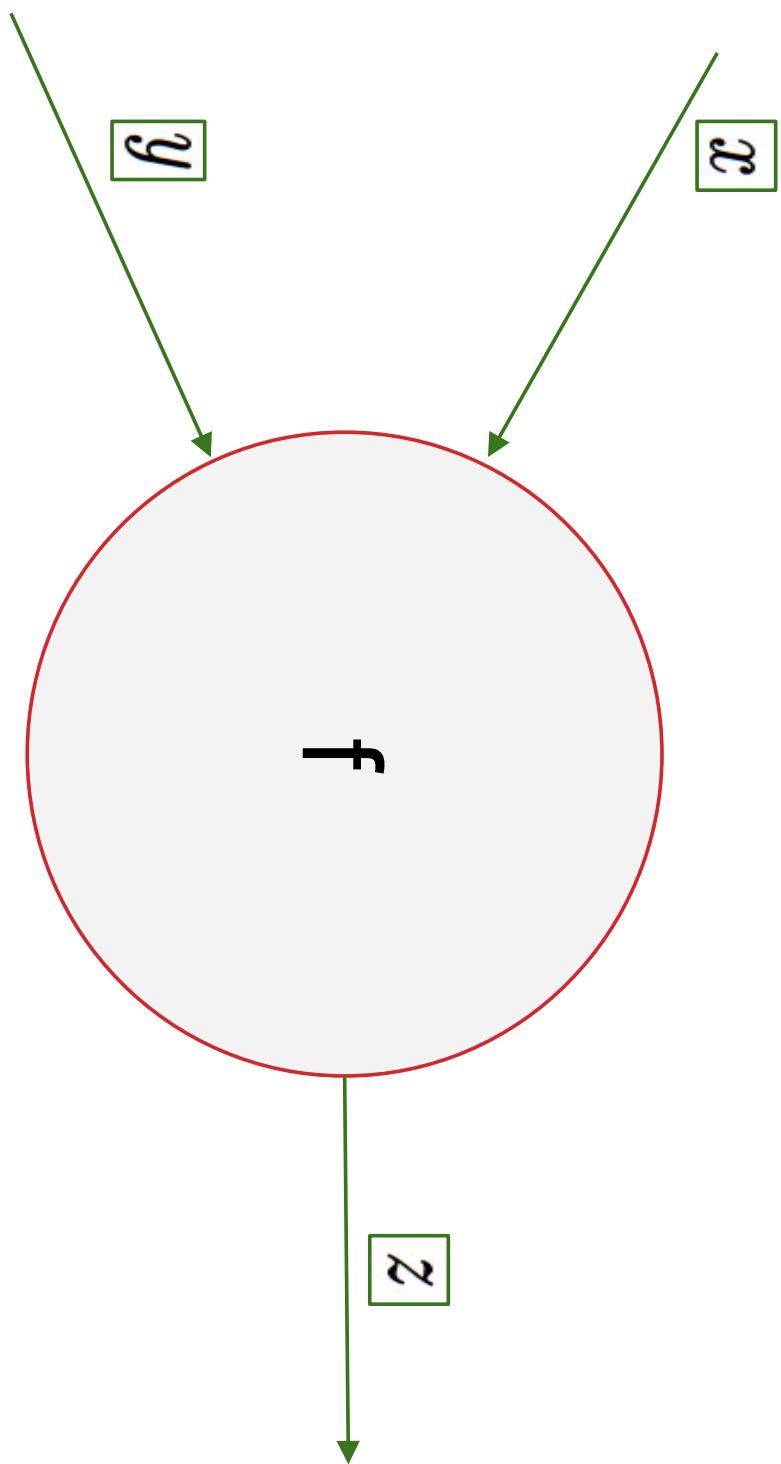
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Regra da Cadeia:

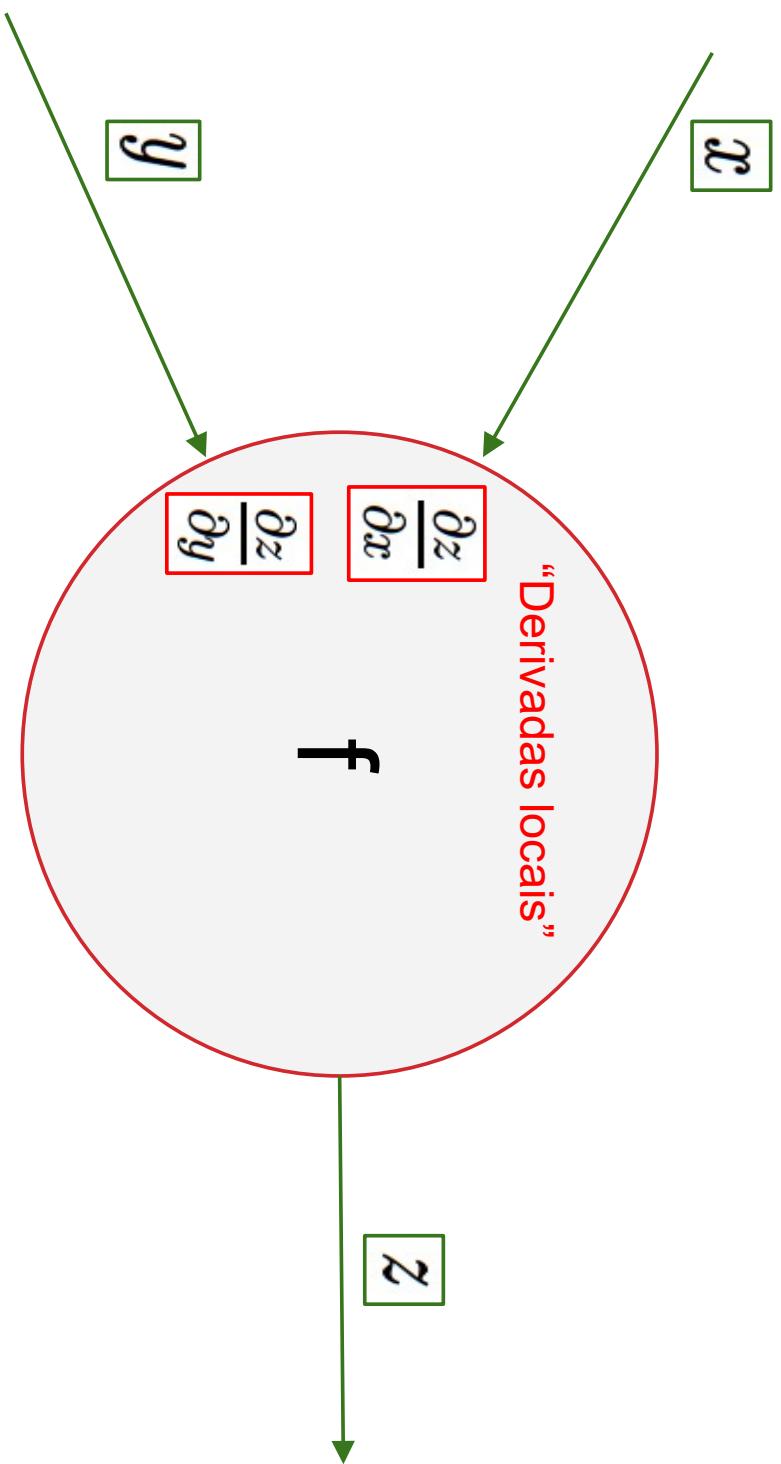
$$\frac{\partial f}{\partial x}$$



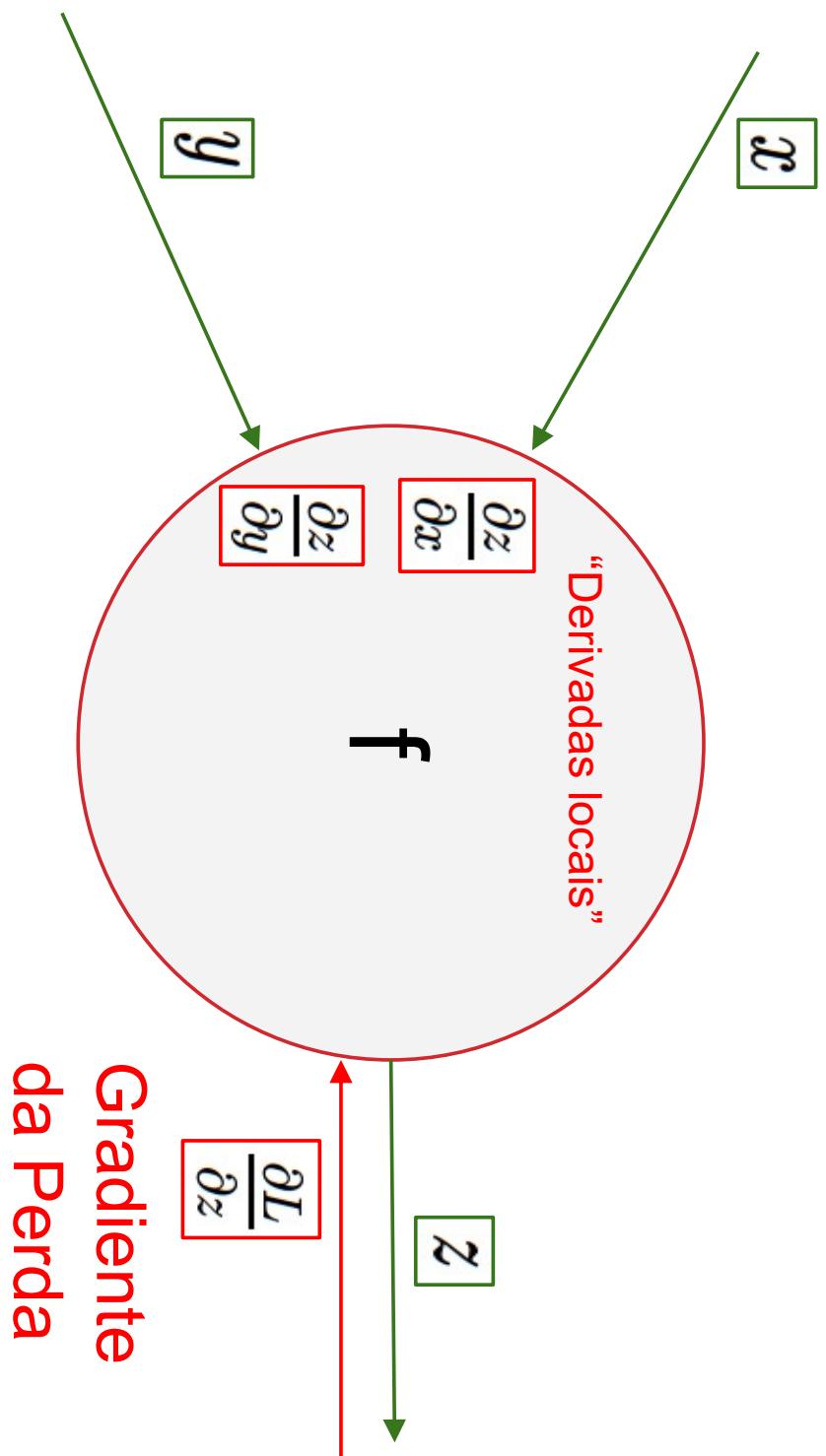
# Passo Retrógrado (*Backward Pass*)



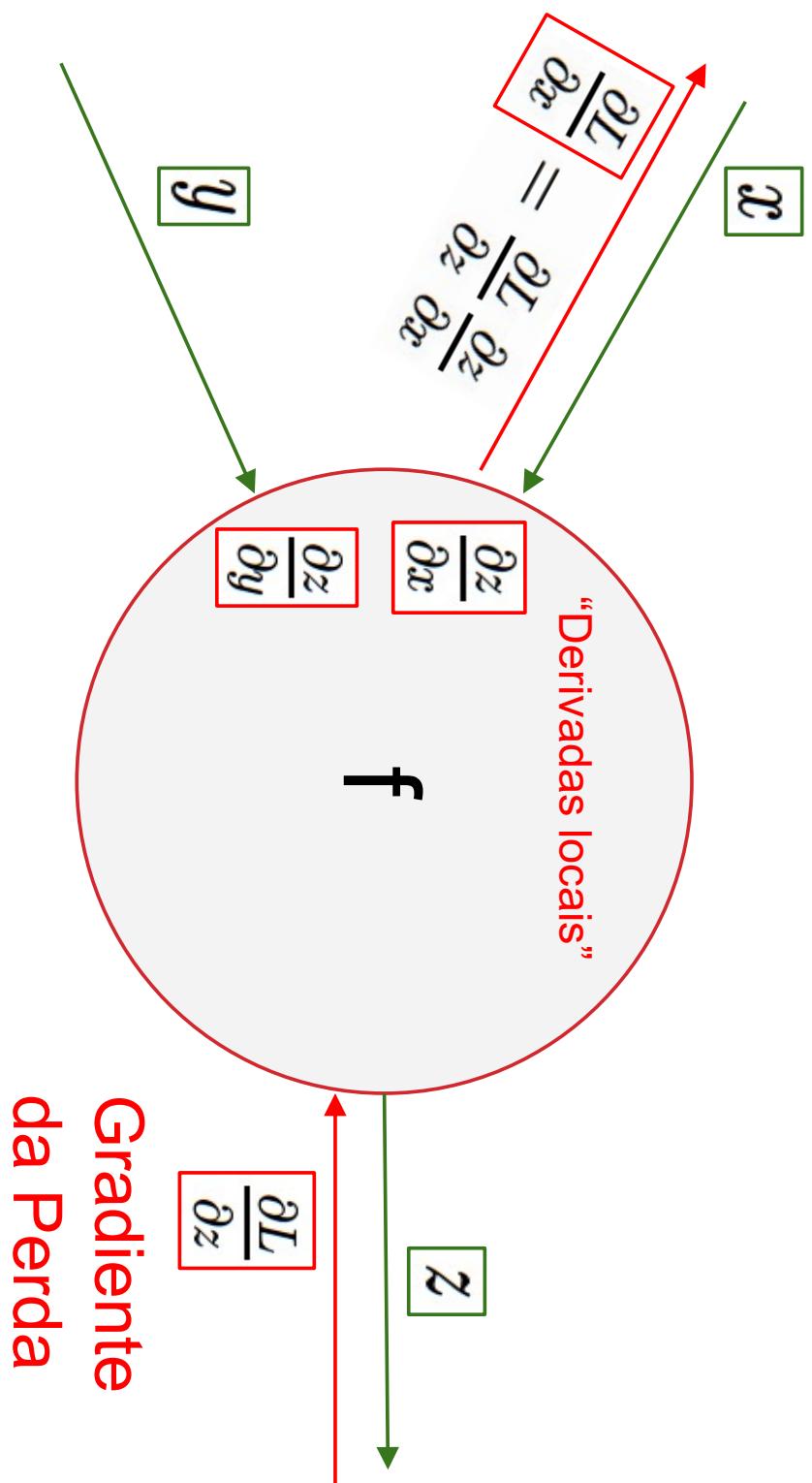
# Passo Retrógrado (*Backward Pass*)



# Passo Retrógrado (*Backward Pass*)

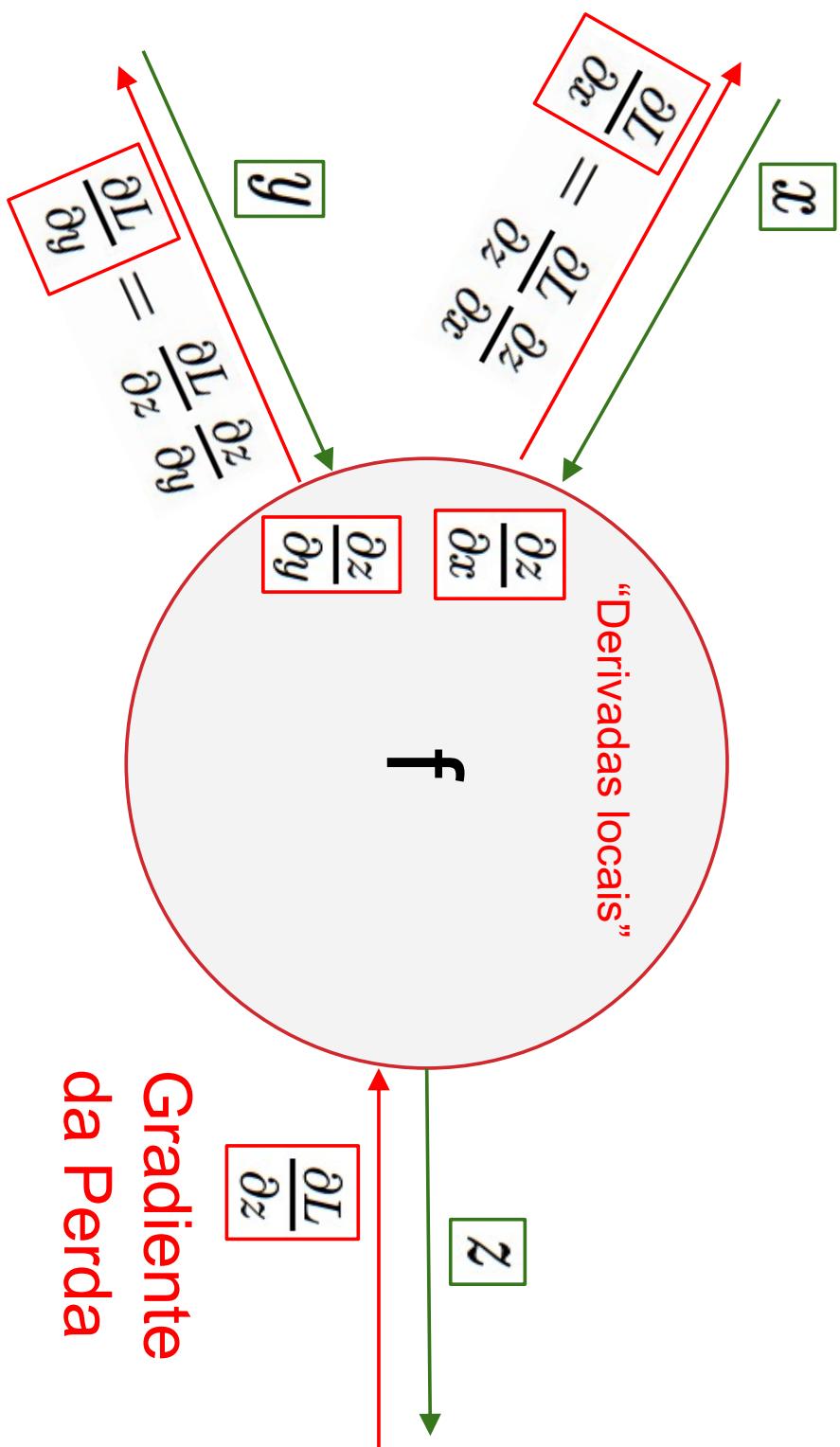


# Passo Retrógrado (*Backward Pass*)



# Passo Retrógrado (*Backward Pass*)

Gradiente  
da Perda



# Passo Retrógrado (Backward Pass)

Gradiente  
da Perda

$$\frac{\partial L}{\partial z}$$

$z$

“Derivadas locais”

$f$

$$\frac{\partial z}{\partial x}$$

$$\frac{\partial z}{\partial y}$$

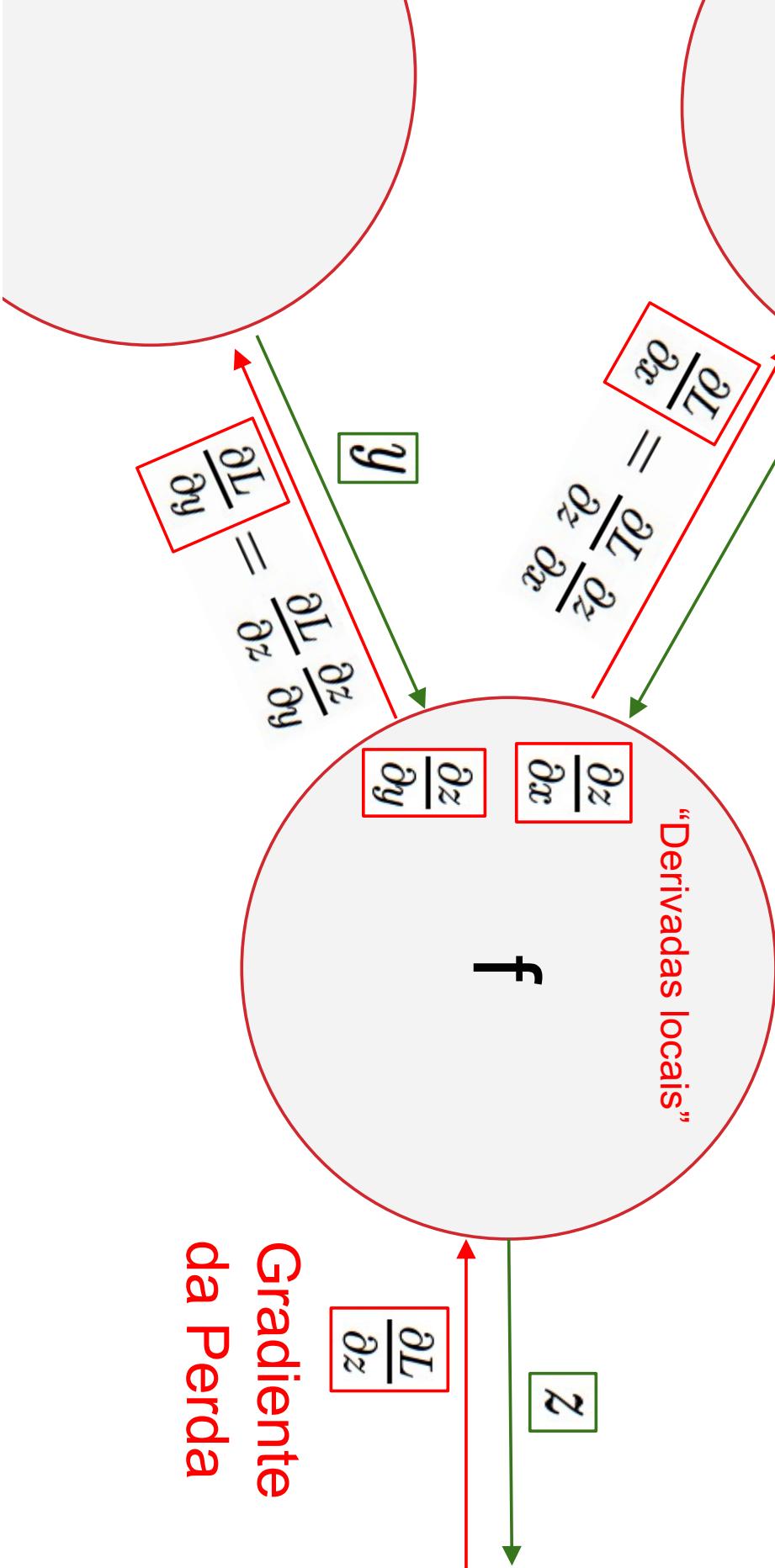
$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}$$

$$\frac{\partial L}{\partial x} =$$

$$\frac{\partial L}{\partial x}$$

$y$

$x$



# Redes Neurais e Aprendizagem Profunda

## REDES NEURAIS ARTIFICIAIS PROPAGAÇÃO RETRÓGRADA (II)

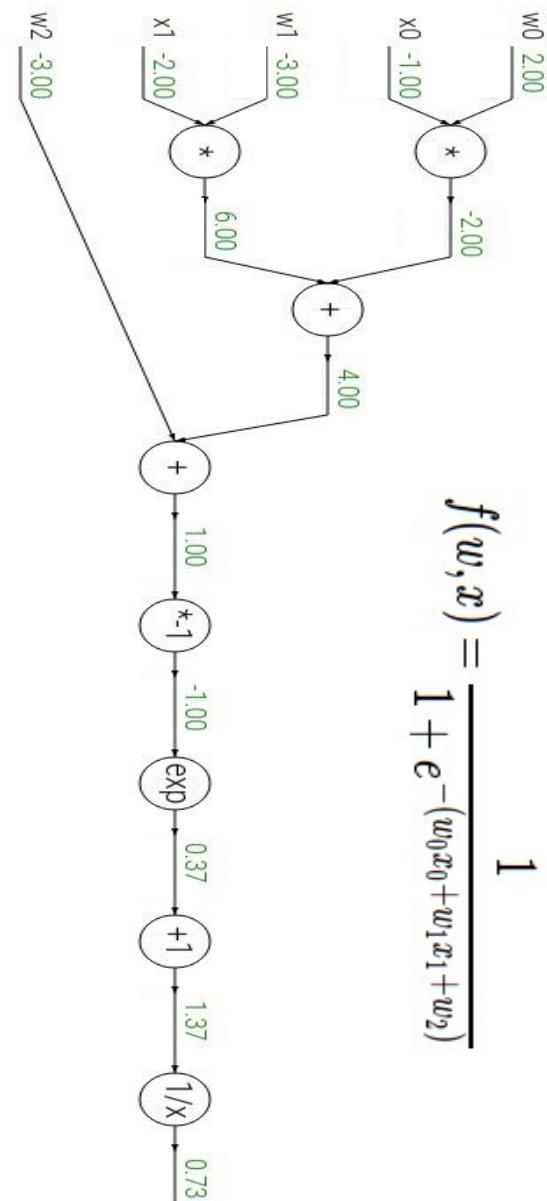
---

Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

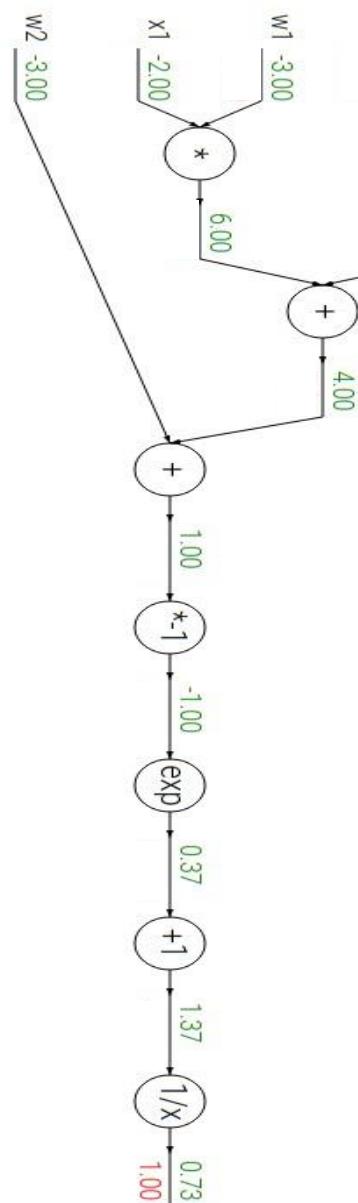
# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



# Outro Exemplo – Passo Retrógrado

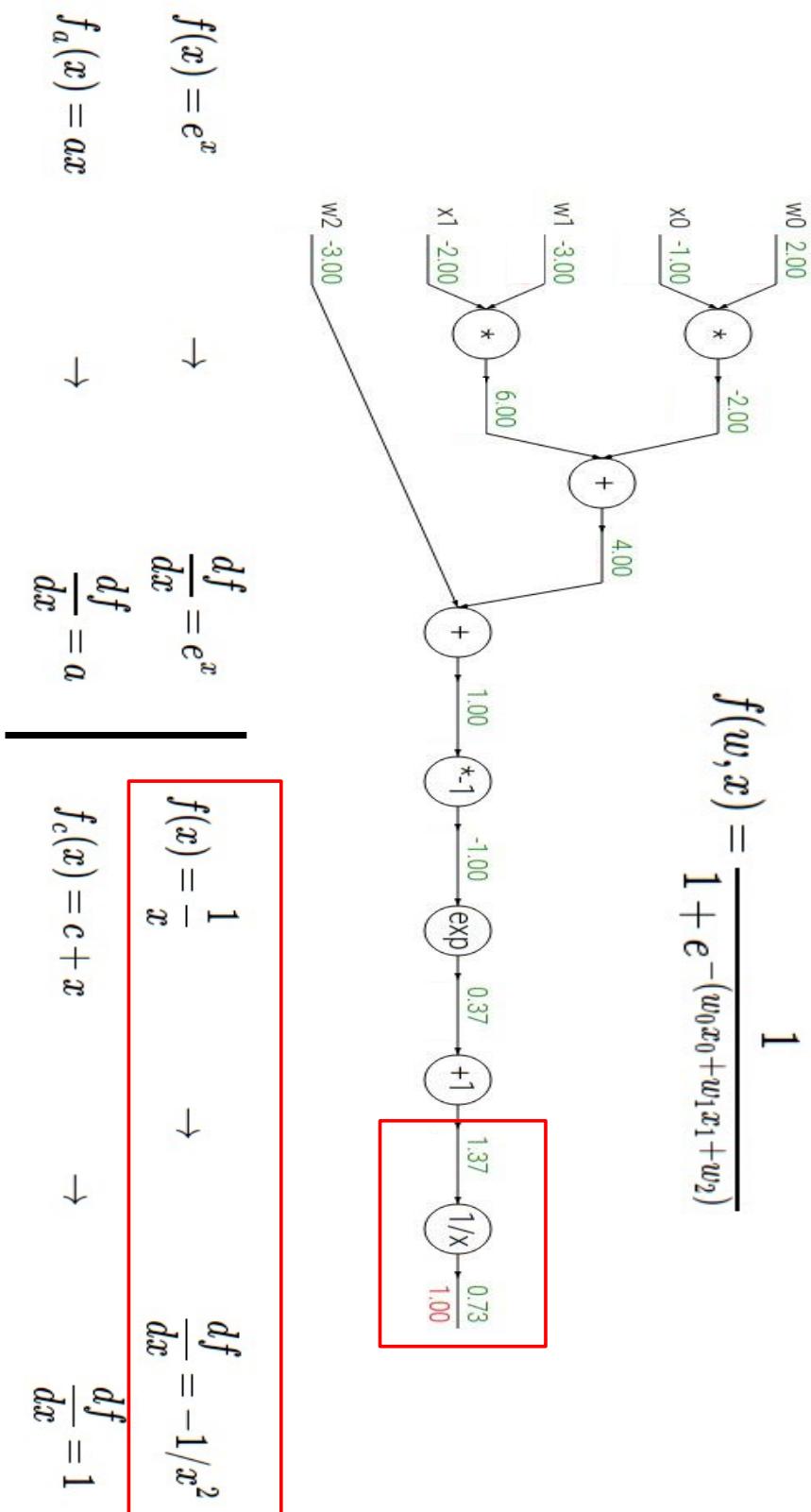
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



|               |               |                      |               |                          |
|---------------|---------------|----------------------|---------------|--------------------------|
| $f(x) = e^x$  | $\rightarrow$ | $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_a(x) = ax$ | $\rightarrow$ | $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

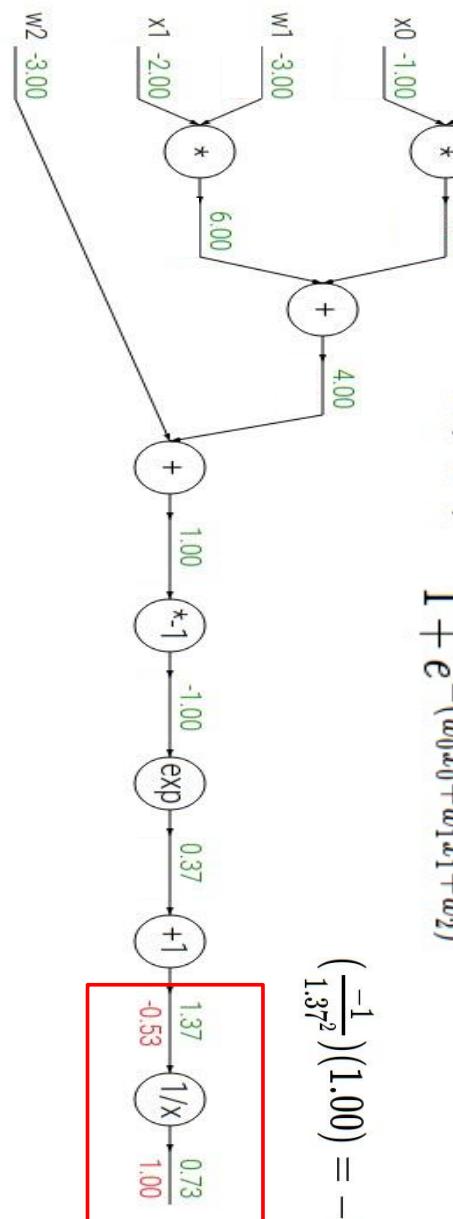


|                      |               |                          |
|----------------------|---------------|--------------------------|
| $f(x) = e^x$         | $\rightarrow$ | $\frac{df}{dx} = e^x$    |
| $f_a(x) = ax$        | $\rightarrow$ | $\frac{df}{dx} = a$      |
| $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

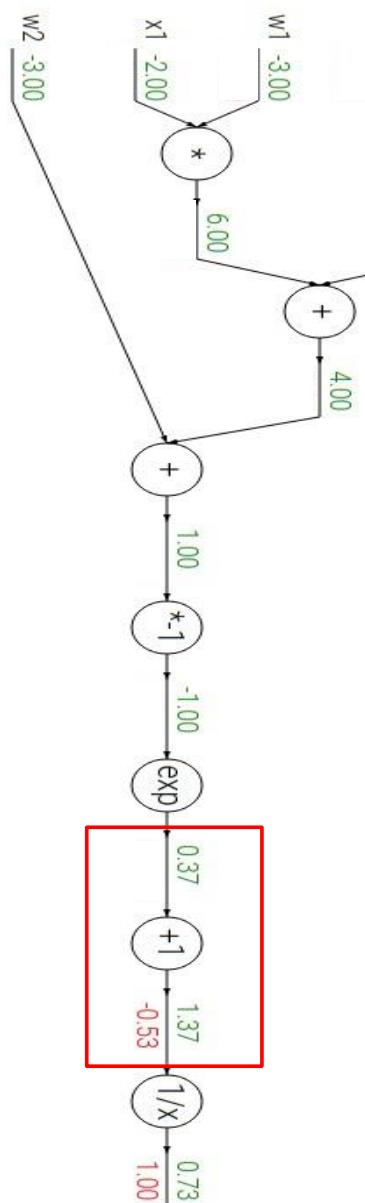
$$\left(\frac{-1}{1.37^2}\right)(1.00) = -0.53$$



|                      |               |                          |
|----------------------|---------------|--------------------------|
| $f(x) = e^x$         | $\rightarrow$ | $\frac{df}{dx} = e^x$    |
| $f_a(x) = ax$        | $\rightarrow$ | $\frac{df}{dx} = a$      |
| $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$

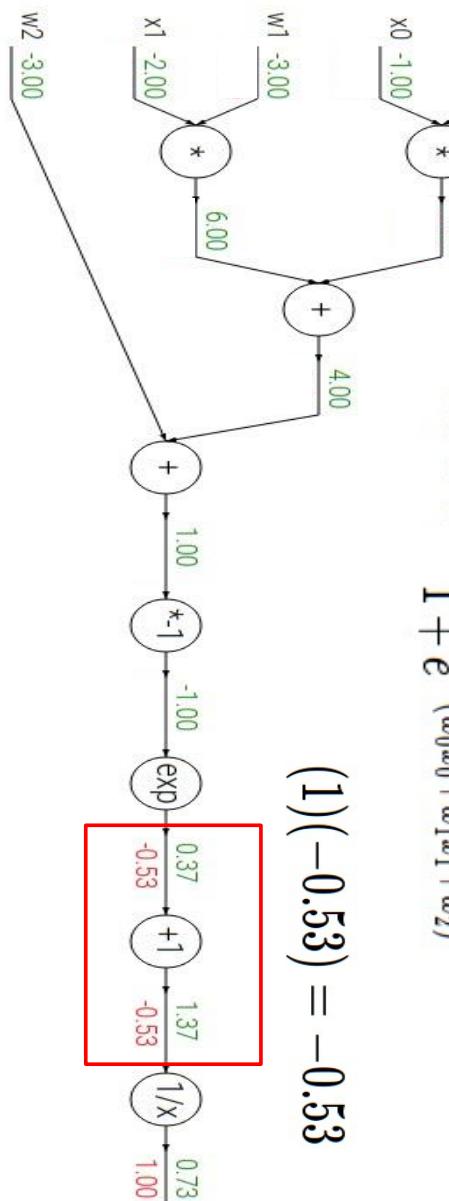


|                       |               |                          |               |                     |
|-----------------------|---------------|--------------------------|---------------|---------------------|
| $f(x) = e^x$          | $\rightarrow$ | $f(x) = \frac{1}{x}$     | $\rightarrow$ | $f_c(x) = c + x$    |
| $\frac{df}{dx} = e^x$ |               | $\frac{df}{dx} = -1/x^2$ |               | $\frac{df}{dx} = 1$ |
| $\frac{df}{dx} = a$   |               |                          |               |                     |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

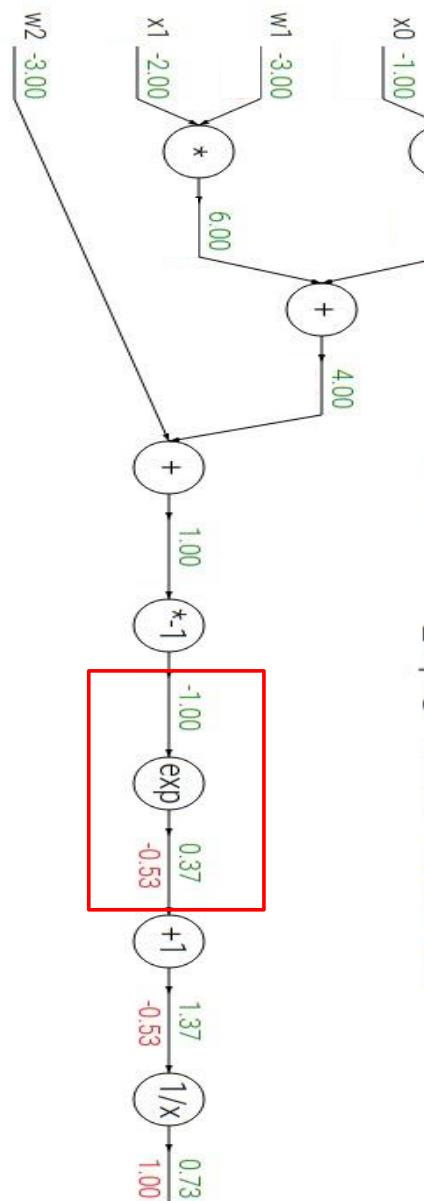
$$(1)(-0.53) = -0.53$$



|               |               |                      |               |                       |               |                     |               |                          |
|---------------|---------------|----------------------|---------------|-----------------------|---------------|---------------------|---------------|--------------------------|
| $f(x) = e^x$  | $\rightarrow$ | $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = e^x$ | $\rightarrow$ | $f_c(x) = c + x$    | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_a(x) = ax$ | $\rightarrow$ | $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = a$   | $\rightarrow$ | $\frac{df}{dx} = 1$ |               |                          |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



|               |               |                       |
|---------------|---------------|-----------------------|
| $f(x) = e^x$  | $\rightarrow$ | $\frac{df}{dx} = e^x$ |
| $f_a(x) = ax$ | $\rightarrow$ | $\frac{df}{dx} = a$   |

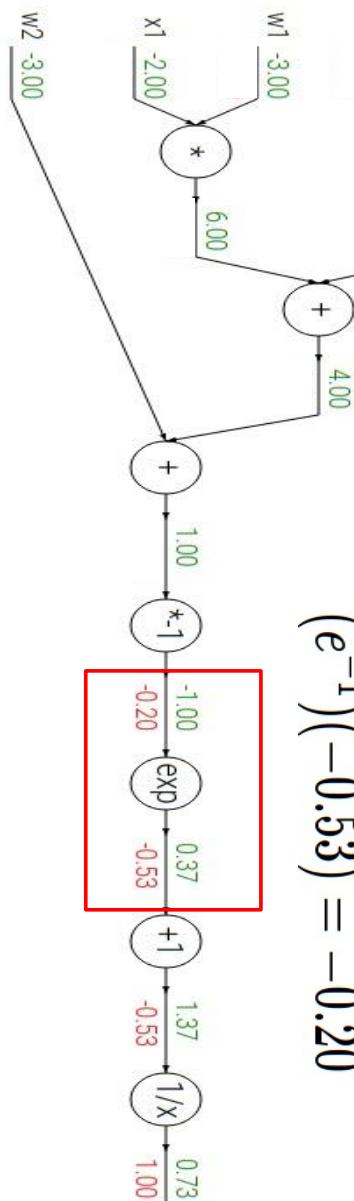
---

|                      |               |                          |
|----------------------|---------------|--------------------------|
| $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

$$(e^{-1})(-0.53) = -0.20$$



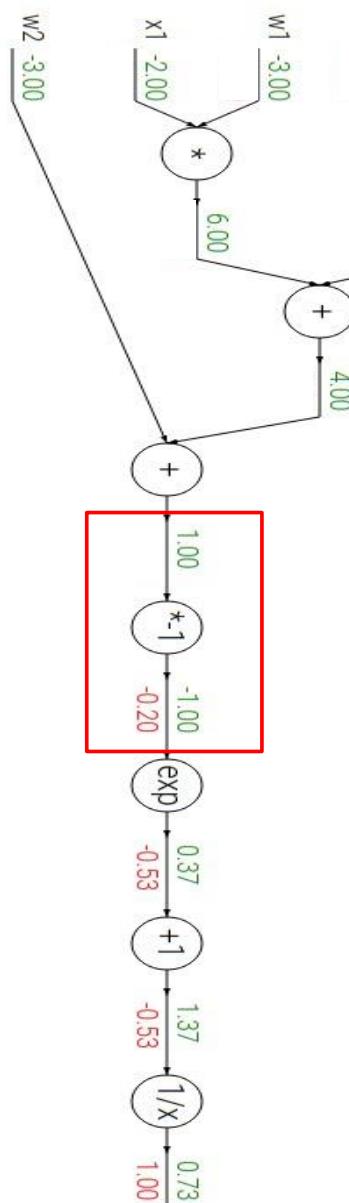
|               |               |                       |
|---------------|---------------|-----------------------|
| $f(x) = e^x$  | $\rightarrow$ | $\frac{df}{dx} = e^x$ |
| $f_a(x) = ax$ | $\rightarrow$ | $\frac{df}{dx} = a$   |

|                      |               |                          |
|----------------------|---------------|--------------------------|
| $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



|               |               |                       |
|---------------|---------------|-----------------------|
| $f(x) = e^x$  | $\rightarrow$ | $\frac{df}{dx} = e^x$ |
| $f_a(x) = ax$ | $\rightarrow$ | $\frac{df}{dx} = a$   |

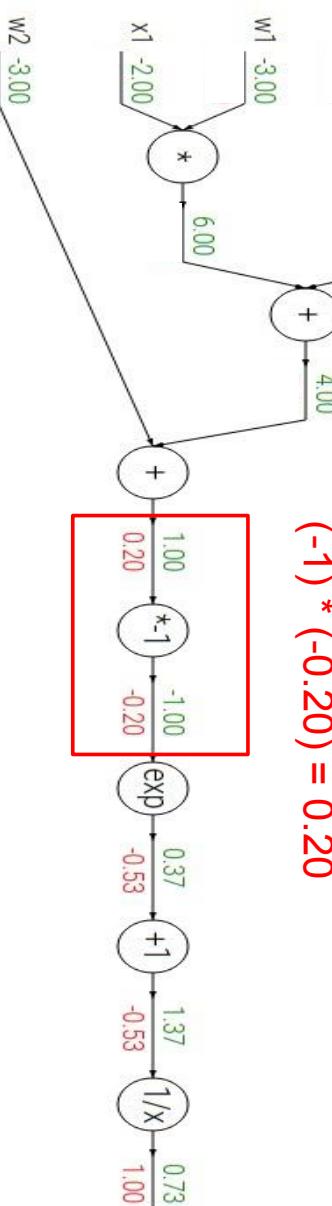
---

|                      |               |                          |
|----------------------|---------------|--------------------------|
| $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

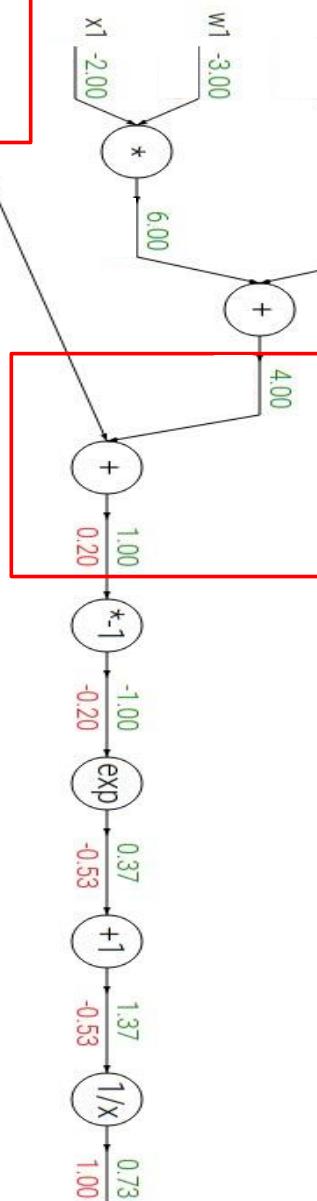
$$(-1) * (-0.20) = 0.20$$



|               |               |                       |                      |               |                          |
|---------------|---------------|-----------------------|----------------------|---------------|--------------------------|
| $f(x) = e^x$  | $\rightarrow$ | $\frac{df}{dx} = e^x$ | $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_a(x) = ax$ | $\rightarrow$ | $\frac{df}{dx} = a$   | $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



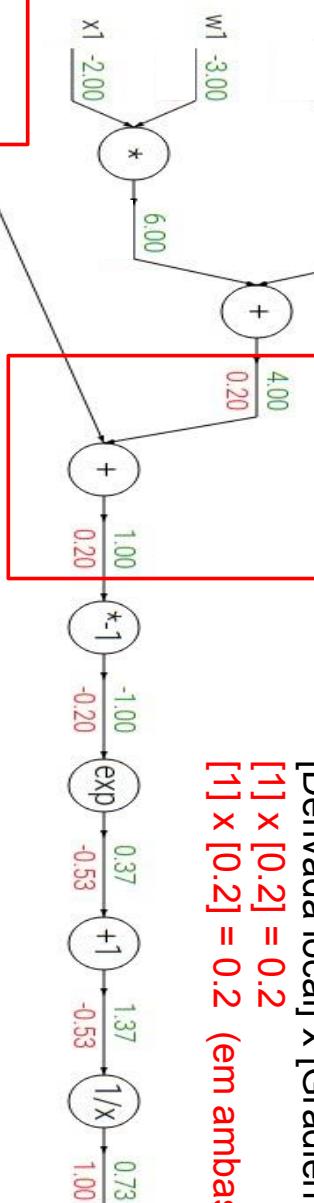
|               |               |                      |               |                          |
|---------------|---------------|----------------------|---------------|--------------------------|
| $f(x) = e^x$  | $\rightarrow$ | $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_a(x) = ax$ | $\rightarrow$ | $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

[Derivada local]  $\times$  [Gradiente]

$$\begin{aligned} [1] \times [0.2] &= 0.2 \\ [1] \times [0.2] &= 0.2 \text{ (em ambas entradas!)} \end{aligned}$$

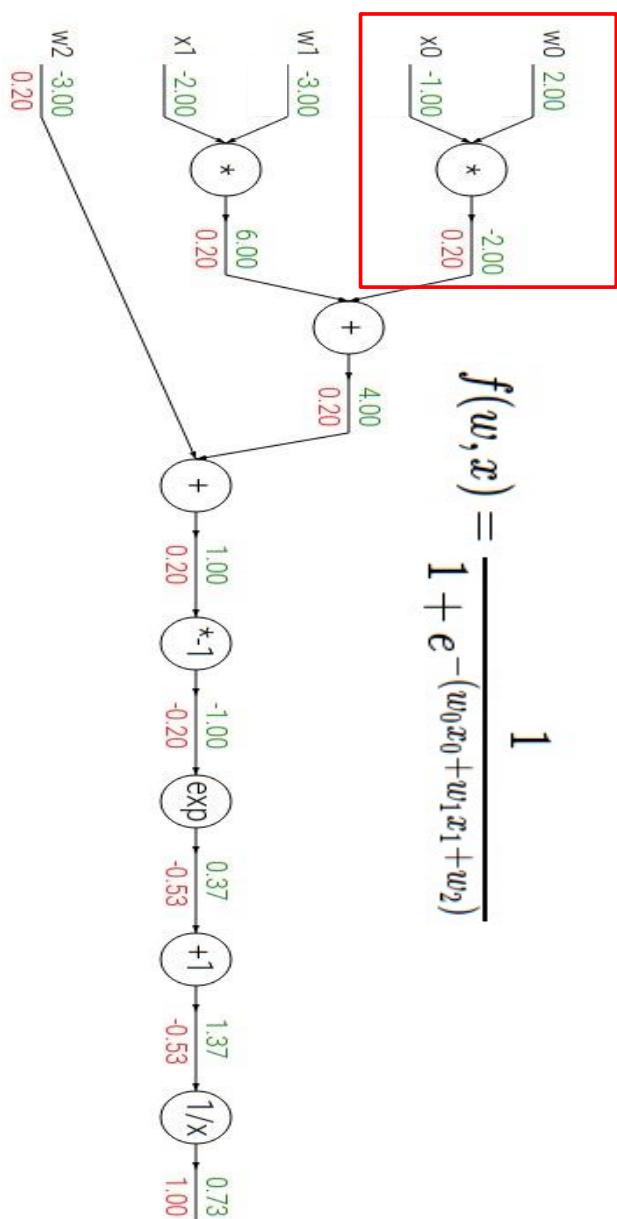


$$\boxed{\begin{array}{c} w_2 \quad -3.00 \\ \hline 0.20 \end{array}}$$

$$\begin{array}{lcl} f(x) = e^x & \rightarrow & f(x) = \frac{1}{x} \\ \frac{df}{dx} = e^x & & \rightarrow & \frac{df}{dx} = -1/x^2 \\ \frac{df}{dx} = a & \rightarrow & f_c(x) = c + x & \rightarrow & \frac{df}{dx} = 1 \end{array}$$

# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



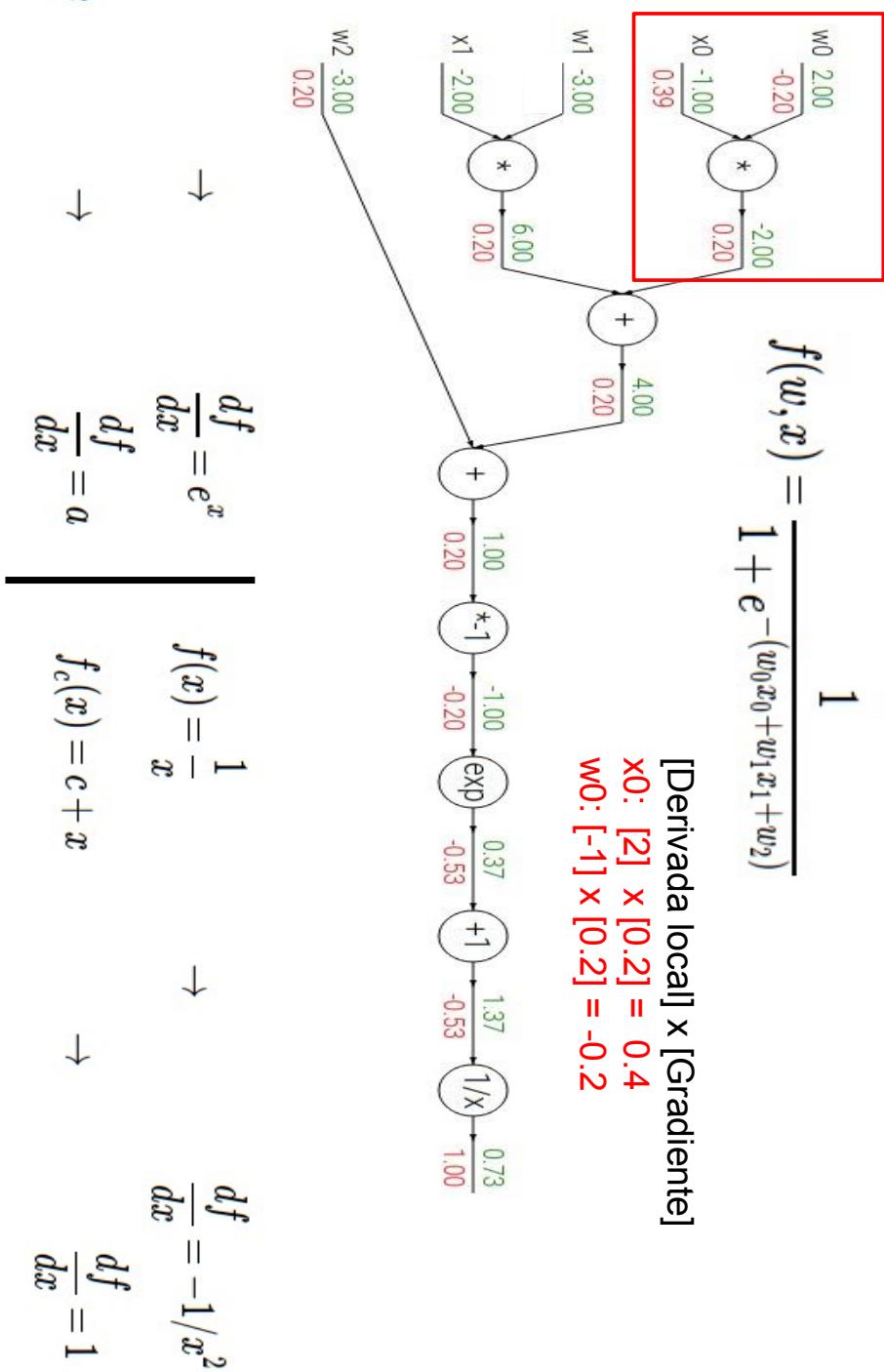
|               |               |                       |                      |               |                          |
|---------------|---------------|-----------------------|----------------------|---------------|--------------------------|
| $f(x) = e^x$  | $\rightarrow$ | $\frac{df}{dx} = e^x$ | $f(x) = \frac{1}{x}$ | $\rightarrow$ | $\frac{df}{dx} = -1/x^2$ |
| $f_a(x) = ax$ | $\rightarrow$ | $\frac{df}{dx} = a$   | $f_c(x) = c + x$     | $\rightarrow$ | $\frac{df}{dx} = 1$      |

## Outro Exemplo – Passo Retrógrafo

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

[Derivada local] x [Gradiente]

$$\begin{array}{l} x_0: [2] \times [0.2] = 0.4 \\ w_0: [-1] \times [0.2] = -0.2 \end{array}$$



# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

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

Função sigmoide

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

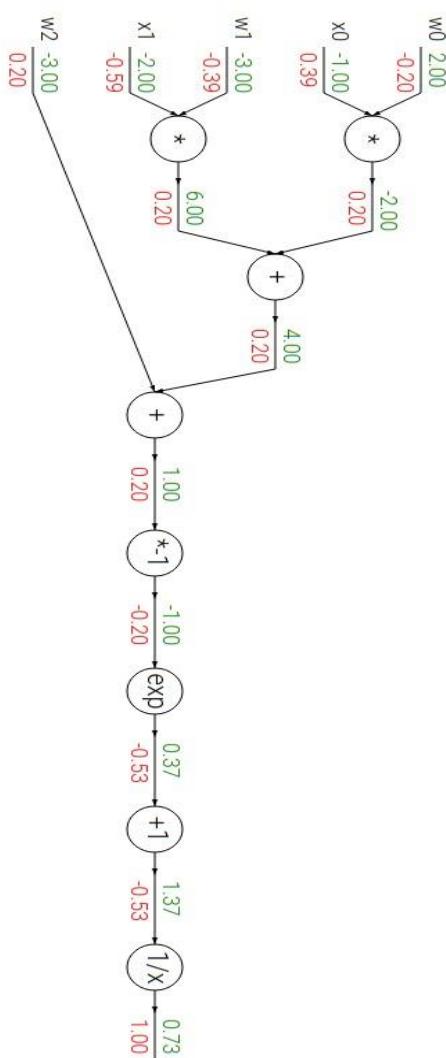
# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

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

Função sigmoide

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$



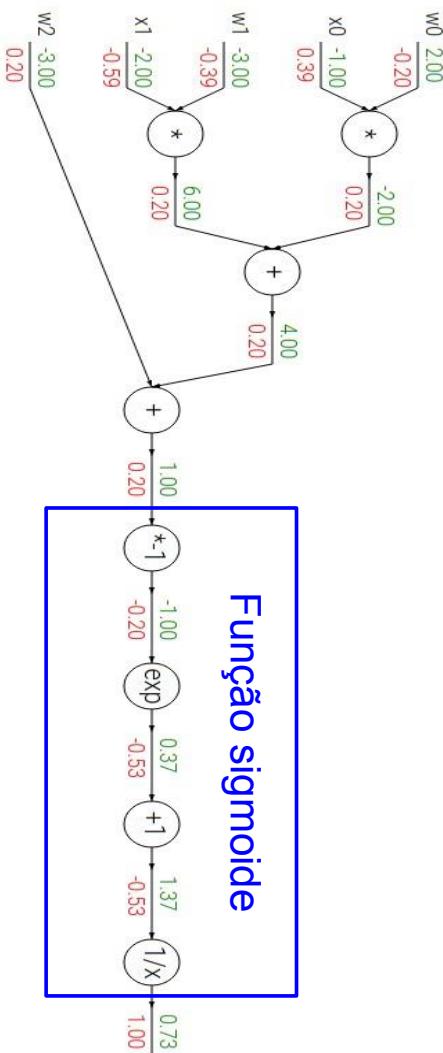
# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

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

Função sigmoide

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$



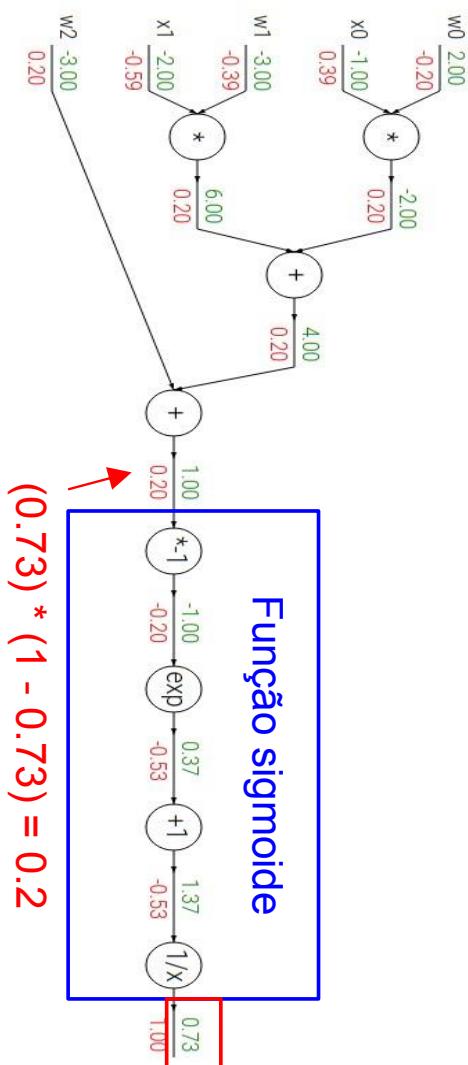
# Outro Exemplo – Passo Retrógrado

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

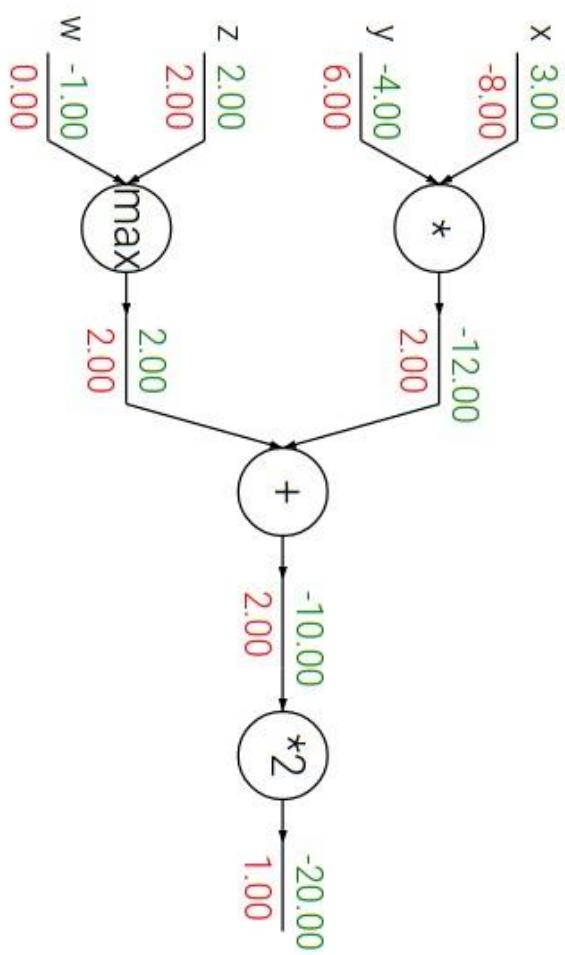
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Função sigmoide

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

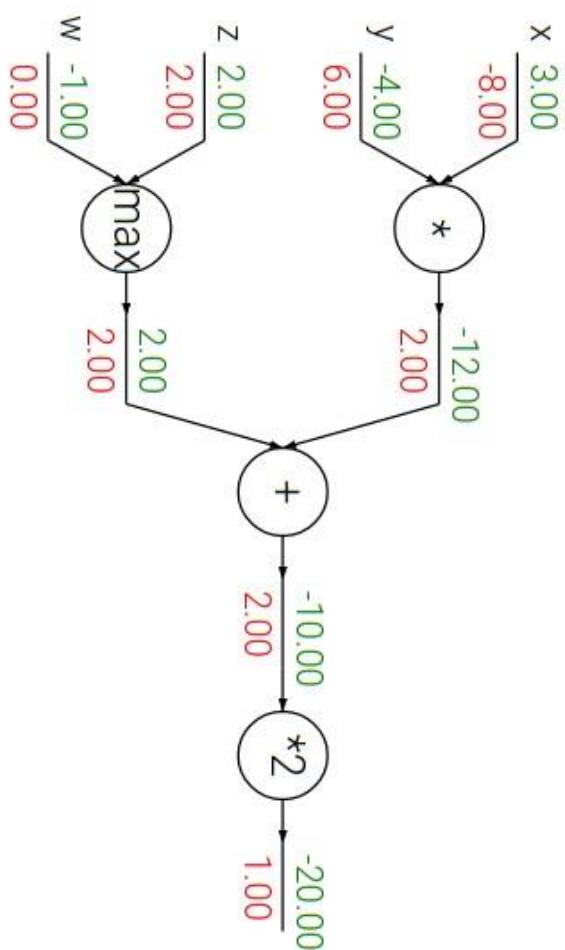


# Padrões no Fluxo Reverso



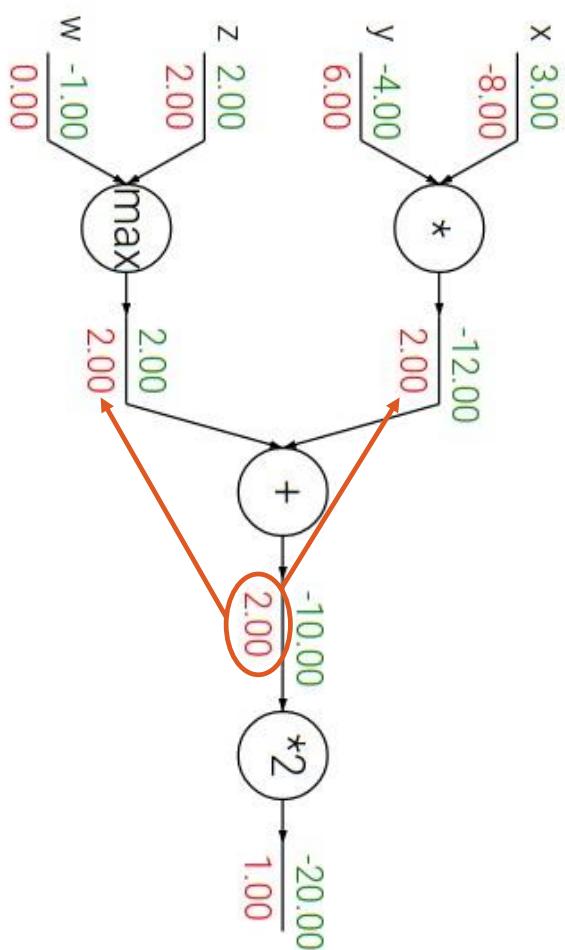
# Padrões no Fluxo Reverso

**Adição** : distribuidor de gradiente



# Padrões no Fluxo Reverso

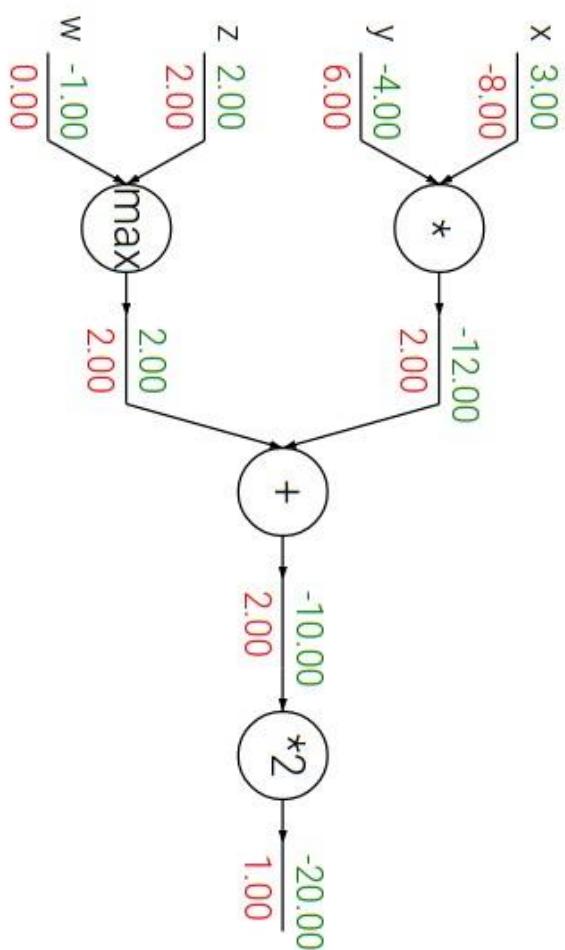
**Adição** : distribuidor de gradiente



# Padrões no Fluxo Reverso

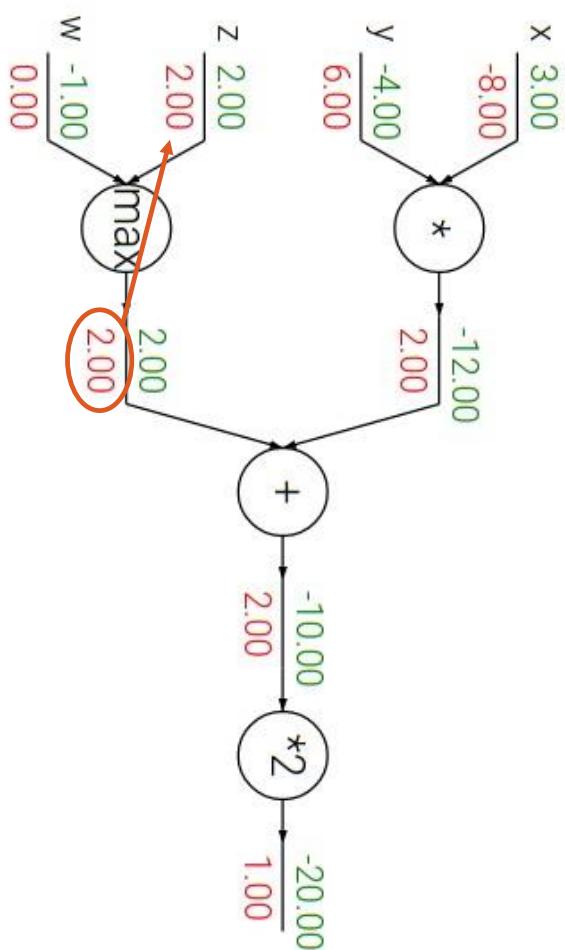
**Adição** : distribuidor de gradiente

**Max** : direcionador de gradiente



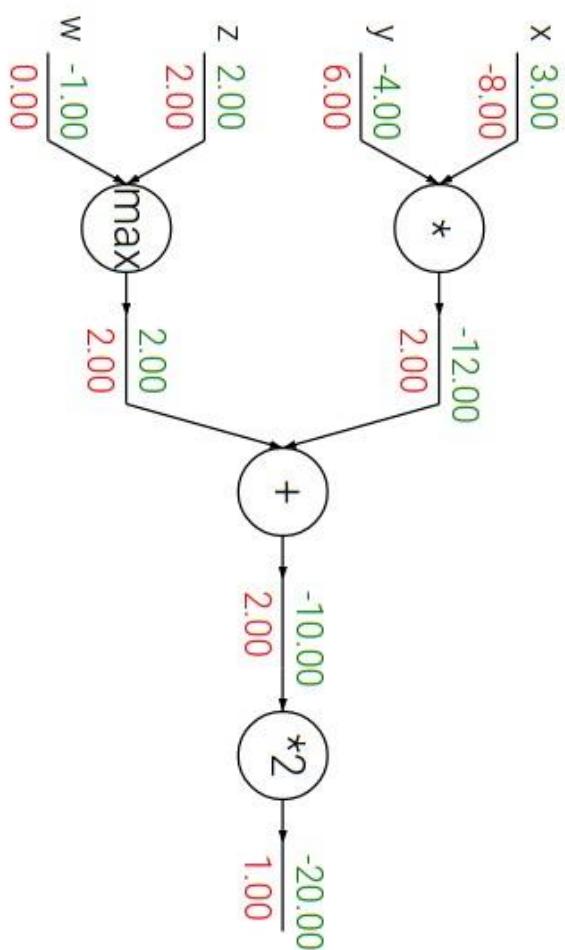
# Padrões no Fluxo Reverso

**Adição** : distribuidor de gradiente  
**Max** : direcionador de gradiente



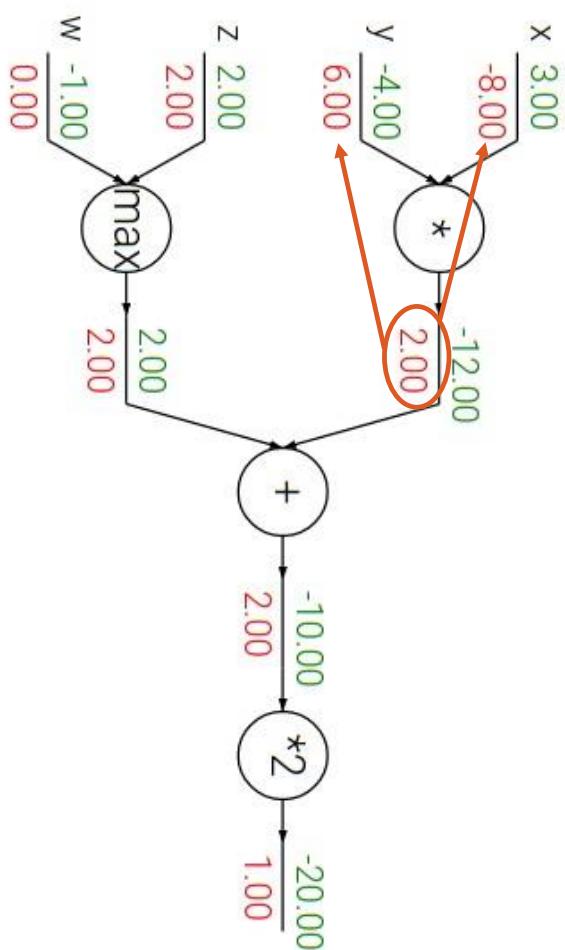
# Padrões no Fluxo Reverso

**Adição** : distribuidor de gradiente  
**Max** : direcionador de gradiente  
**Produto**: “comutador” de gradiente



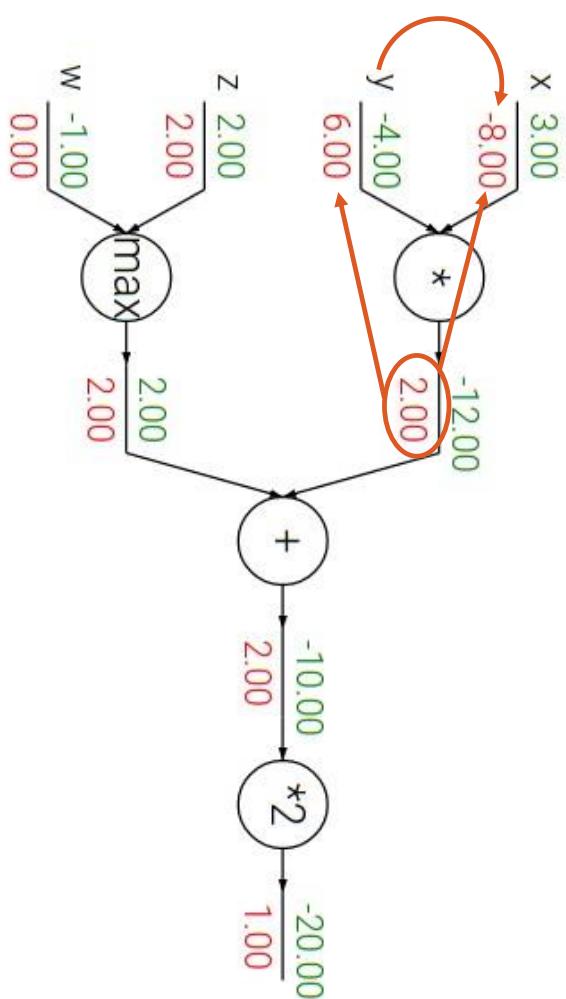
# Padrões no Fluxo Reverso

**Adição** : distribuidor de gradiente  
**Max** : direcionador de gradiente  
**Produto**: “comutador” de gradiente



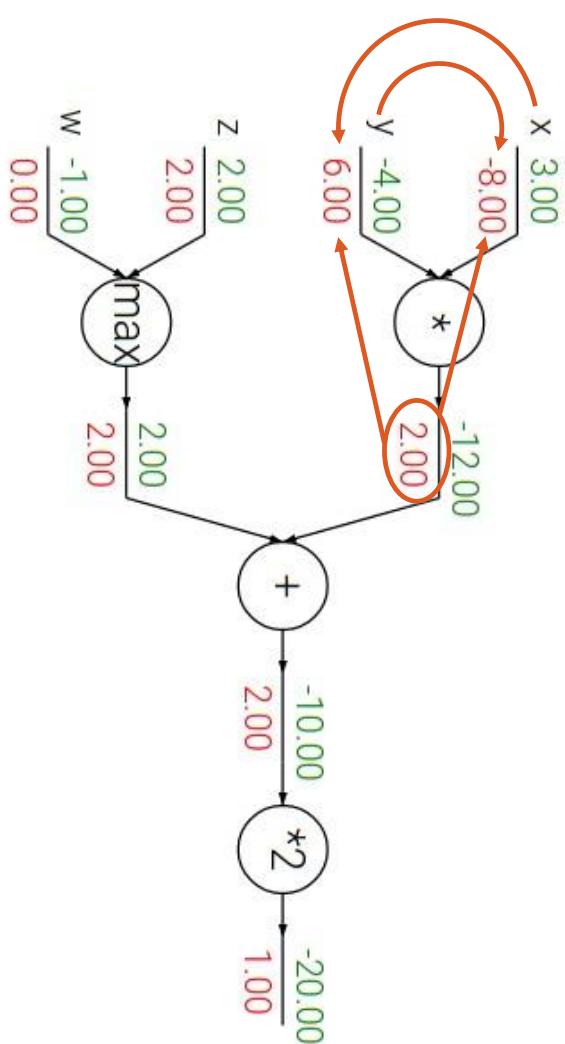
# Padrões no Fluxo Reverso

**Adição** : distribuidor de gradiente  
**Max** : direcionador de gradiente  
**Produto**: “comutador” de gradiente

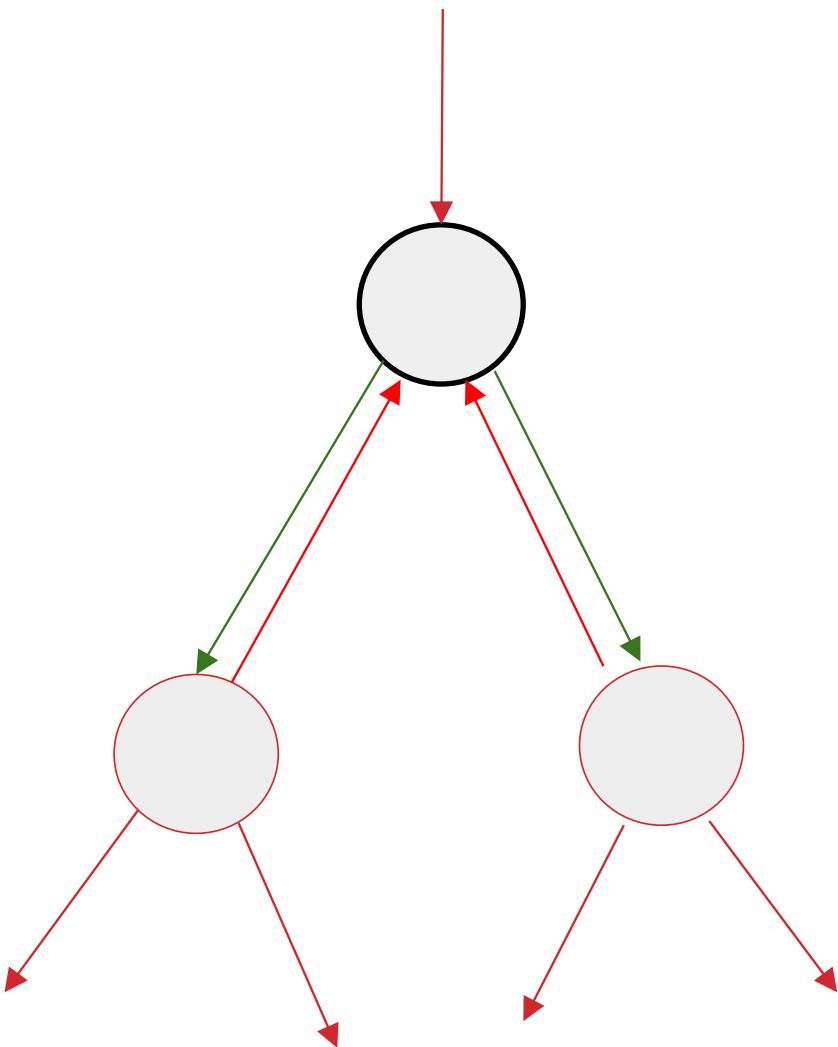


# Padrões no Fluxo Reverso

**Adição** : distribuidor de gradiente  
**Max** : direcionador de gradiente  
**Produto**: “comutador” de gradiente

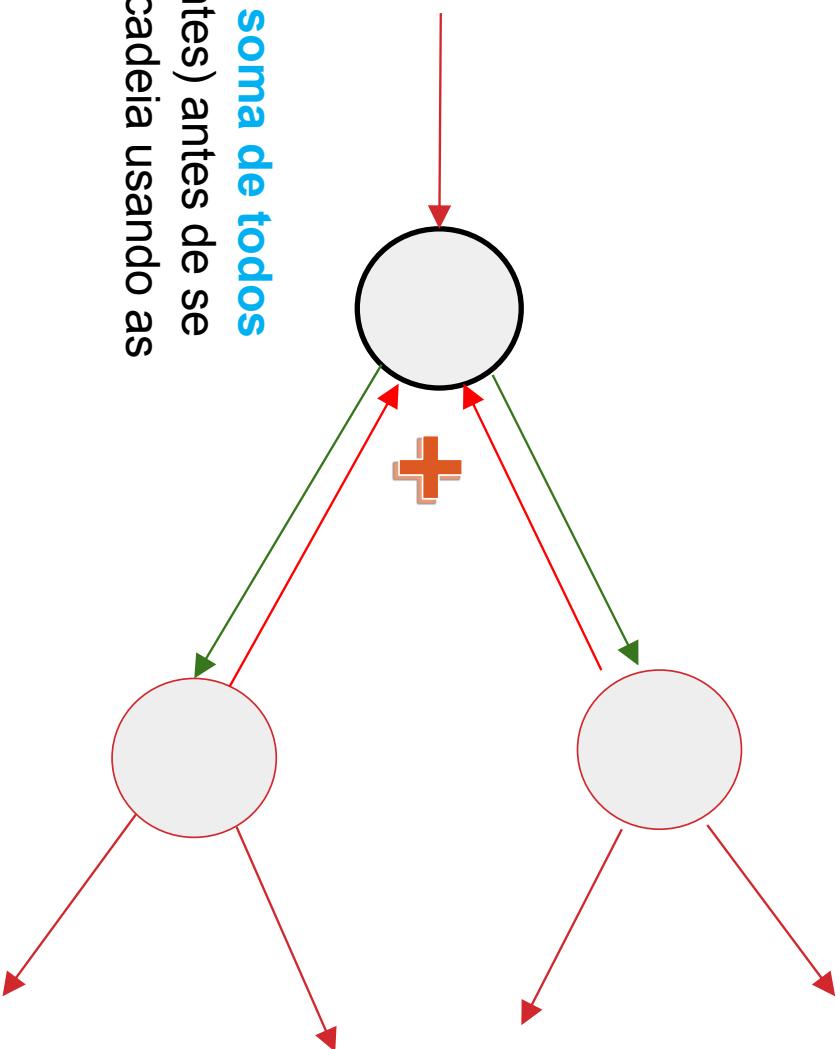


# Gradientes nas Ramificações



# Gradientes nas Ramificações

Deve-se realizar a **soma de todos os fluxos** (gradientes) antes de se aplicar a regra da cadeia usando as derivadas locais



# Redes Neurais e Aprendizagem Profunda

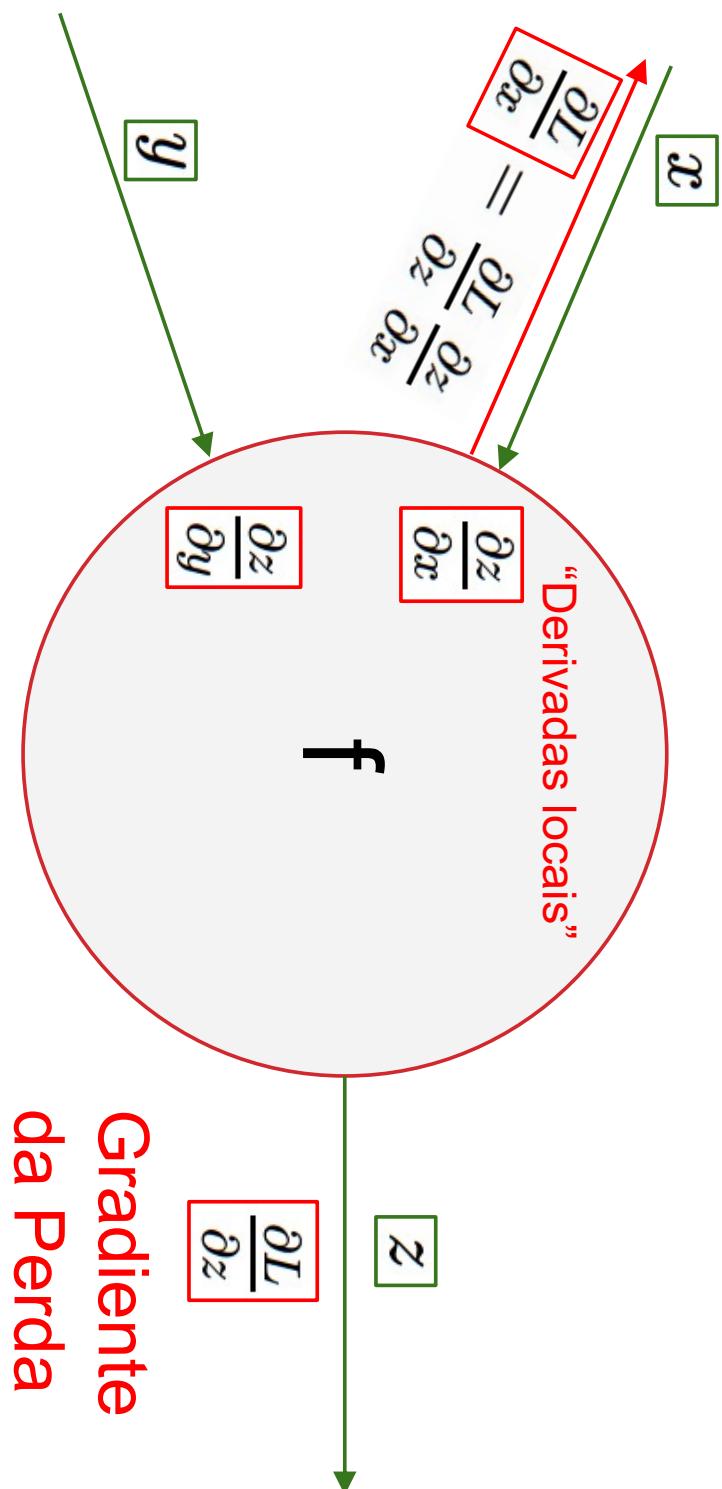
## REDES NEURAIS ARTIFICIAIS PROPAGAÇÃO RETRÓGRADA (III)

---

Zenilton K. G. Patrocínio Jr

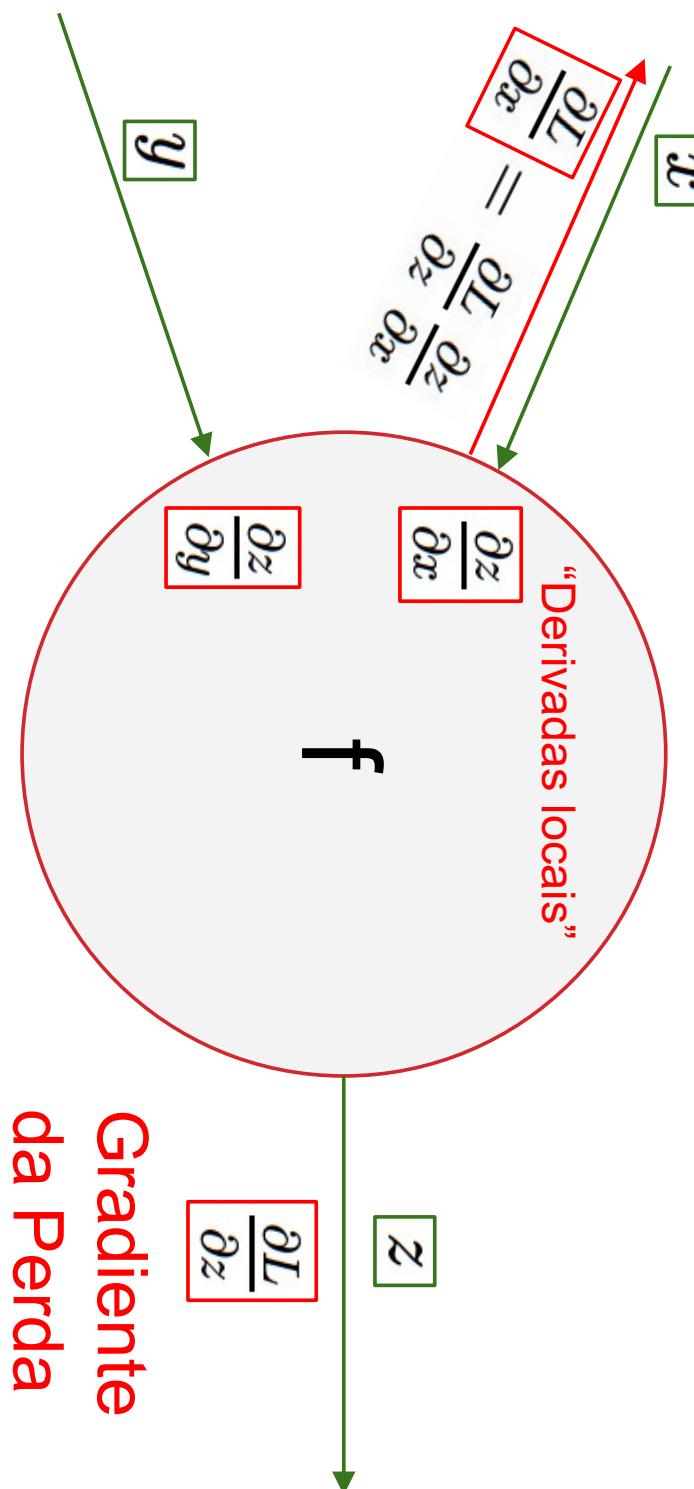
[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

# Gradientes para Dados Multidimensionais

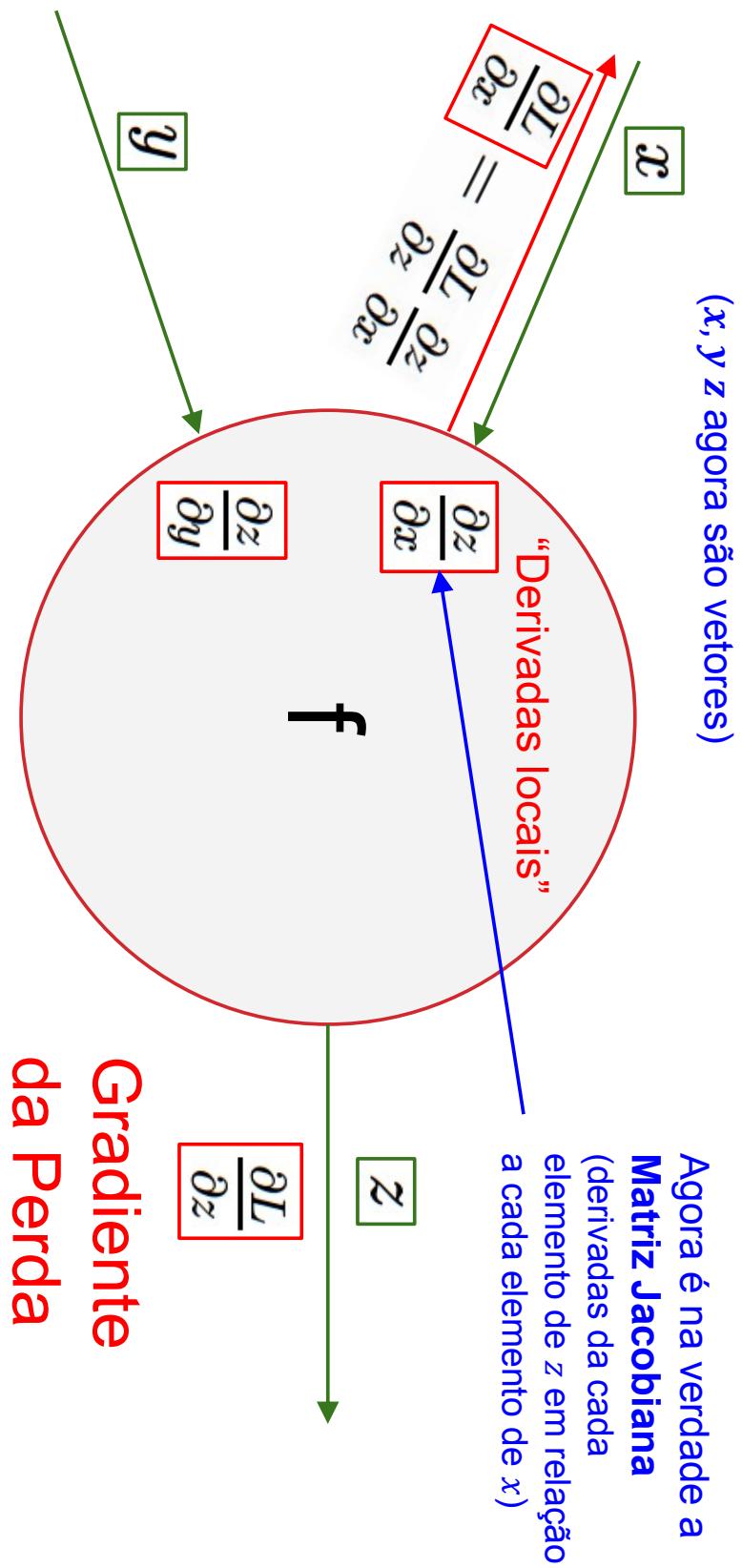


# Gradientes para Dados Multidimensionais

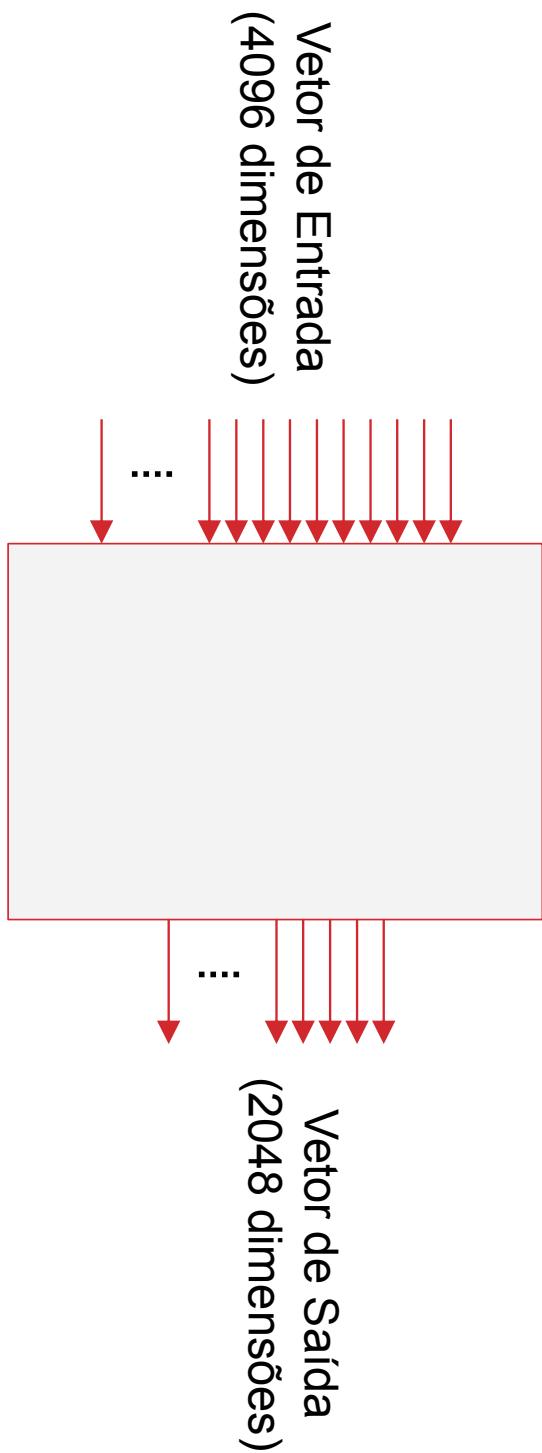
( $x, y, z$  agora são vetores)



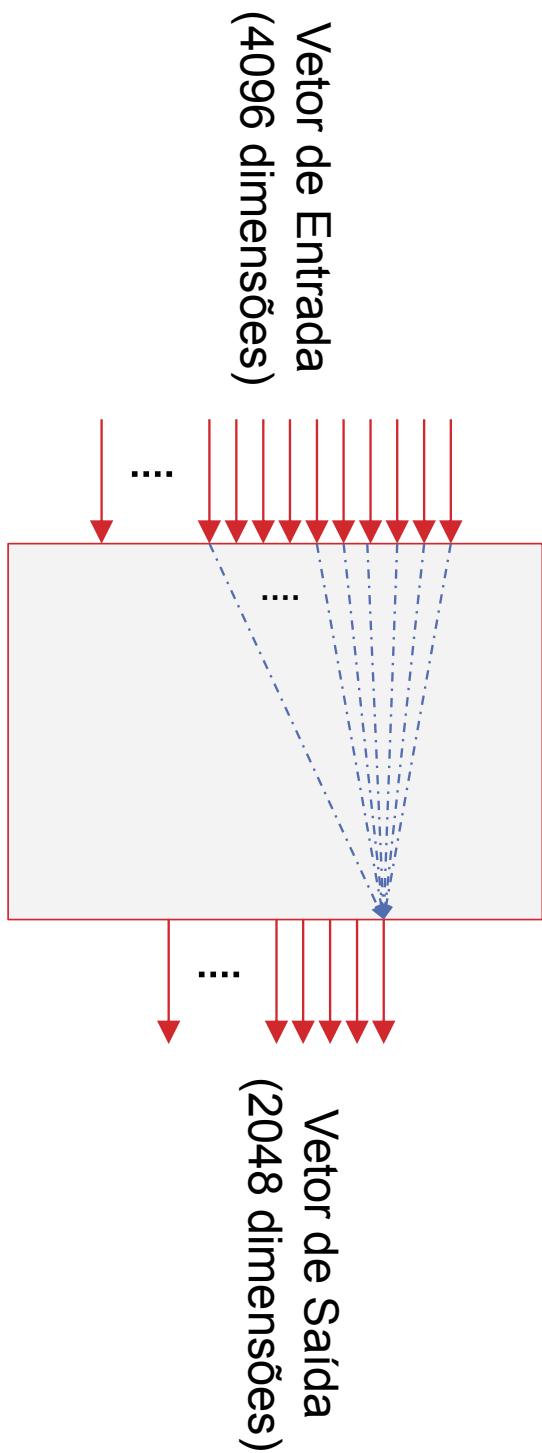
# Gradientes para Dados Multidimensionais



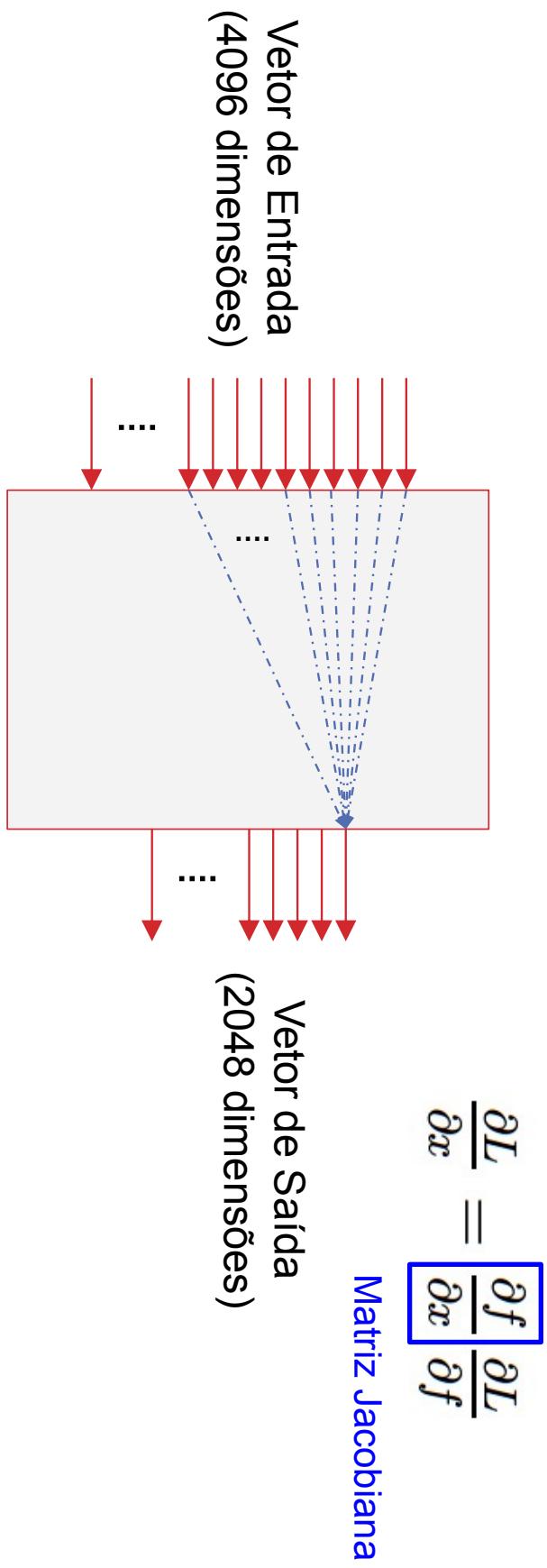
# Gradientes para Dados Multidimensionais



# Gradientes para Dados Multidimensionais



# Gradientes para Dados Multidimensionais



# Gradientes para Dados Multidimensionais

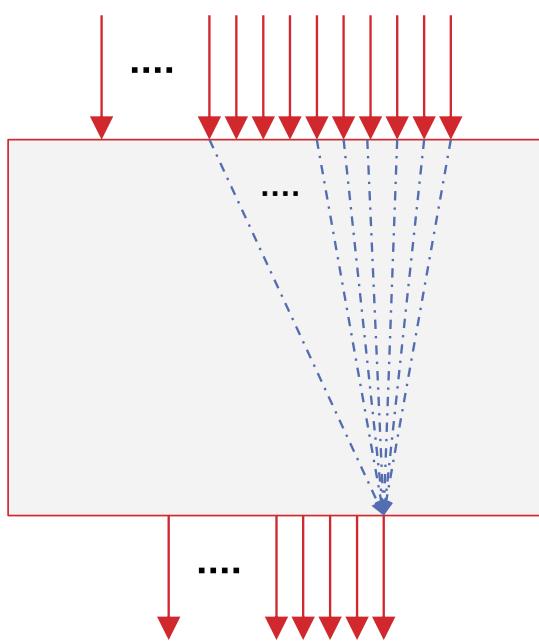
$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Matriz Jacobiana

Vetor de Entrada  
(4096 dimensões)

Vetor de Saída  
(2048 dimensões)

P1: qual o tamanho  
da matriz Jacobiana?



# Gradientes para Dados Multidimensionais

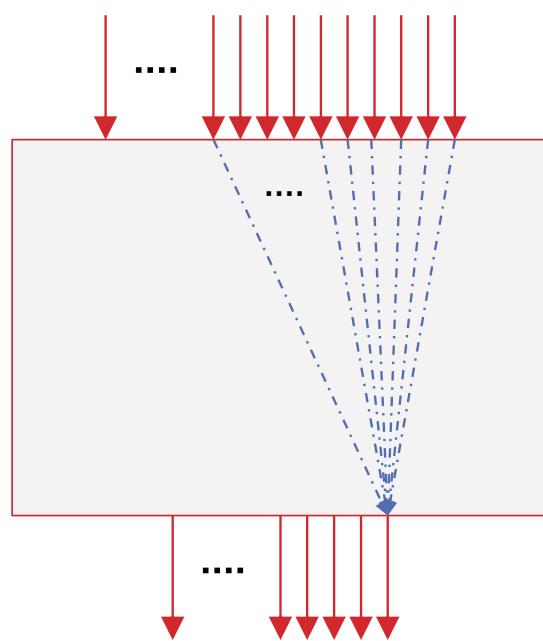
$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Matriz Jacobiana

Vetor de Entrada  
(4096 dimensões)

Vetor de Saída  
(2048 dimensões)

P1: qual o tamanho  
da matriz Jacobiana?  
 $\Rightarrow 2048 \times 4096 !$



# Gradientes para Dados Multidimensionais

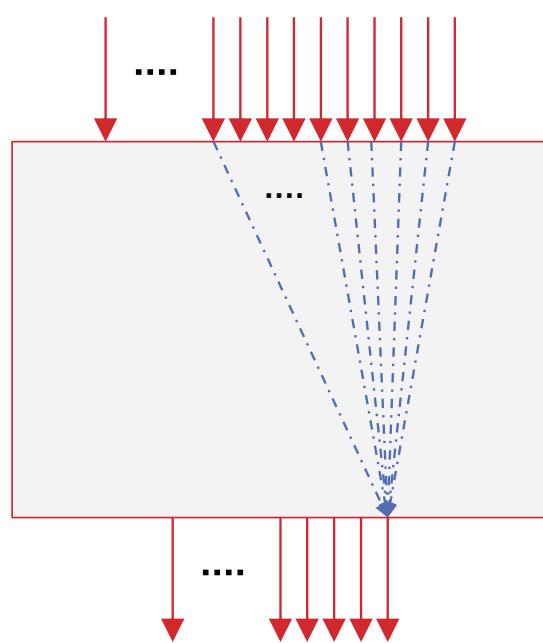
$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Matriz Jacobiana

Vetor de Entrada  
(4096 dimensões)

Vetor de Saída  
(2048 dimensões)

P1: qual o tamanho  
da matriz Jacobiana?  
 $\Rightarrow 2048 \times 4096 !$



P2: qual a aparência  
dessa matriz?  
 $\Rightarrow 2048 \times 4096 !$

# Gradientes para Dados Multidimensionais

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Matriz Jacobiana

Vetor de Entrada  
(4096 dimensões)

Vetor de Saída  
(2048 dimensões)

P1: qual o tamanho  
da matriz Jacobiana?  
 $\Rightarrow 2048 \times 4096 !$

P2: qual a aparência  
dessa matriz?

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial x_1} & \frac{\partial f_k}{\partial x_2} & \dots & \frac{\partial f_k}{\partial x_m} \end{bmatrix}$$

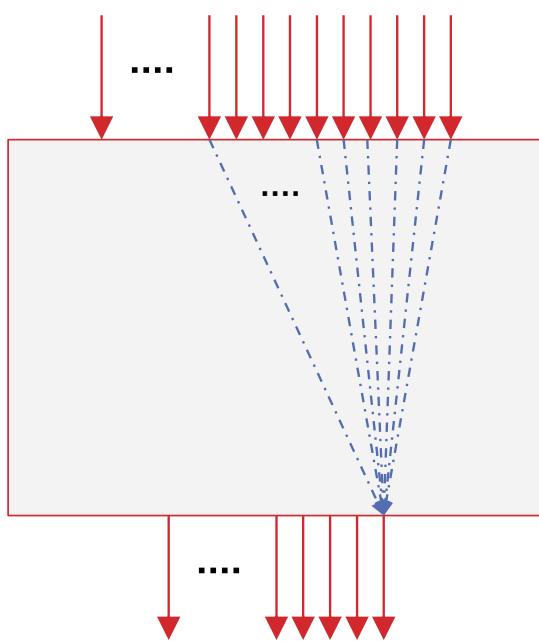
# Gradientes para Dados Multidimensionais

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Matriz Jacobiana

Vetor de Entrada  
(4096 dimensões)

Vetor de Saída  
(2048 dimensões)



Na prática, processa-se todo um lote ou “*minibatch*” (p.ex. 100 amostras) de uma só vez

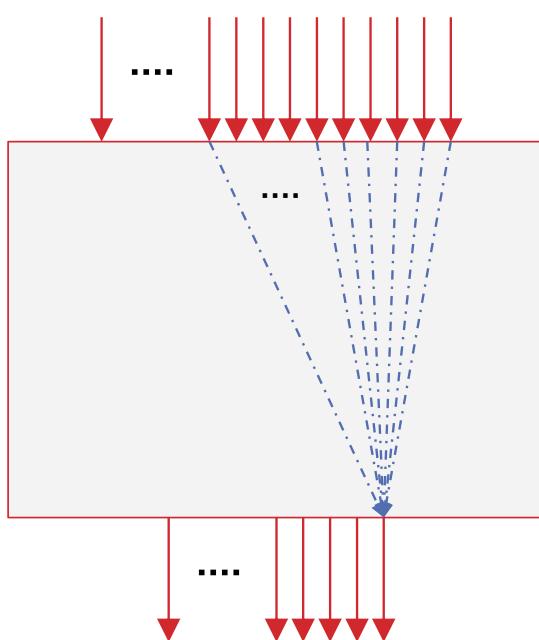
# Gradientes para Dados Multidimensionais

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Matriz Jacobiana

Vetor de Entrada  
(4096 dimensões)

Vetor de Saída  
(2048 dimensões)



Na prática, processa-se todo um lote ou “*minibatch*” (p.ex. 100 amostras) de uma só vez

Assim, as dimensões da matriz Jacobiana desse exemplo seriam

**204.800 × 409.600**

**≈ 83,9 bilhões de pesos :**

# Implementação – Forward / Backward API

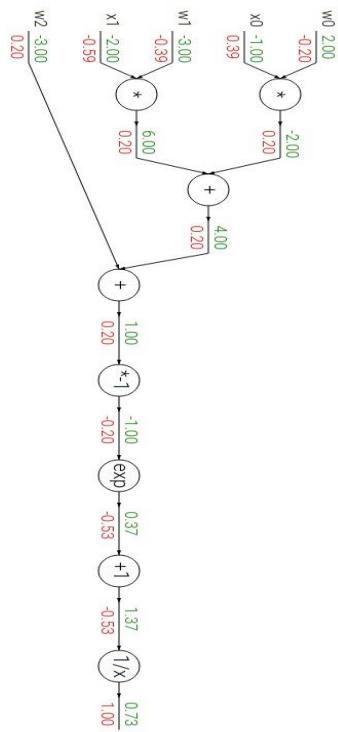
Para um grafo ou rede (*pseudocódigo*)

```
class ComputationalGraph(object):
    ...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()

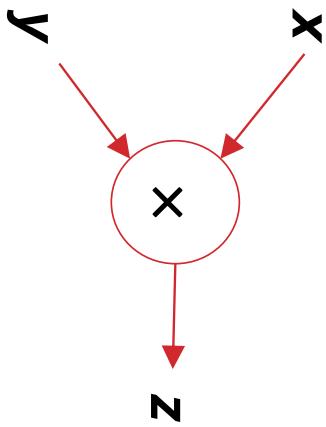
        return loss # the final gate in the graph outputs the loss

    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)

        return inputs_gradients
```



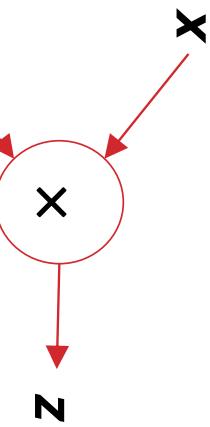
# Implementação – Forward / Backward API



```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        return z  
  
    def backward(dz):  
        # dx = ... #todo  
        # dy = ... #todo  
        return [dx, dy]
```

( $x, y, z$  escalares)

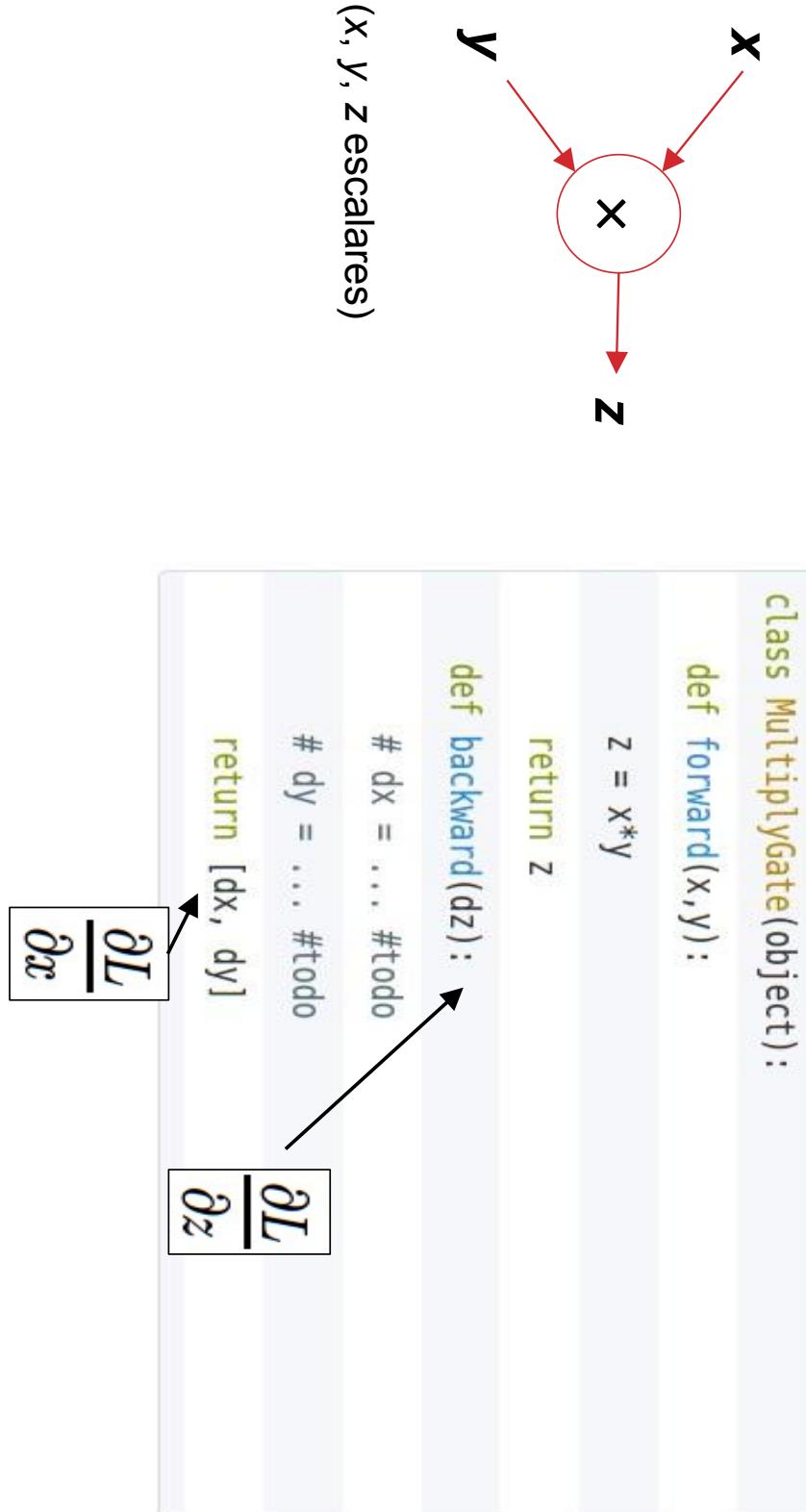
# Implementação – Forward / Backward API



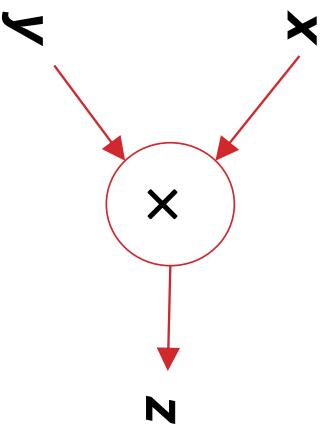
```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        return z  
  
    def backward(dz):  
        # dx = ... #todo  
        # dy = ... #todo  
        return [dx, dy]
```

( $x, y, z$  escalares)

# Implementação – Forward / Backward API



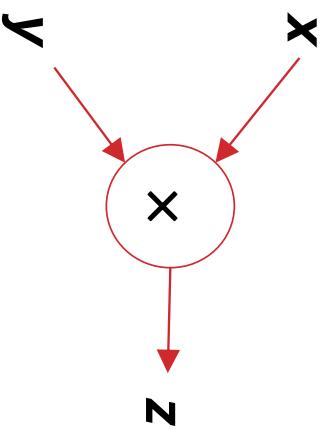
# Implementação – Forward / Backward API



```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        self.x = x # must keep these around!  
        self.y = y  
        return z  
  
    def backward(dz):  
        dx = self.y * dz # [dz/dx * dL/dz]  
        dy = self.x * dz # [dz/dy * dL/dz]  
        return [dx, dy]
```



# Implementação – Forward / Backward API

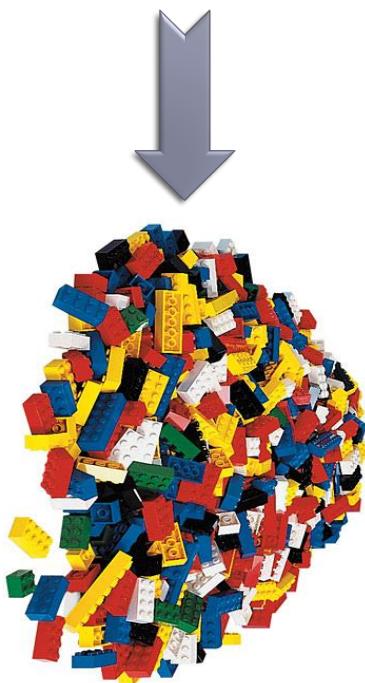


```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        self.x = x # must keep these around!  
        self.y = y  
        return z  
  
    def backward(dz):  
        dx = self.y * dz # [dz/dx * dL/dz]  
        dy = self.x * dz # [dz/dy * dL/dz]  
        return [dx, dy]
```



Biblioteca de Componentes / Camadas

Biblioteca de Componentes / Camadas



Biblioteca de Componentes / Camadas



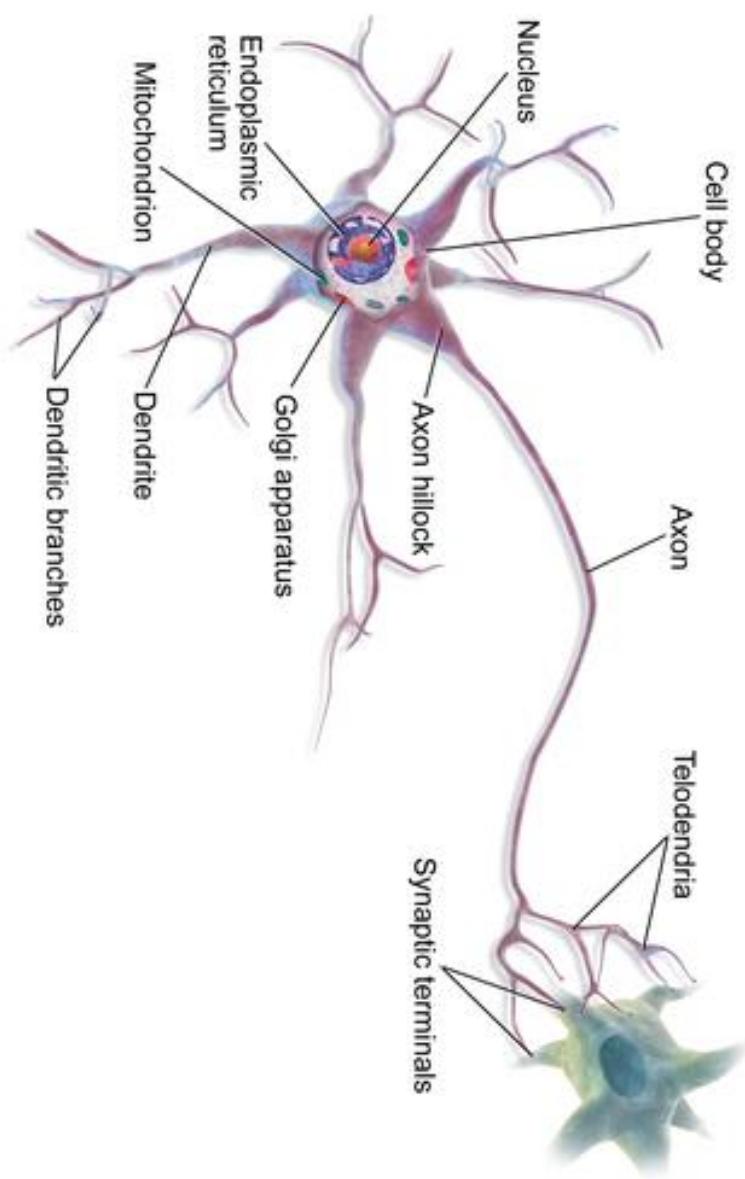
# Redes Neurais e Aprendizagem Profunda

## REDES NEURAIS ARTIFICIAIS FUNÇÃO DE ATIVAÇÃO

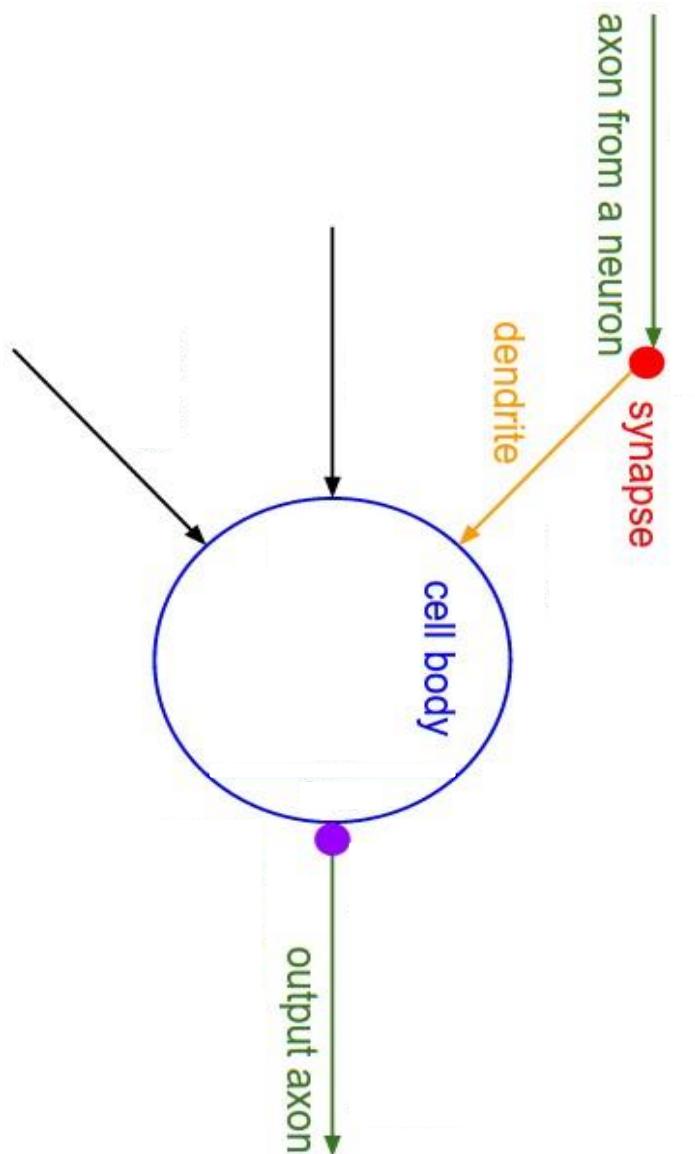
Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

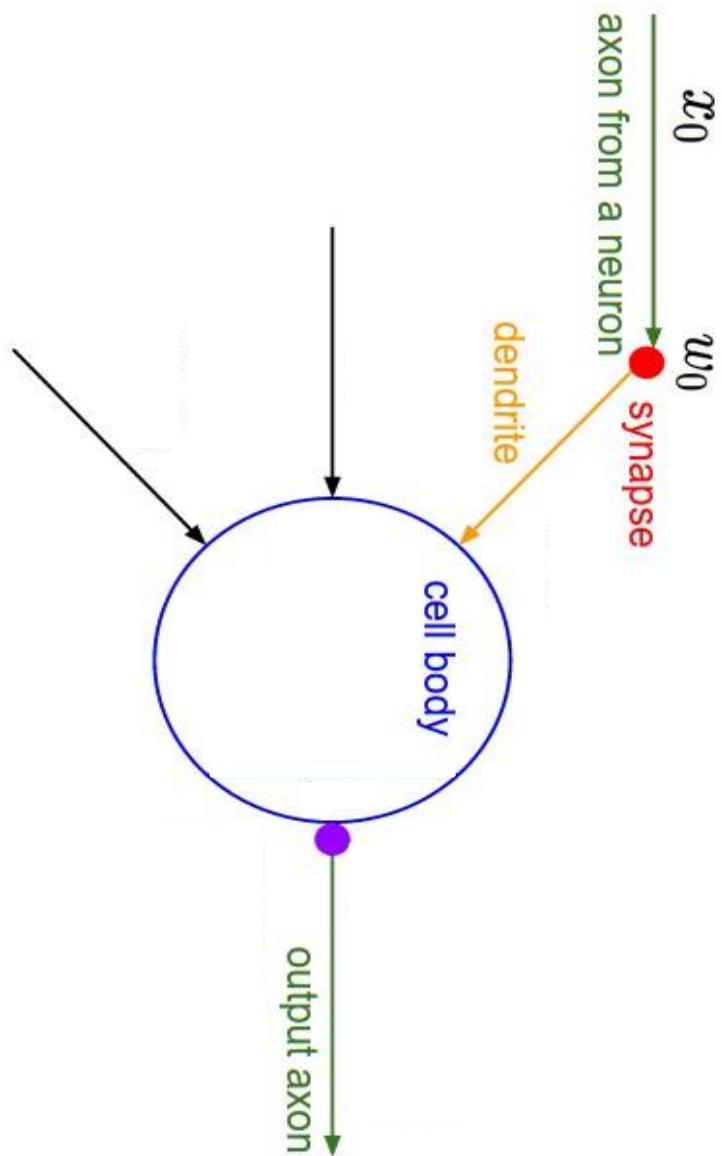
# Funções de Ativação – Inspiração Biológica



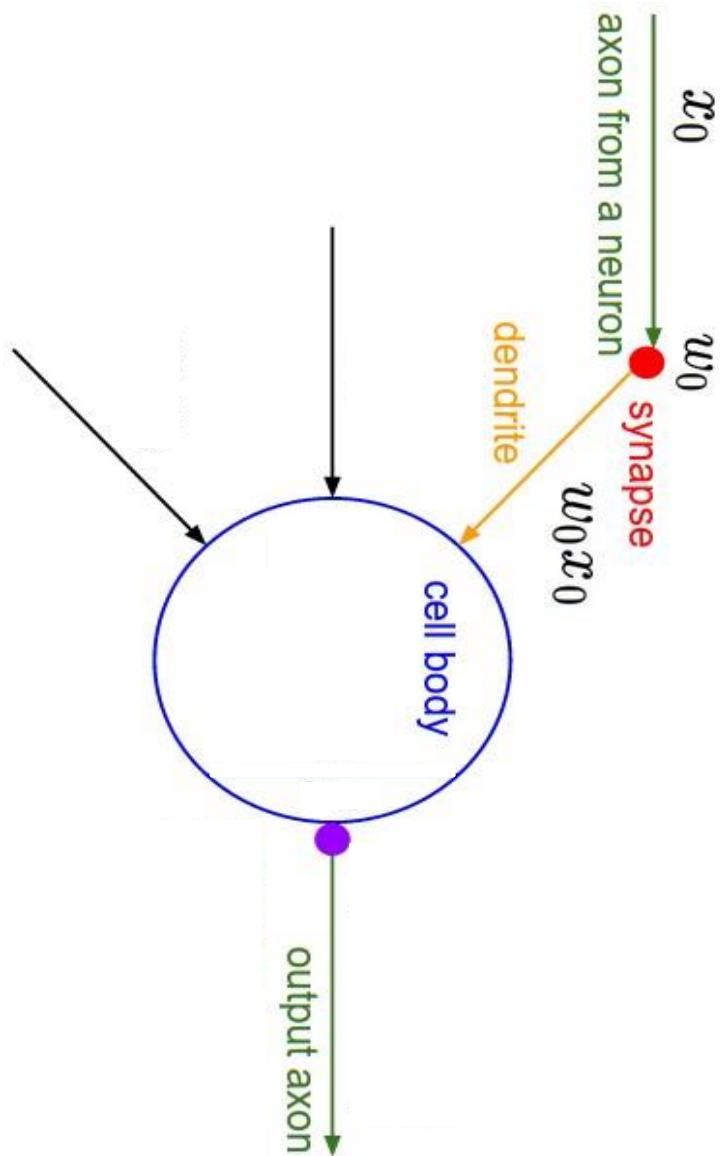
# Funções de Ativação



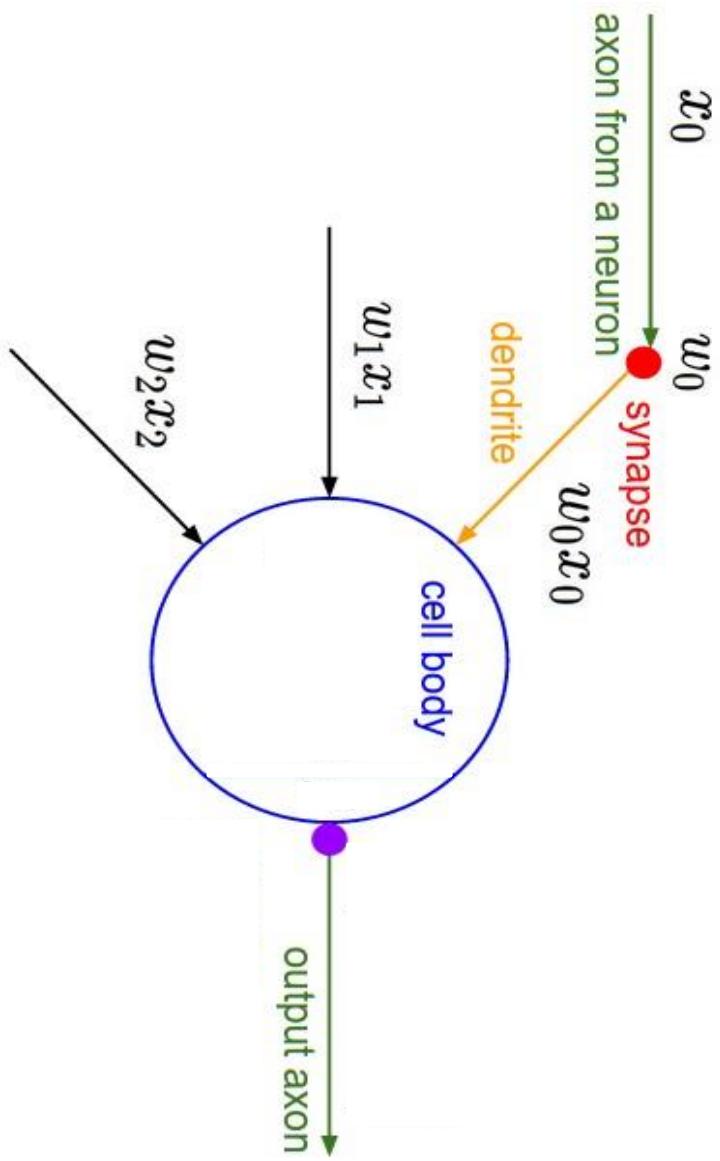
# Funções de Ativação



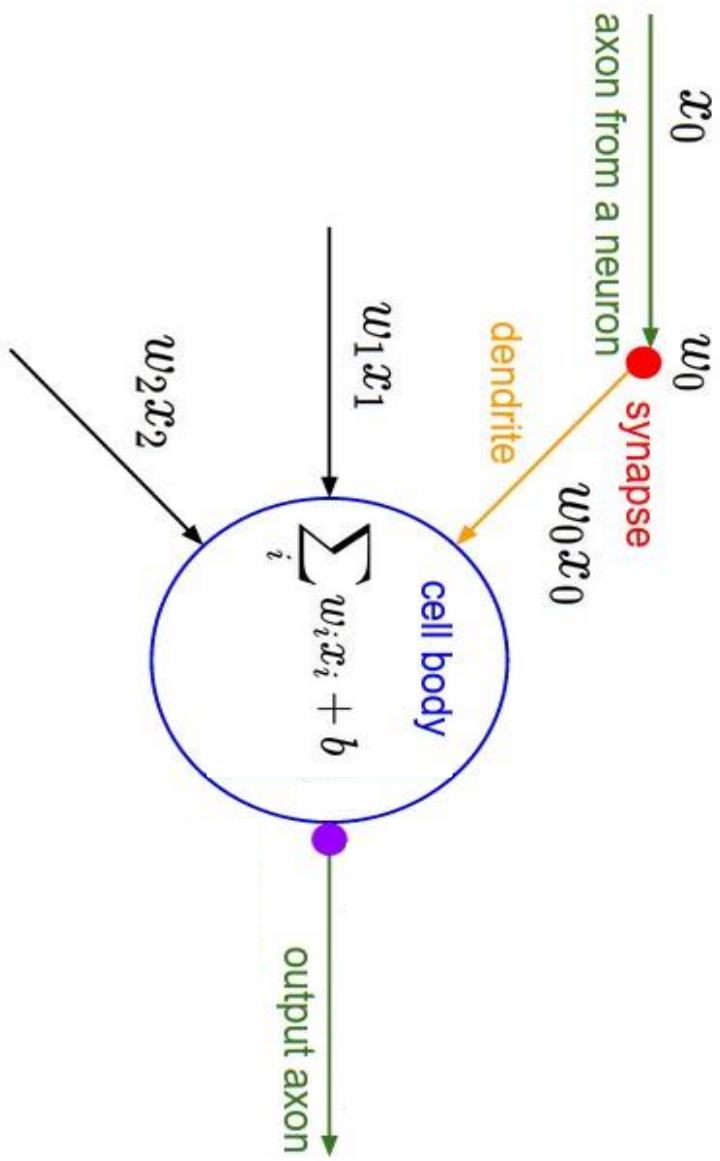
# Funções de Ativação



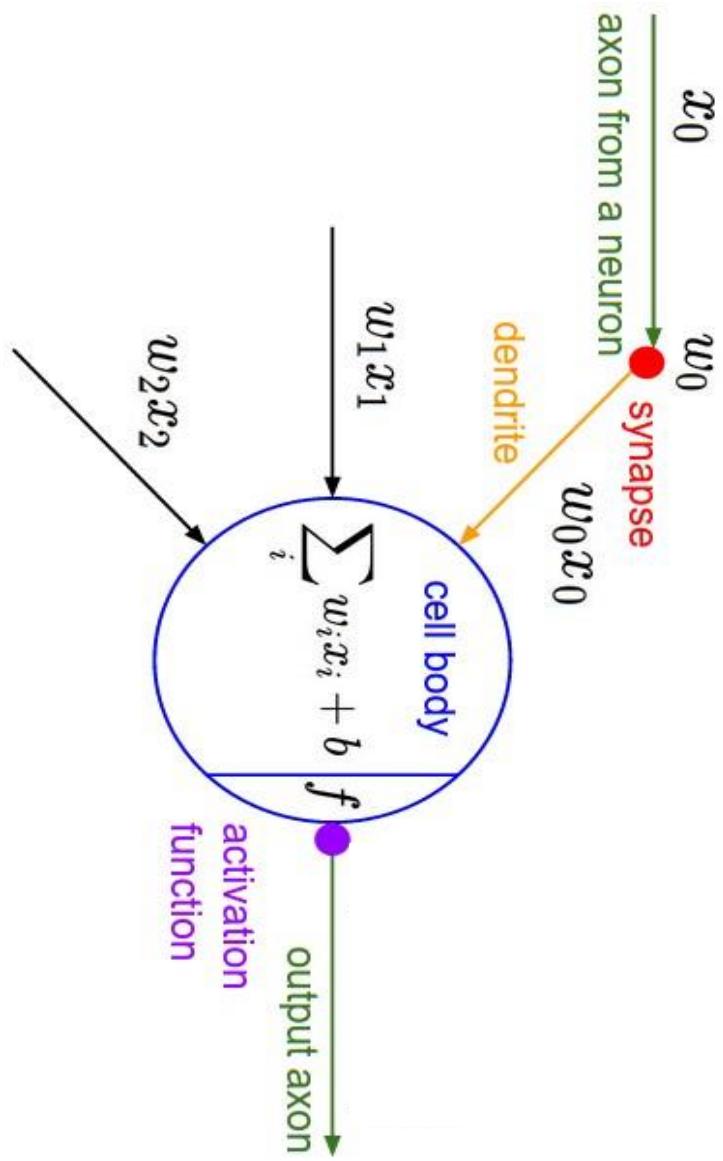
# Funções de Ativação



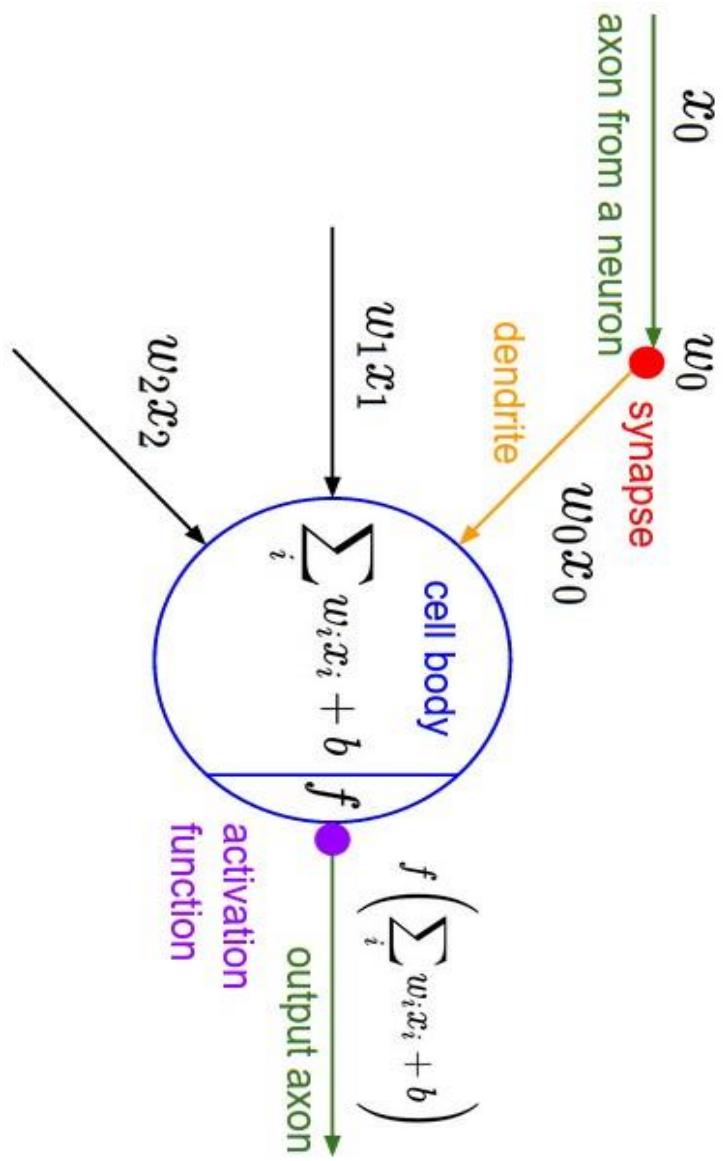
# Funções de Ativação



# Funções de Ativação



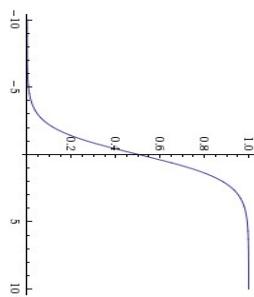
# Funções de Ativação



# Algumas Funções de Ativação

## Sigmoid

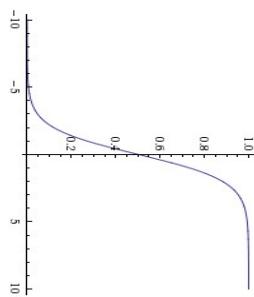
$$\sigma(x) = 1/(1 + e^{-x})$$



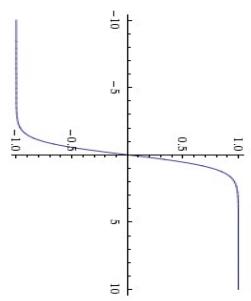
# Algumas Funções de Ativação

## Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



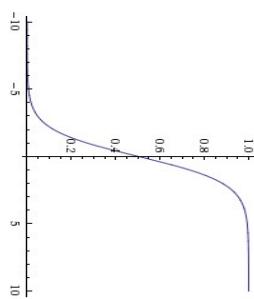
## Tanh tanh(x)



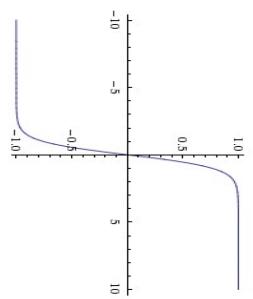
# Algumas Funções de Ativação

## Sigmoid

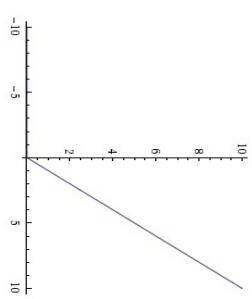
$$\sigma(x) = 1/(1 + e^{-x})$$



## Tanh

$$\tanh(x)$$


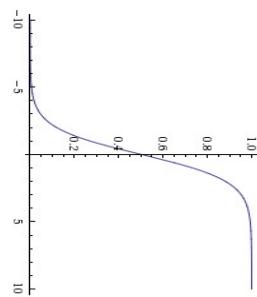
## ReLU

$$\max(0, x)$$


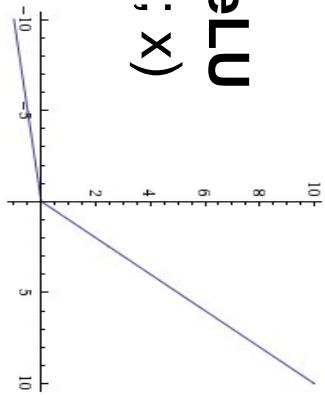
# Algumas Funções de Ativação

## Sigmoid

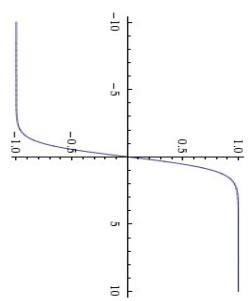
$$\sigma(x) = 1/(1 + e^{-x})$$



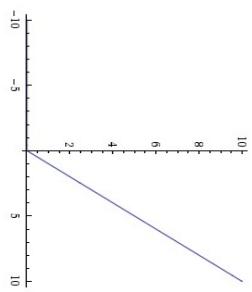
## Leaky ReLU $\max(0, \alpha x; x)$



## Tanh $\tanh(x)$



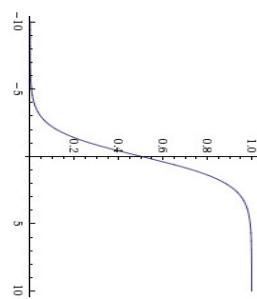
## ReLU $\max(0, x)$



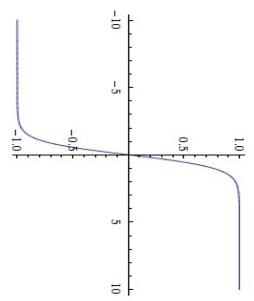
# Algumas Funções de Ativação

## Sigmoid

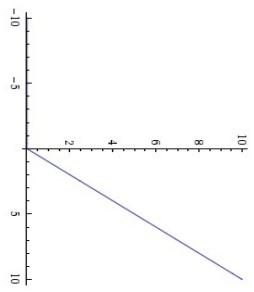
$$\sigma(x) = 1/(1 + e^{-x})$$



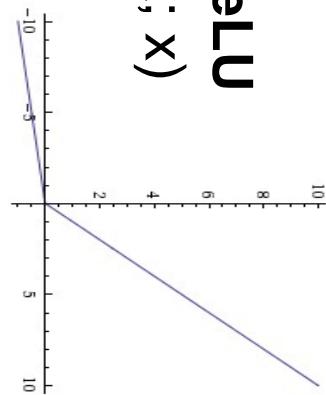
## Tanh tanh(x)



## ReLU max(0,x; x)

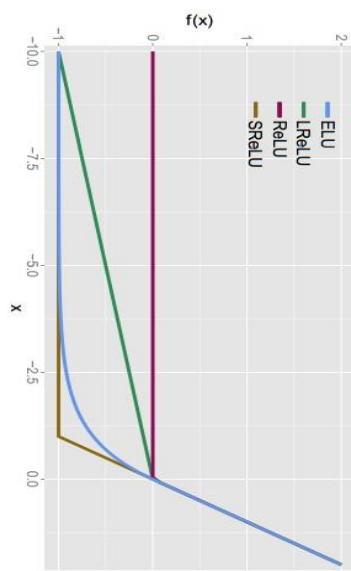


## Leaky ReLU $\max(0, \alpha x; x)$

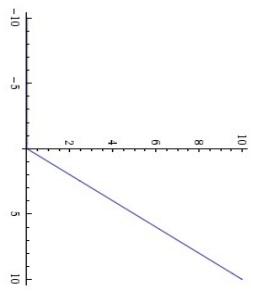


## ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



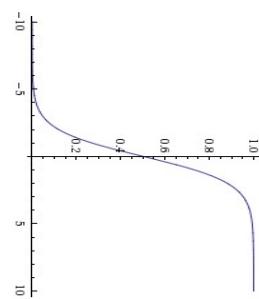
## ReLU max(0,x; x)



# Algumas Funções de Ativação

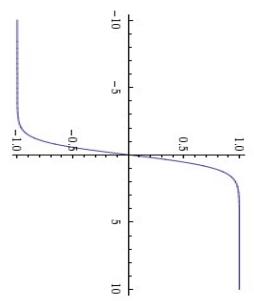
## Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



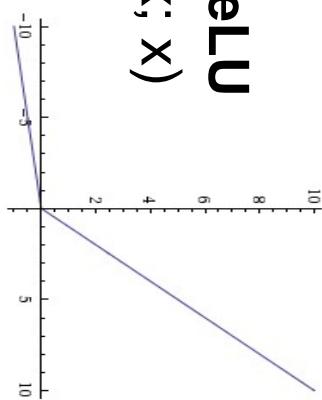
## Tanh

$$\tanh(x)$$



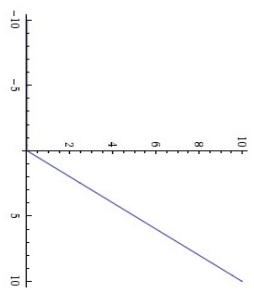
## ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



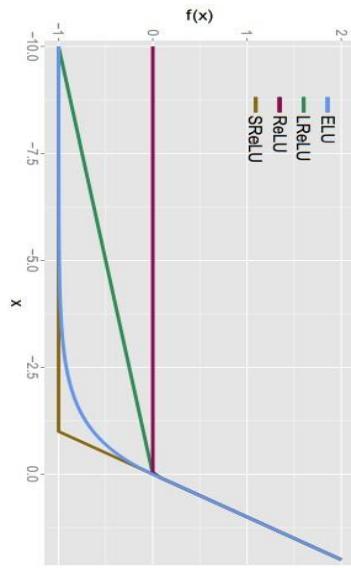
## ReLU

$$\max(0, x)$$



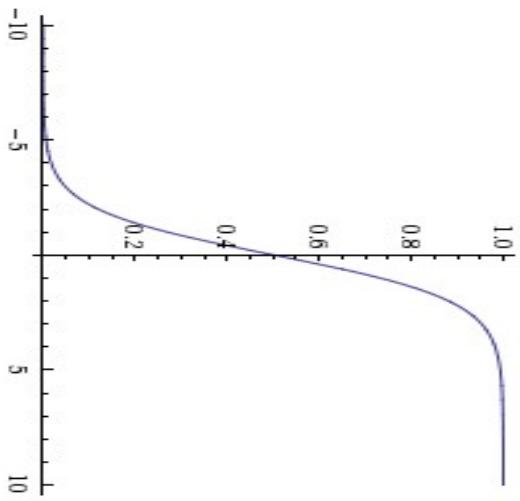
## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$



# Função de Ativação – Sígmide

- Historicamente popular, uma vez que tem uma boa interpretação como uma "taxa de disparo" de um neurônio saturado

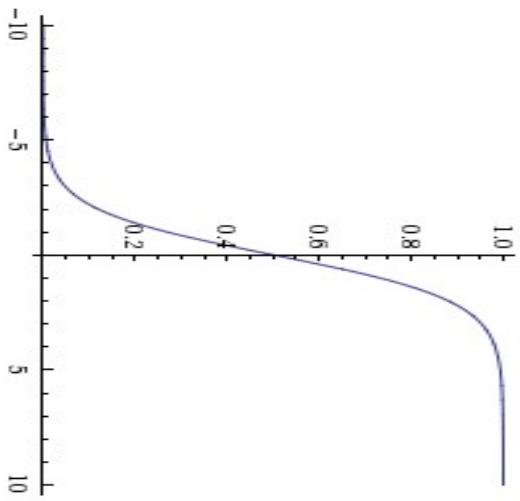


**Função Sígmide (logística)**

$$\sigma(x) = 1/(1 + e^{-x})$$

# Função de Ativação – Sigmoid

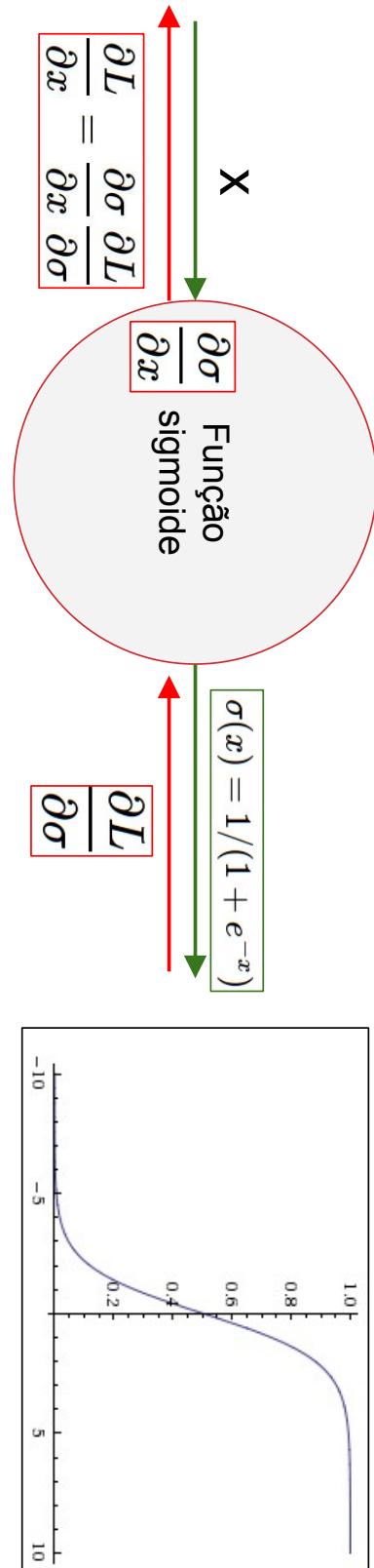
- Historicamente popular, uma vez que tem uma boa interpretação como uma "taxa de disparo" de um neurônio saturado
- "Espreme" os valores para o intervalo  $[0,1]$  –  
**pode "matar" (zerar) os gradientes**



## Função Sigmoid (logística)

$$\sigma(x) = 1 / (1 + e^{-x})$$

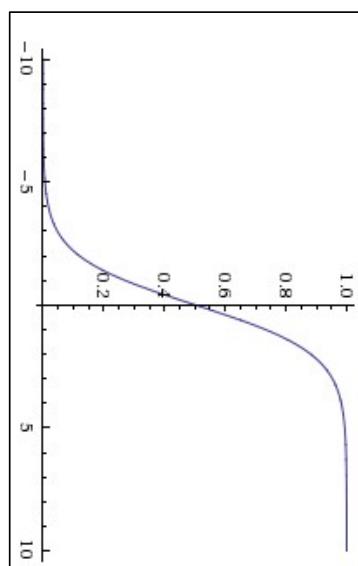
# Função de Ativação – Sísmoide



O que acontece quando  $x = -10$ ?

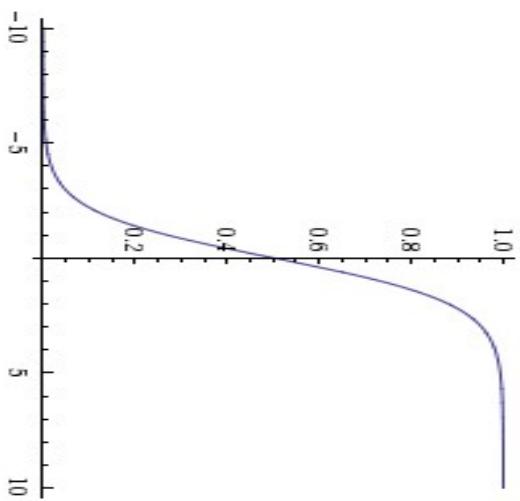
O que acontece quando  $x = 0$ ?

O que acontece quando  $x = 10$ ?



# Função de Ativação – Sigmoid

- Historicamente popular, uma vez que tem uma boa interpretação como uma "taxa de disparo" de um neurônio saturado
- "Espreme" os valores para o intervalo  $[0,1]$  – pode "matar" (zerar) os gradientes
- Não é centrada em torno de zero

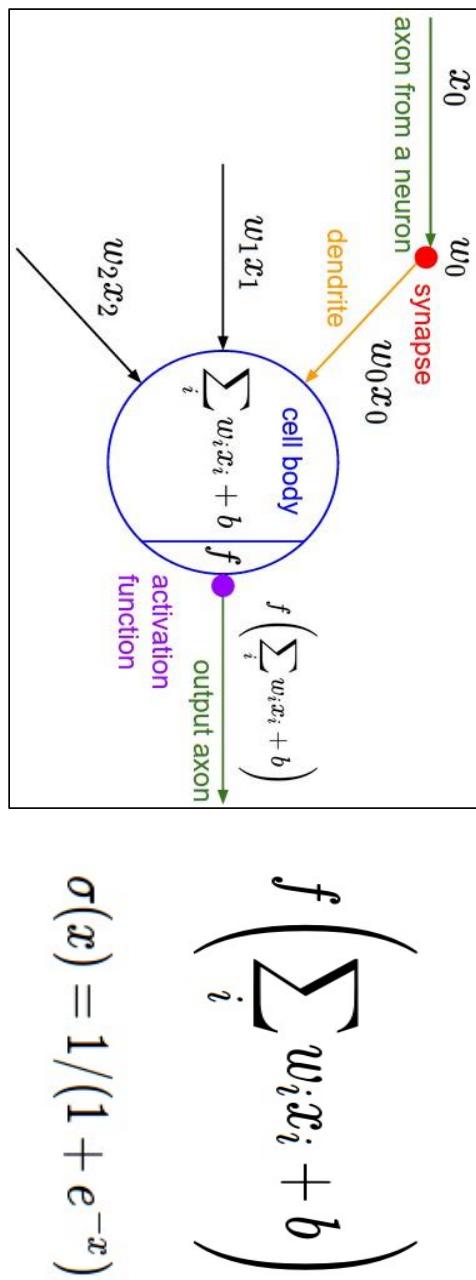


## Função Sigmoid (logística)

$$\sigma(x) = 1/(1 + e^{-x})$$

# Função de Ativação – Sísmoide

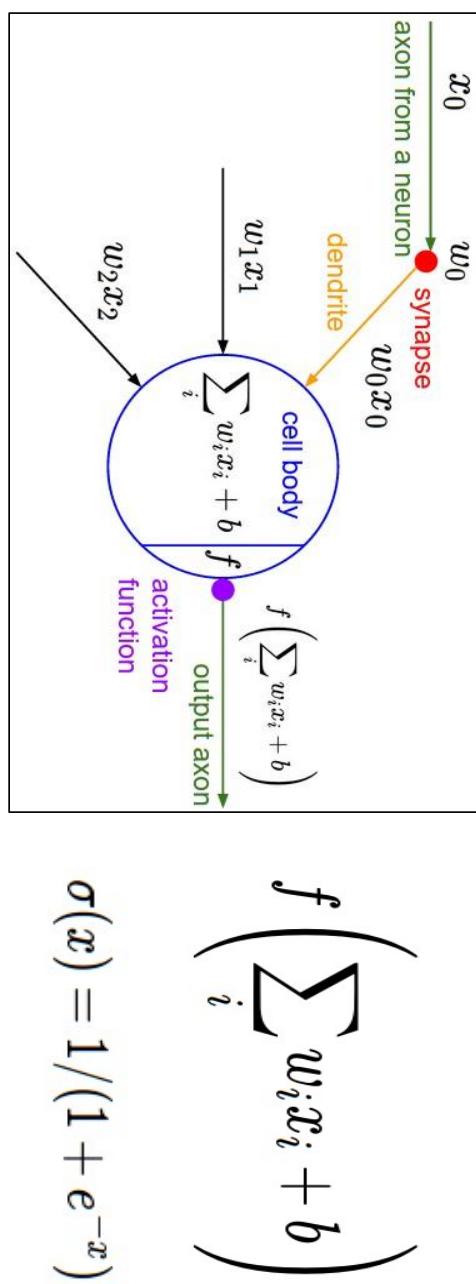
Considere o que acontece quando a entrada de um neurônio ( $x$ ) é sempre positiva:



O que se pode dizer sobre os gradientes em relação a  $w$ ?

# Função de Ativação – Símoide

Considere o que acontece quando a entrada de um neurônio ( $x$ ) é sempre positiva:



O que se pode dizer sobre os gradientes em relação a  $w$ ?

**Sempre todos positivos ou negativos :(**

# Função de Ativação – Sigmoid

Considere o que acontece quando a entrada de um neurônio ( $x$ ) é sempre positiva:

$$f\left(\sum_i w_i x_i + b\right)$$

|   |
|---|
| Direções possíveis para atualização de gradientes |
| Direções possíveis para atualização de gradientes |

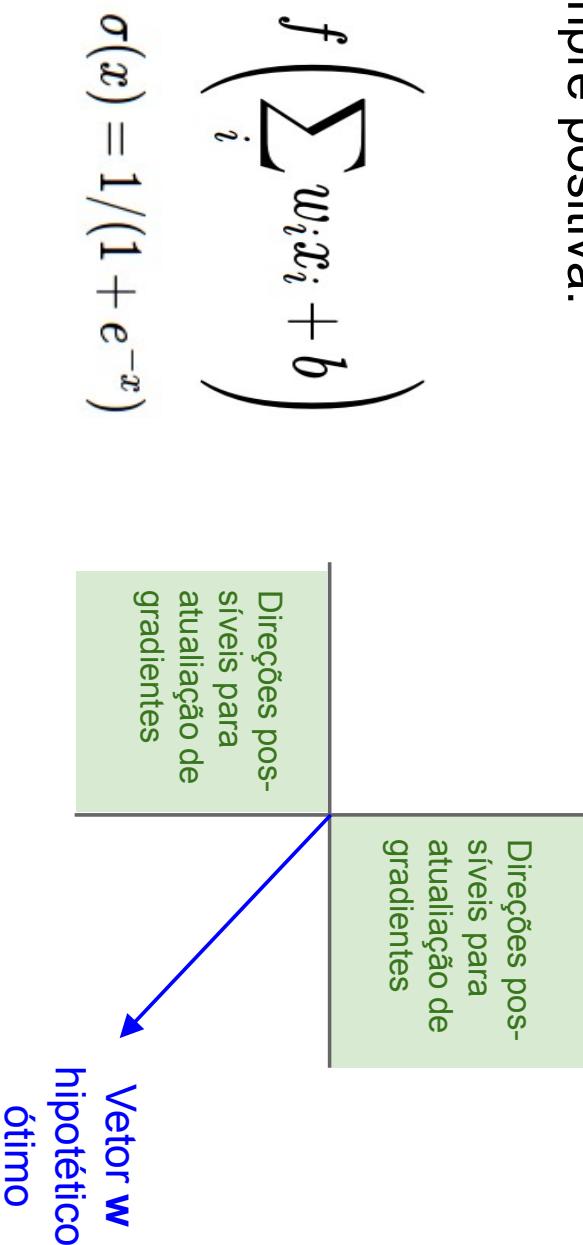
$$\sigma(x) = 1/(1 + e^{-x})$$

O que se pode dizer sobre os gradientes em relação a  $w$ ?

**Sempre todos positivos ou negativos :(**

# Função de Ativação – Sigmoid

Considere o que acontece quando a entrada de um neurônio ( $x$ ) é sempre positiva:

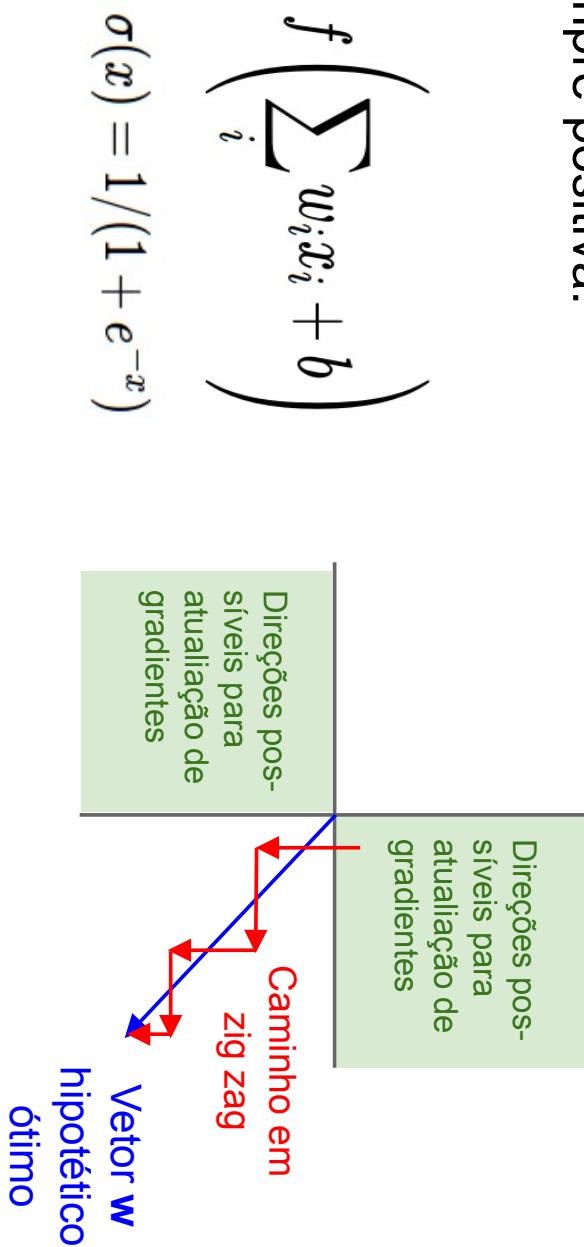


O que se pode dizer sobre os gradientes em relação a  $w$ ?

**Sempre todos positivos ou negativos :(**

# Função de Ativação – Sigmoid

Considere o que acontece quando a entrada de um neurônio ( $x$ ) é sempre positiva:

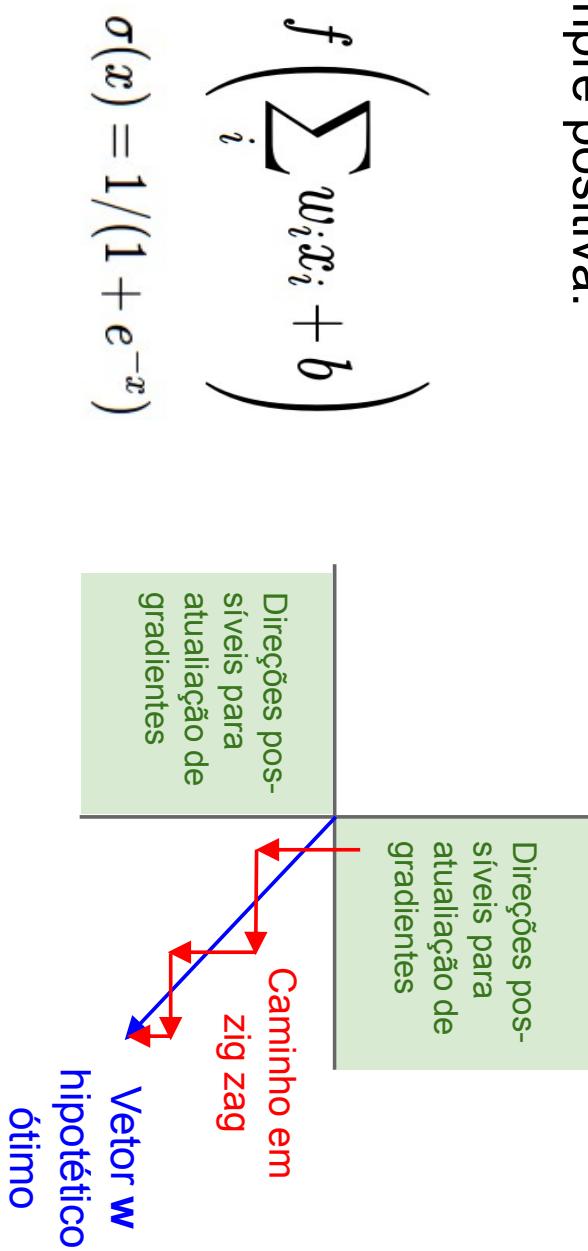


O que se pode dizer sobre os gradientes em relação a  $w$ ?

**Sempre todos positivos ou negativos :(**

# Função de Ativação – Sigmoid

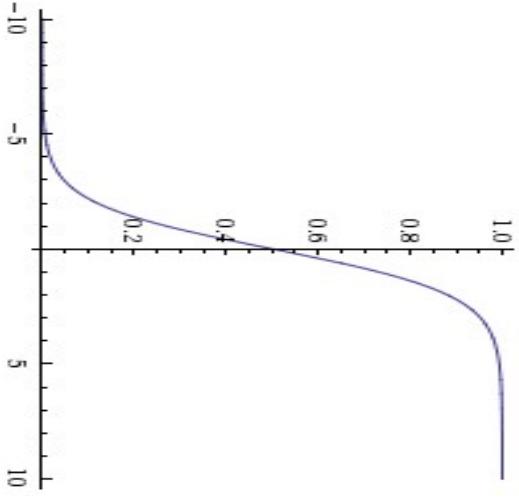
Considere o que acontece quando a entrada de um neurônio ( $x$ ) é sempre positiva:



O que se pode dizer sobre os gradientes em relação a  $w$ ?

**Sempre todos positivos ou negativos :(**  
**É também por isso que se deseja dados com média zero!**

# Função de Ativação – Sigmoid

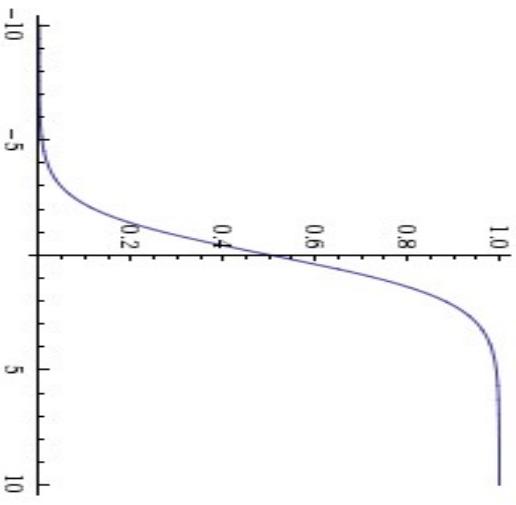


- Historicamente popular, uma vez que tem uma boa interpretação como uma "taxa de disparo" de um neurônio saturado
- "Espreme" os valores para o intervalo [0,1] – pode "matar" (zerar) os gradientes
- Não é centrada em torno de zero
- O uso de `exp()` é um pouco "caro"
- Não é adequada para tratamento de imagens (substituída por ReLU)

## Função Sigmoid (logística)

$$\sigma(x) = 1 / (1 + e^{-x})$$

# Função de Ativação – Sigmoid



- Historicamente popular, uma vez que tem uma boa interpretação como uma "taxa de disparo" de um neurônio saturado

- "Espreme" os valores para o intervalo  $[0,1]$  – pode "matar" (zerar) os gradientes

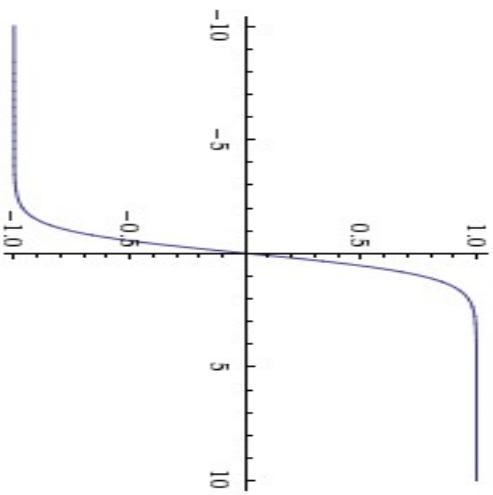
- Não é centrada em torno de zero
- O uso de `exp()` é um pouco "caro"
- Não é adequada para tratamento de imagens (substituída por ReLU)

## Função Sigmoid (logística)

$$\sigma(x) = 1/(1 + e^{-x})$$

- É um elemento chave em redes LSTM – "controle de sinais"
- Ideal para aprendizado de funções "lógicas" – pois produz resultado no intervalo  $[0, 1]$

# Função de Ativação – Tanh

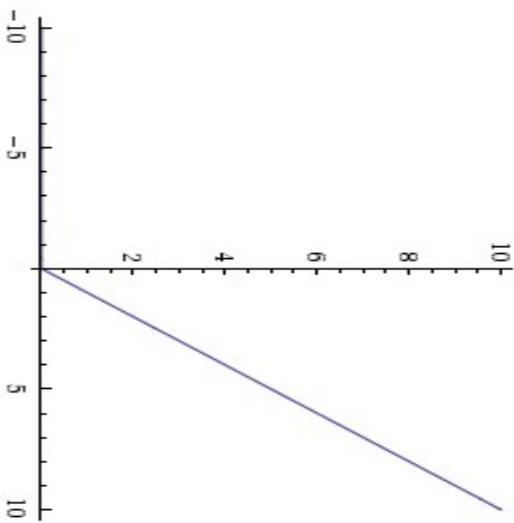


- “Espreme” os valores para o intervalo [-1,1]
- É centrada em zero (que é bom)
- Ainda “mata” os gradientes quando saturada :(
- Também é usada em redes LSTM para valores limitados e com sinal
- Não é “boa” para funções binárias

**Tanh(x)**

[LeCun et al., 1991]

# Função de Ativação – ReLU



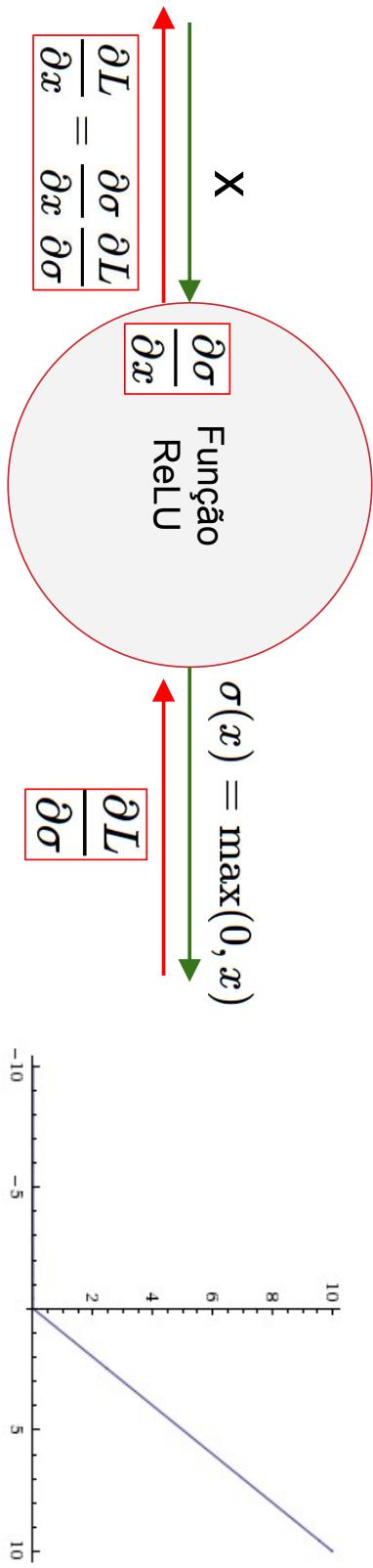
## ReLU

(*Rectified Linear Unit*)

[Krizhevsky et al., 2012]

- **Computa  $f(x) = \max(0, x)$**
- **Não fica saturada na região positiva**
- **É muito eficiente computacionalmente**
- **Converge mais rapidamente que sigmoide/tanh sobre imagens ( $\approx 6x$ )**
- **Sua saída não é centrada em zero**
- **Não é adequada para funções lógicas recorrentes**
- **Não é usada no controle de redes recorrentes**
- **Apresenta uma inconveniência : qual o gradiente quando  $x < 0$ ?**

# Função de Ativação – ReLU

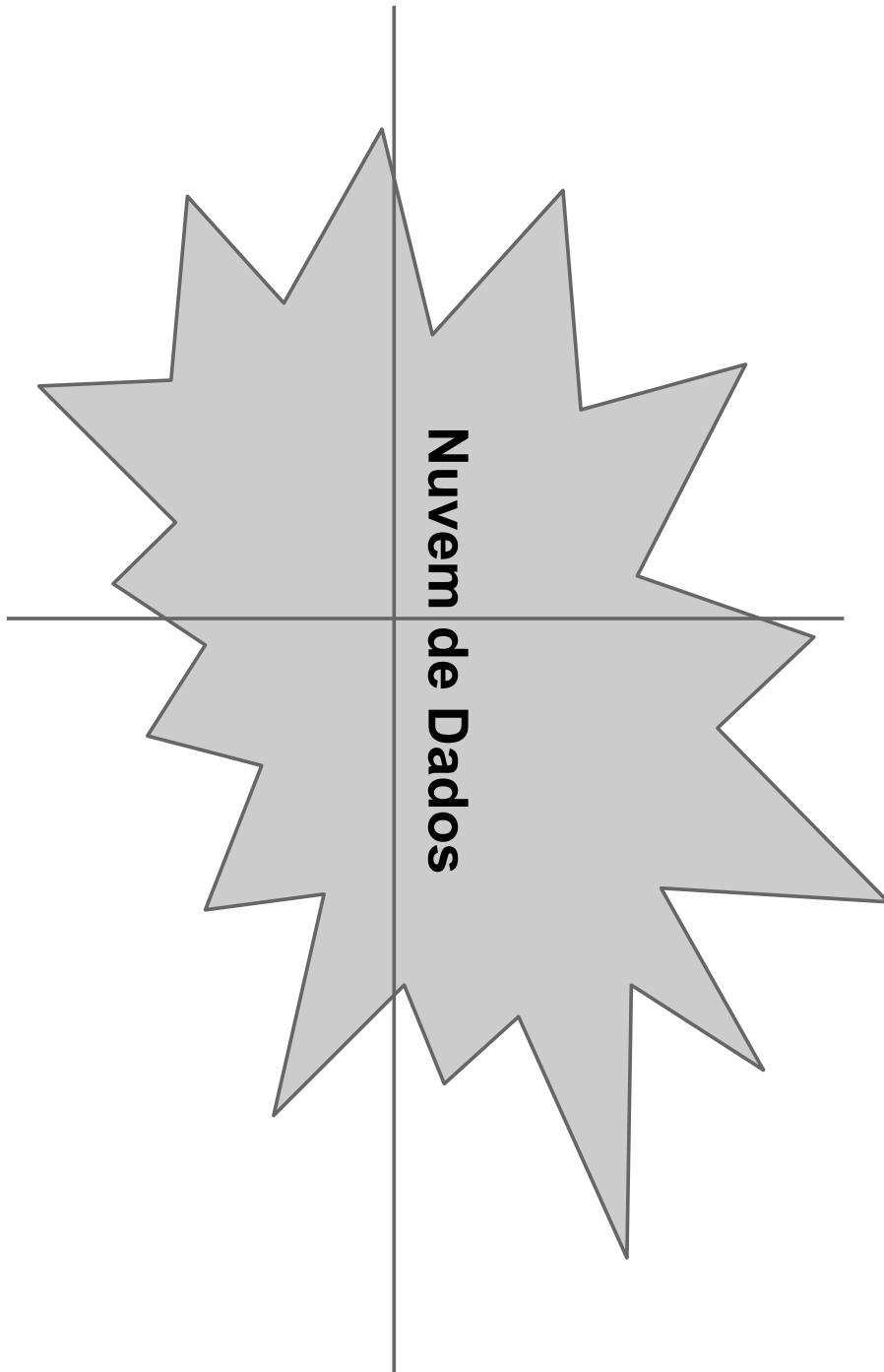


O que acontece quando  $x = -10$ ?

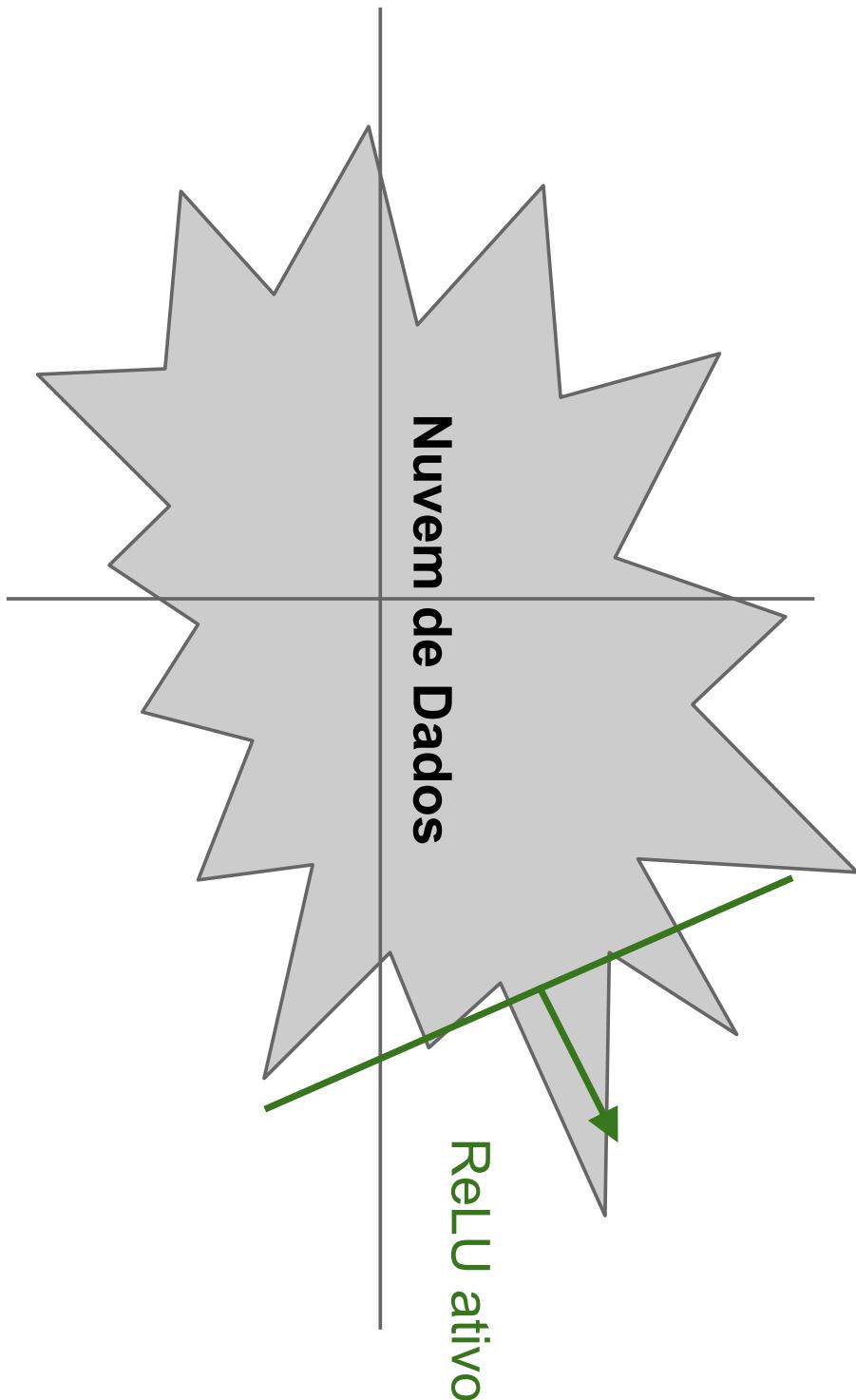
O que acontece quando  $x = 0$ ?

O que acontece quando  $x = 10$ ?

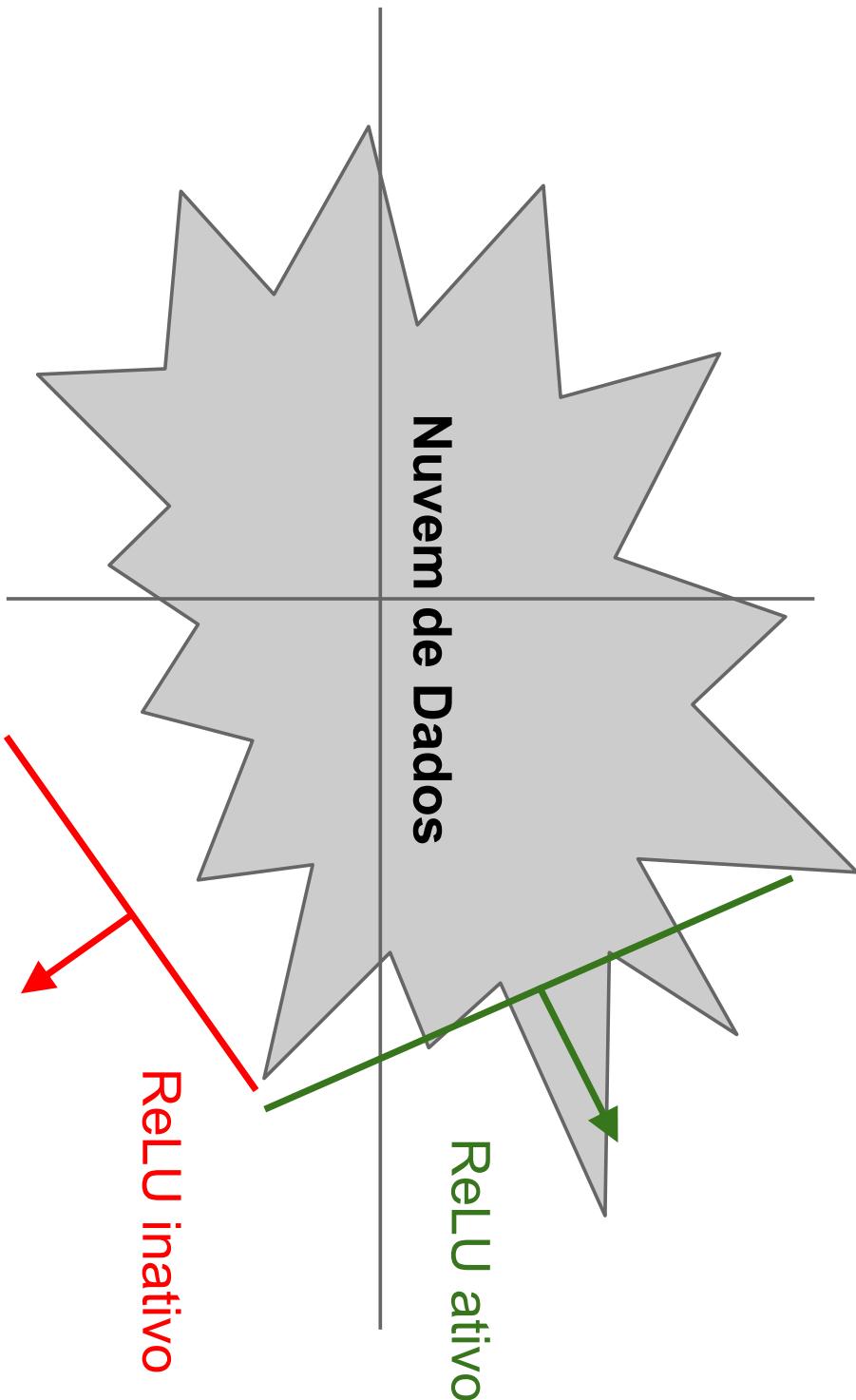
# Função de Ativação – ReLU



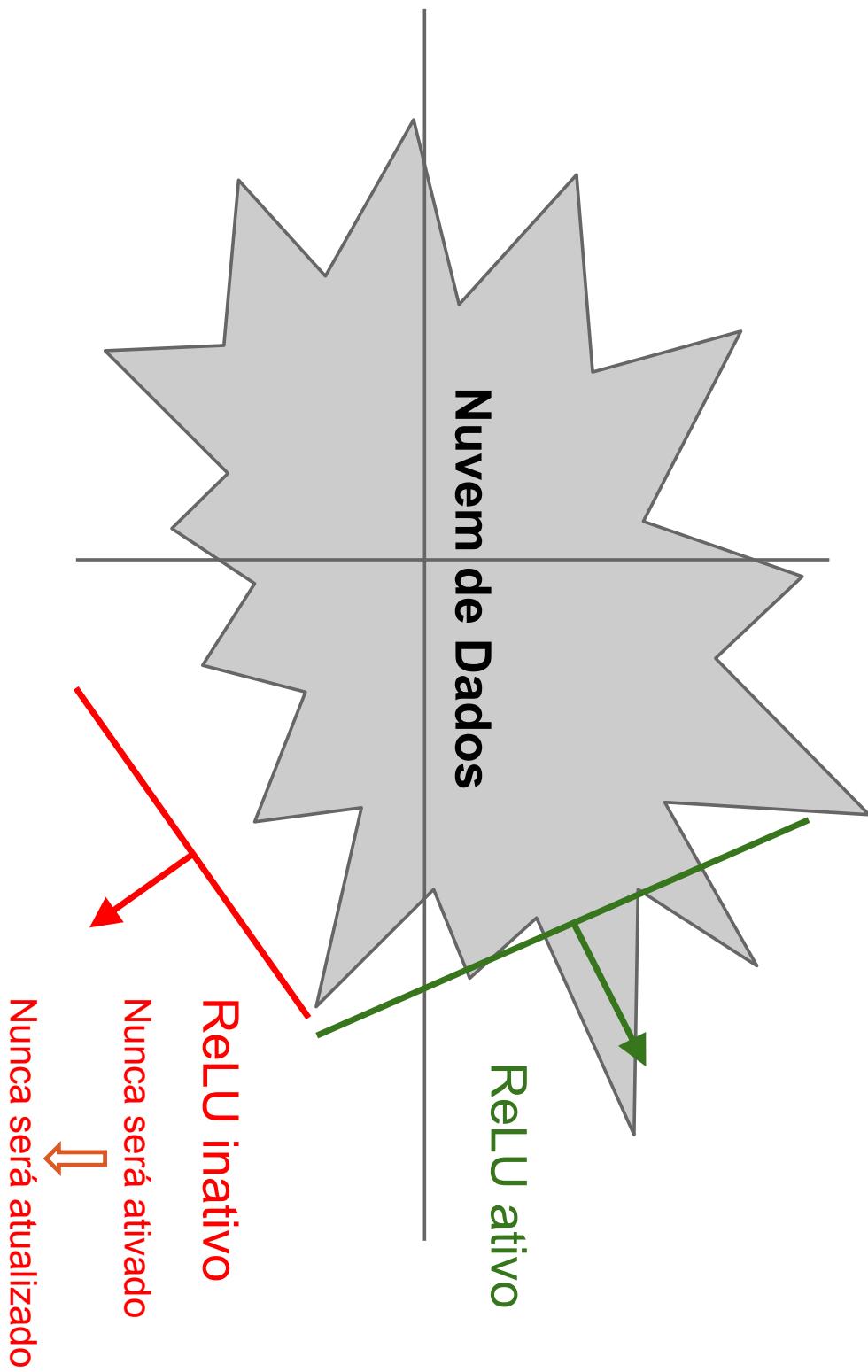
# Função de Ativação – ReLU



# Função de Ativação – ReLU

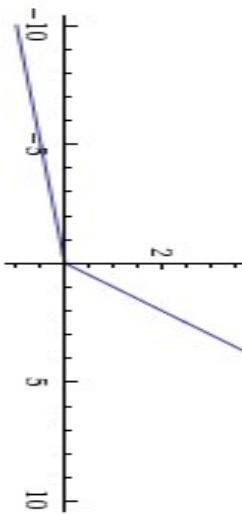


# Função de Ativação – ReLU



# Função de Ativação – Leaky ReLU

- Não satura nunca
- É computacionalmente eficiente
- Converge mais rapidamente que sigmoide/tanh sobre imagens ( $\approx 6x$ )
- Nunca “mata” os gradientes



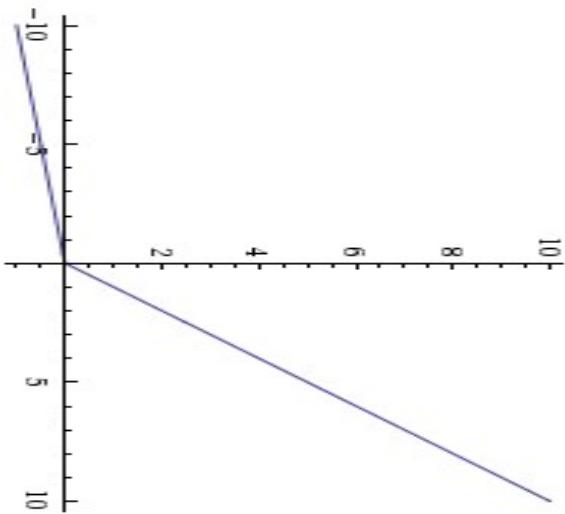
## Leaky ReLU

$$f(x) = \max(0.01x, x)$$

[Mass et al., 2013]  
[He et al., 2015]

# Função de Ativação – Leaky ReLU

- Não satura nunca
- É computacionalmente eficiente
- Converge mais rapidamente que sigmoide/tanh sobre imagens ( $\approx 6x$ )
- Nunca “mata” os gradientes



## Leaky ReLU

$$f(x) = \max(0.01x, x)$$

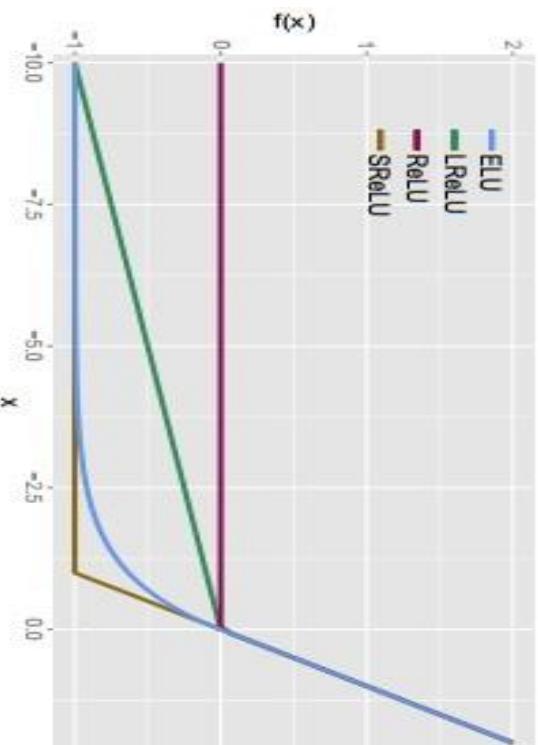
PRelu (Parametric Rectifier Linear Unit)

$$f(x) = \max(\alpha x, x)$$

[Mass et al., 2013]  
[He et al., 2015]

BackProp sobre  $\alpha$   
(parâmetro)

# Função de Ativação – ELU



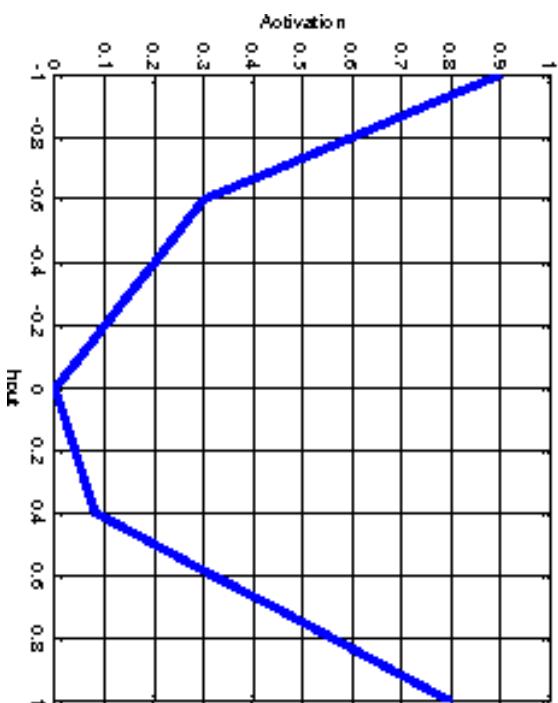
- Apresenta todos os benefícios de ReLU
- Não “mata” os gradientes
- Produz saídas com médias próximas de zero
- **Necessita de `exp()` para seu cálculo**

**ELU**  
(*Exponential Linear Units*)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

[Clevert et al., 2015]

# Função de Ativação – Maxout



- Não possui a forma básica de produto interno seguido por não-linearidade
- Generaliza ReLU e Leaky ReLU
- Apresenta um regime linear! Nunca satura!  
Nunca “mata” os gradientes!

Problema: aumenta o número  
de parâmetros por neurônio :)

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

[Goodfellow et al., 2013]

# Redes Neurais e Aprendizagem Profunda

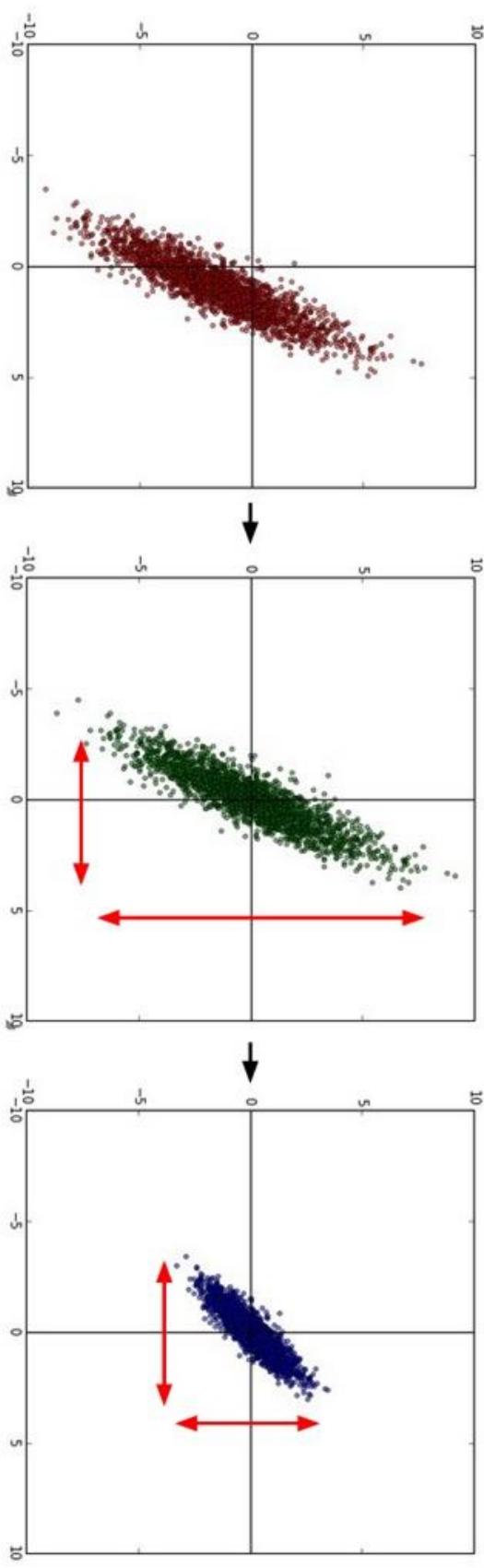
## REDES NEURAIS ARTIFICIAIS PREPARAÇÃO DE DADOS / INICIALIZAÇÃO

Zenilton K. G. Patrocínio Jr

[zenilton@pucminas.br](mailto:zenilton@pucminas.br)

# Pré-Processamento de Dados – Básico

Dados originais  
Dados centrados  
em zero  
Dados normalizados

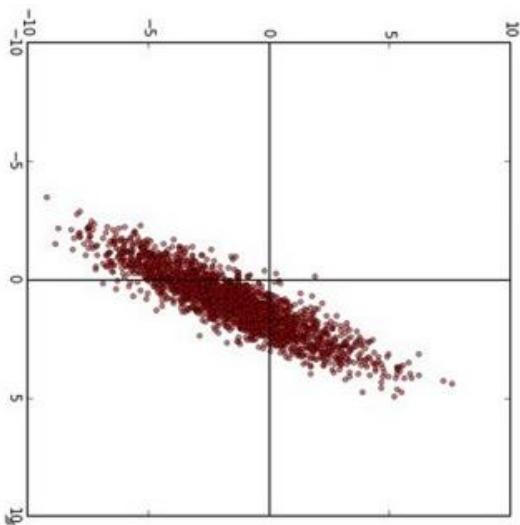


```
X -= np.mean(X, axis = 0)
```

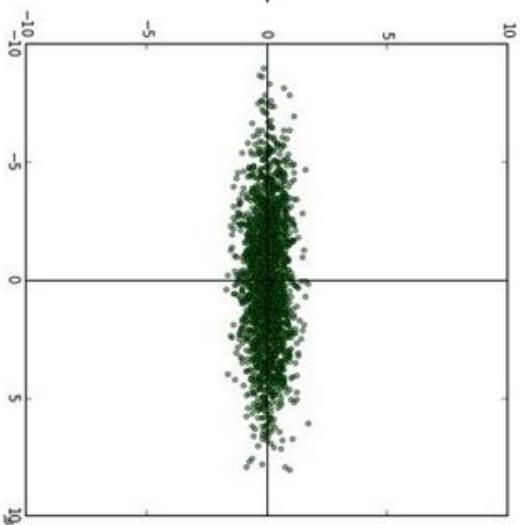
```
X /= np.std(X, axis = 0)
```

# Pré-Processamento de Dados – Avançado

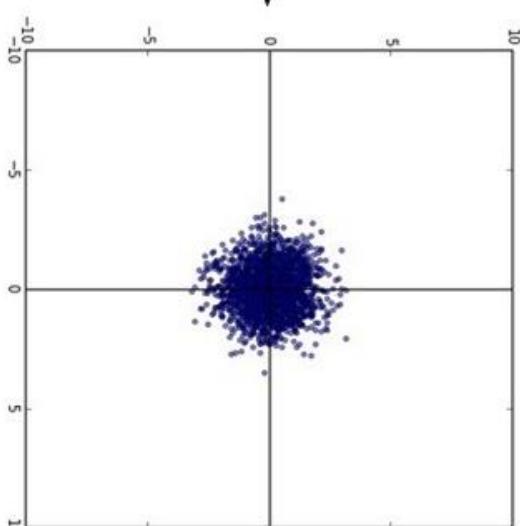
Dados originais



Dados sem  
correlação



Dados sem correlação  
e variância 1



Matriz de covariância  
diagonal

Matriz de covariância  
é a identidade

# Pré-Processamento de Imagens

Apenas centrar os dados em zero

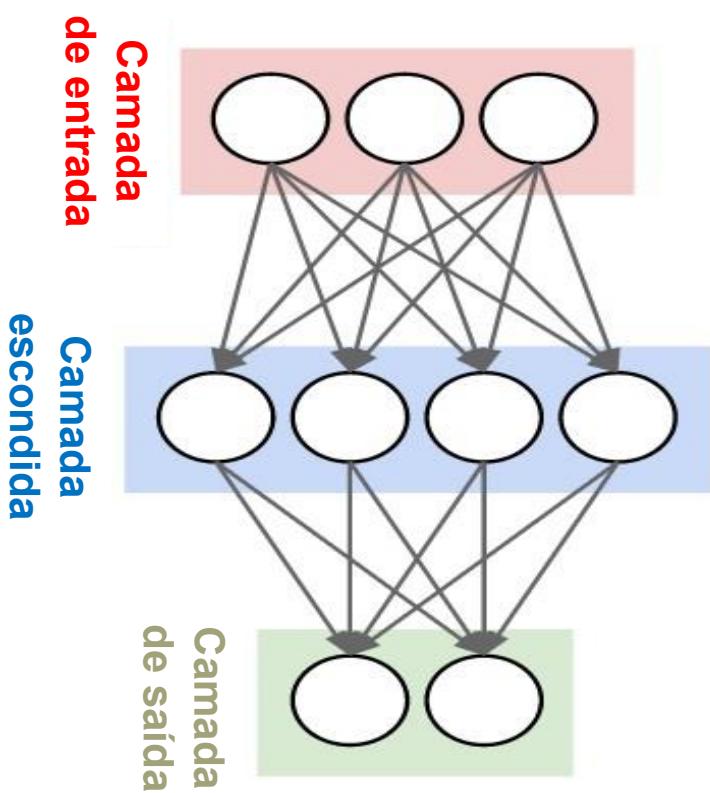
p.ex. considere o dataset CIFAR-10 com imagens [32,32,3]

- Subtrair a imagem média (p.ex. AlexNet)  
imagem média = vetor de dimensões [32,32,3]
- Subtrair a média por canal (p.ex. VGGNet)  
média ao longo de cada canal = 3 números

**Não é comum normalizar nem decorrelacionar os pixels de uma imagem**

# Inicialização de Pesos

- Pergunta: que ocorre quando os pesos são inicializados com zero, isto é,  $W = 0$ ?



# Inicialização de Pesos

- Uma primeira ideia: **Usar números randômicos pequenos**  
Usar uma distribuição normal com média zero e desvio padrão  $10^{-2}$

```
W = 0.01 * np.random.randn(D,H)
```

Funciona bem para redes pequenas, mas pode  
levar a distribuições não homogêneas de ativações  
ao longo das camadas de uma rede

# Inicialização de Pesos – Estatísticas de Ativação

Exemplo:

- Rede de 10 camadas
- 500 neurônios por camada
- Ativação Tanh
- Pesos inicializados com

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}

for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer

# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]

for i,H in Hs.items():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

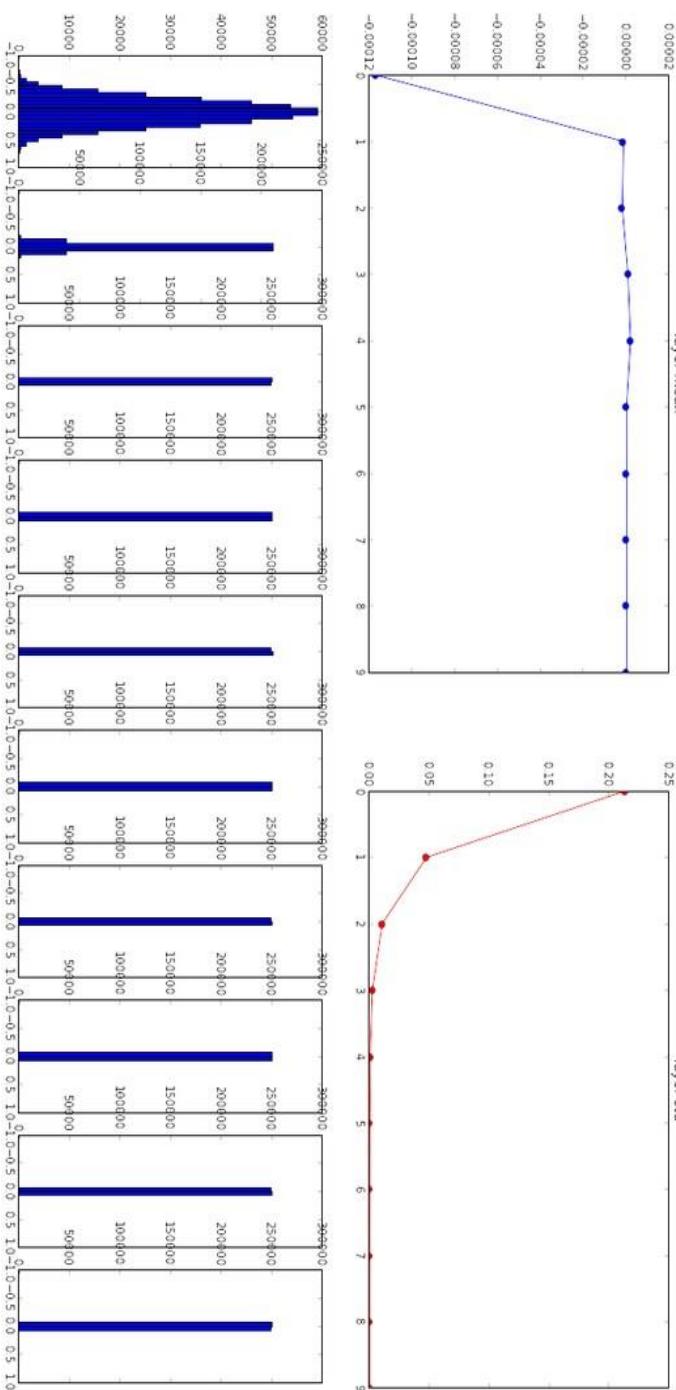
# plot the raw distributions
plt.figure()
for i,H in Hs.items():
    plt.subplot(1, len(Hs), i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

# Inicialização de Pesos – Estatísticas de Ativação

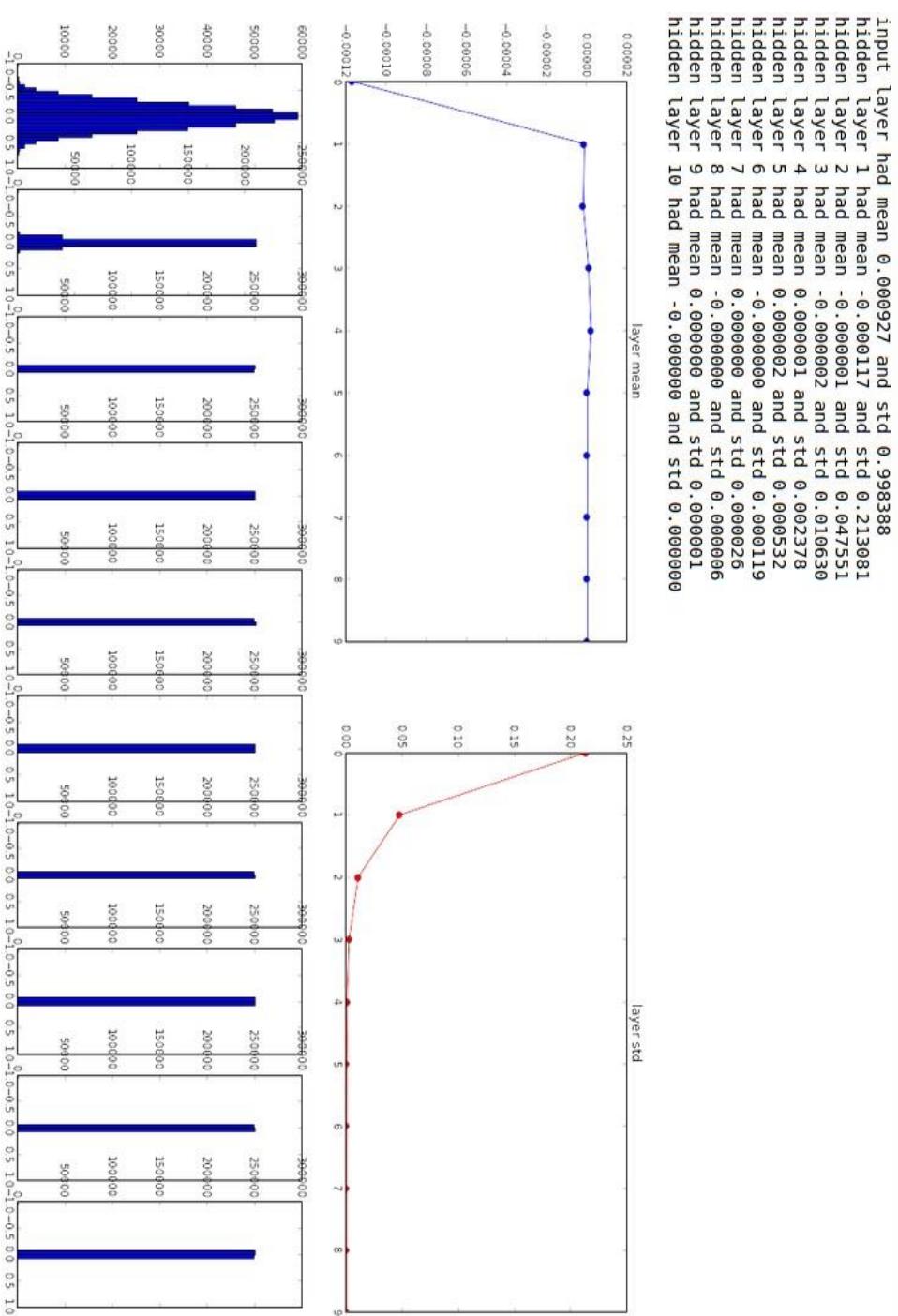
```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000019
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```

layer mean

layer std



# Inicialização de Pesos – Estatísticas de Ativação



Todas as ativações se tornam zero!

P: pense sobre o passo retrógrado. Como são os gradientes?

Dica: lembre que a função de ativação é aplicada após o produto interno  $Wx$

# Inicialização de Pesos – Estatísticas de Ativação

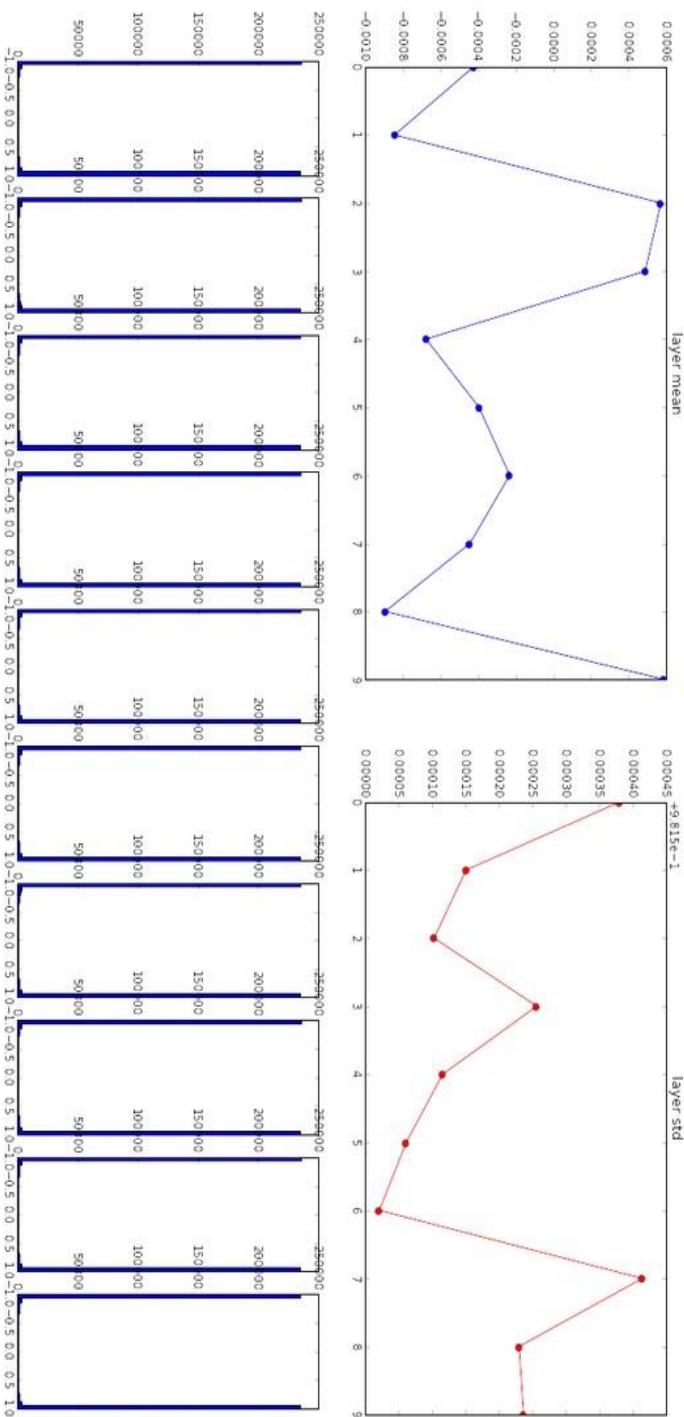
```
input layer had mean 0.661800 and std 1.001311  
hidden layer 1 had mean -0.000430 and std 0.931879  
hidden layer 2 had mean -0.000849 and std 0.931649  
hidden layer 3 had mean 0.000566 and std 0.918601  
hidden layer 4 had mean 0.000483 and std 0.981755  
hidden layer 5 had mean -0.000682 and std 0.931614  
hidden layer 6 had mean -0.000401 and std 0.931569  
hidden layer 7 had mean -0.000237 and std 0.931520  
hidden layer 8 had mean -0.000448 and std 0.931913  
hidden layer 9 had mean -0.000899 and std 0.931728  
hidden layer 10 had mean 0.000584 and std 0.931736
```

```
W = np.random.randn(fan in, fan out) * 1.0 # layer initialization
```

\*1.0 ao invés de \*0.01

**Quase todos os neurônios ficaram completamente saturados em -1 ou +1**

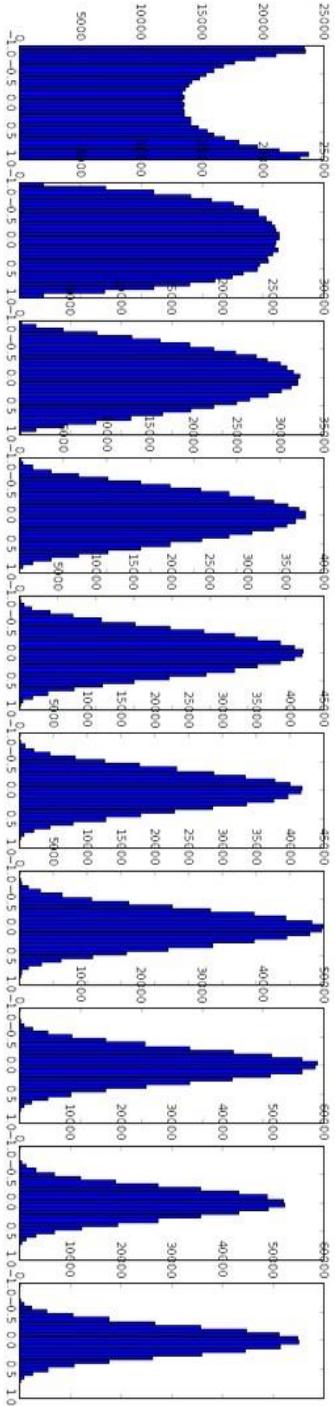
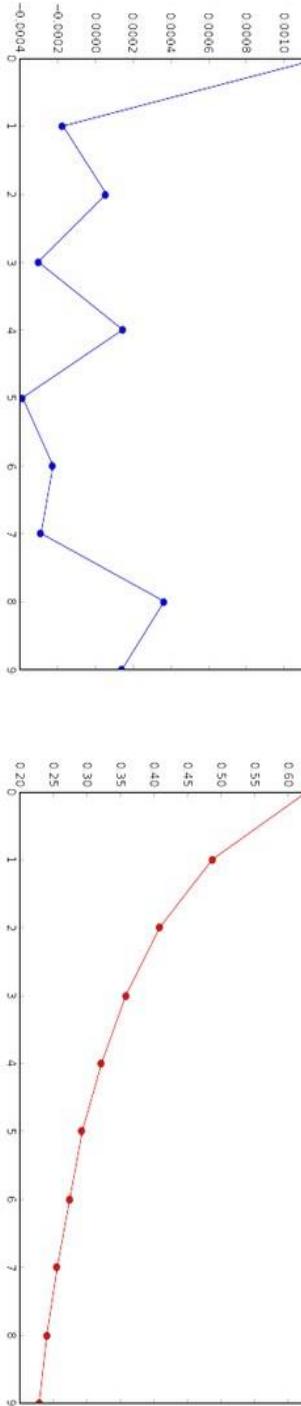
**Gradientes serão todos nulos !!!**



# Inicialização de Pesos – Estatísticas de Ativação

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000228 and std 0.292116
hidden layer 7 had mean -0.000389 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

`W = np.random.randn(fan in, fan out) / np.sqrt(fan in) # layer initialization`



“Inicialização de Xavier”  
[Glorot et al., 2010]

- Derivação matemática supondo ativação linear

## Resultados razoáveis

# Inicialização de Pesos – Estatísticas de Ativação

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.27352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.146299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

```
W = np.random.randn(fan in, fan out) / np.sqrt(fan in) # layer initialization
```

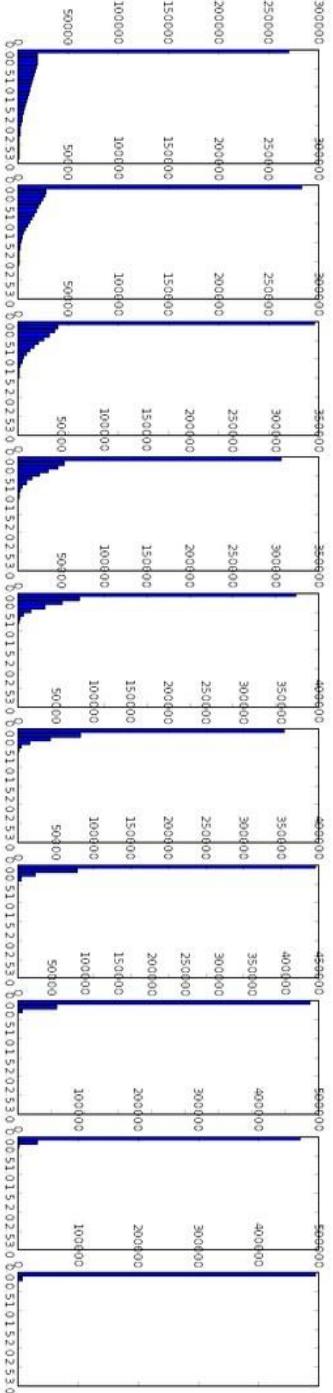


“Inicialização de Xavier”  
[Glorot et al., 2010]

## Resultados razoáveis

- Derivação matemática supondo ativação linear

Mas ao usar ReLU,  
não funciona : (

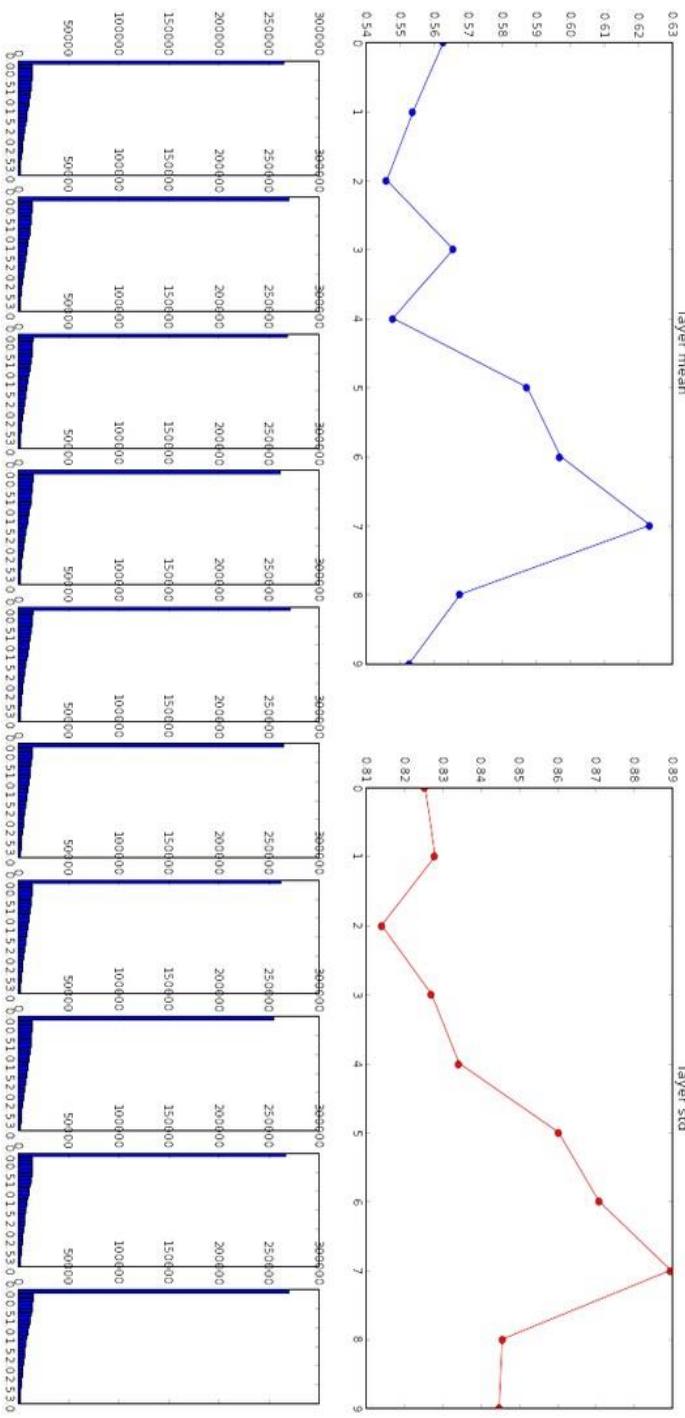


# Inicialização de Pesos – Estatísticas de Ativação

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.828232
hidden layer 2 had mean 0.53614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.818855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834692
hidden layer 6 had mean 0.587103 and std 0.866035
hidden layer 7 had mean 0.596867 and std 0.876610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```

$W = \text{np.random.randn(fan in, fan out)} / \text{np.sqrt(fan in/2)}$  # layer initialization

He et al., 2015  
(observe a divisão por 2)



# Inicialização de Pesos – Estatísticas de Ativação

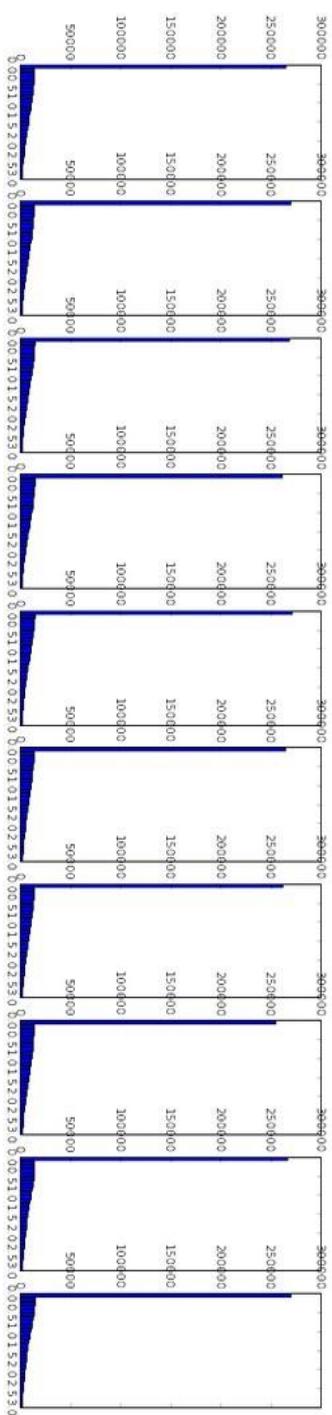
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

```
input layer had mean 0.000501 and std 0.999444  
hidden layer 1 had mean 0.562488 and std 0.828232  
hidden layer 2 had mean 0.536114 and std 0.827835  
hidden layer 3 had mean 0.545867 and std 0.818855  
hidden layer 4 had mean 0.565396 and std 0.826902  
hidden layer 5 had mean 0.547678 and std 0.834692  
hidden layer 6 had mean 0.587103 and std 0.866035  
hidden layer 7 had mean 0.596867 and std 0.876610  
hidden layer 8 had mean 0.623214 and std 0.889348  
hidden layer 9 had mean 0.567498 and std 0.845357  
hidden layer 10 had mean 0.552531 and std 0.844523
```



He et al., 2015  
(observe a divisão por **2**)

O fator 2 não parece muito,  
mas lembre-se de que se  
aplica de forma multiplicativa  
(por exemplo, 150 vezes na  
ResNet)



# Inicialização de Pesos

Initialization adequada é uma área ativa de pesquisa...

- *Understanding the difficulty of training deep feedforward neural networks* by Glorot and Bengio, 2010
- *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks* by Saxe et al., 2013
- *Random walk initialization for training very deep feedforward networks* by Sussillo and Abbott, 2014
- *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* by He et al., 2015
- *Data-dependent Initializations of Convolutional Neural Networks* by Krähenbühl et al., 2015
- *All you need is a good init*, Mishkin and Matas, 2015
- ...