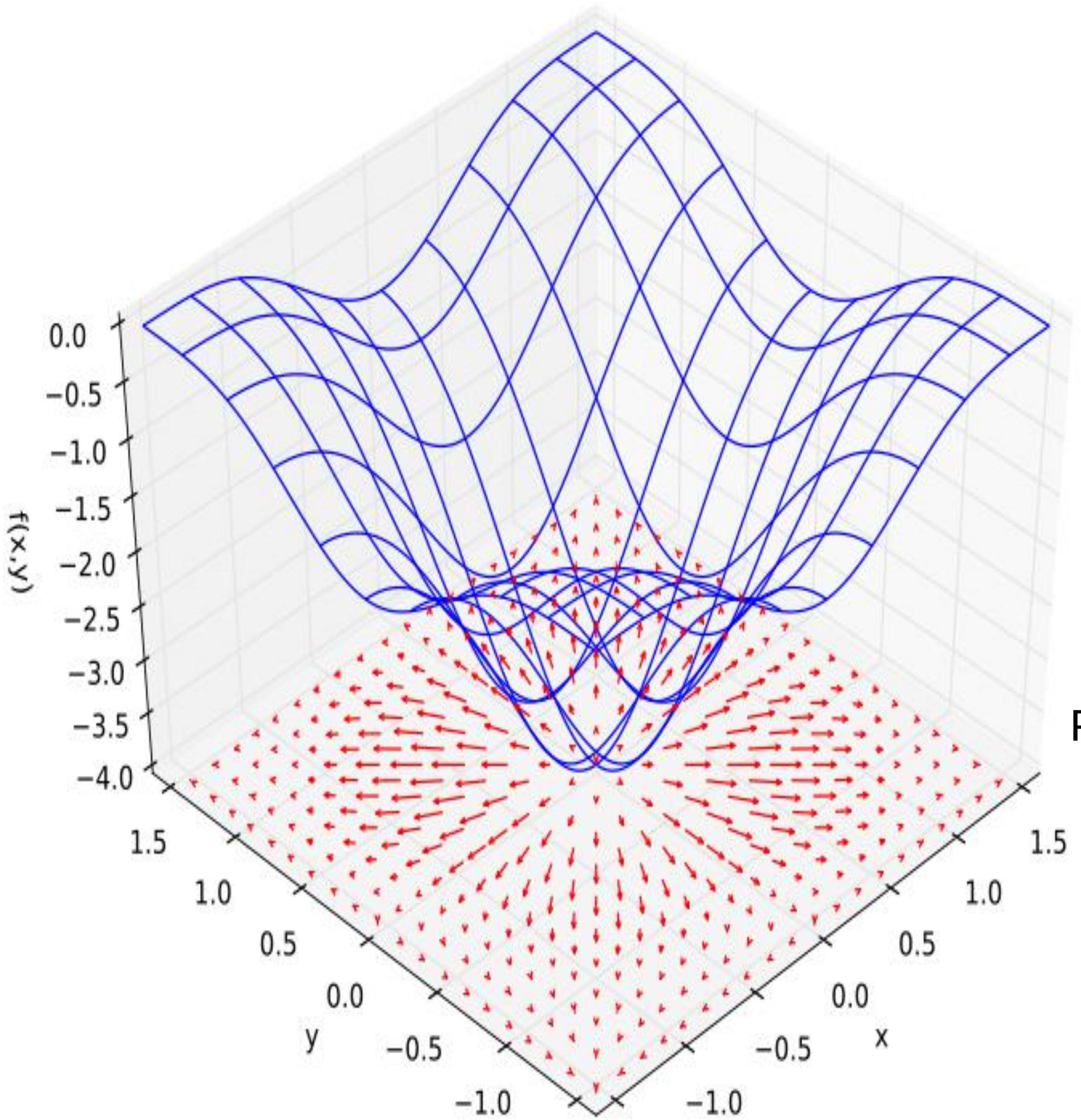
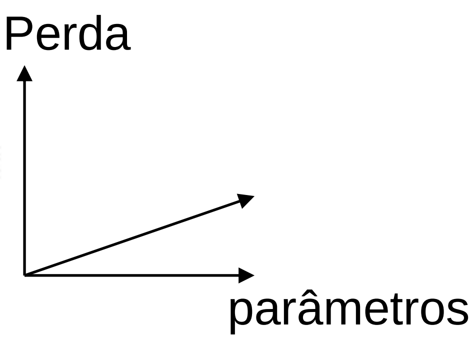
REDES NEURAIS ARTIFICIAIS

OTIMIZAÇÃO DA FUNÇÃO DE PERDA

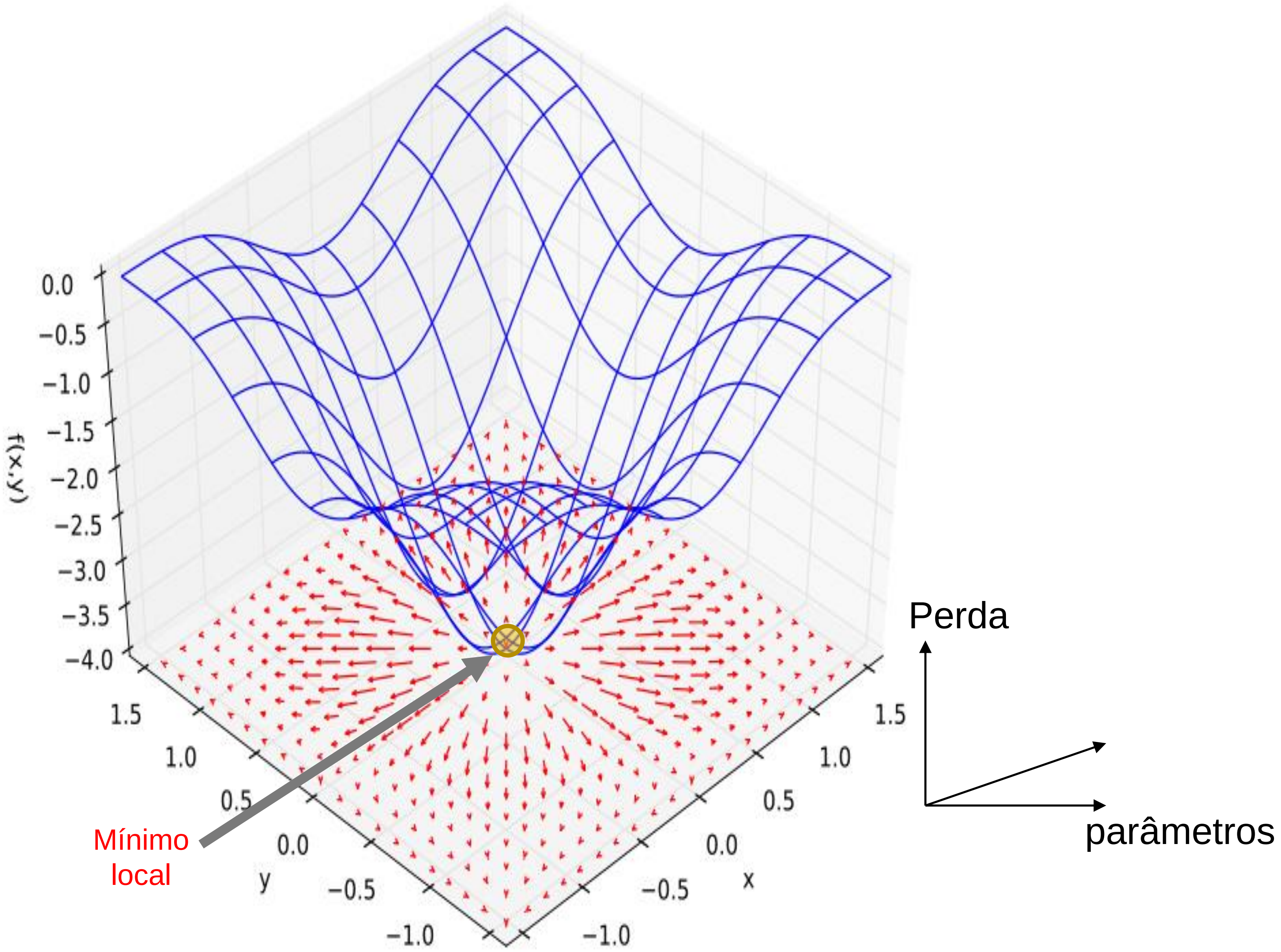
Zenilton K. G. Patrocínio Jr
zenilton@pucminas.br

Otimização da Função de Perda

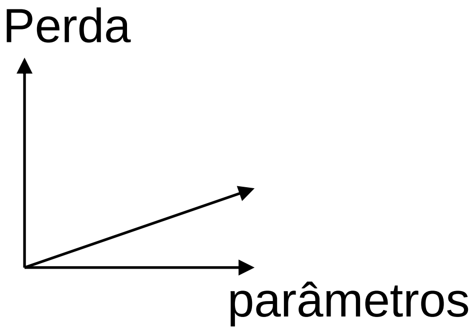




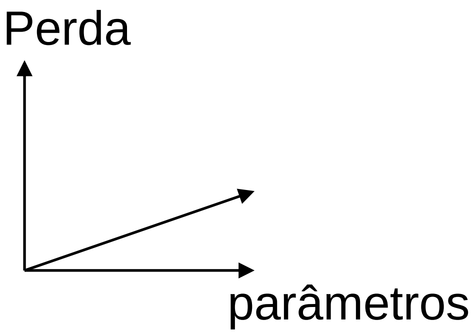
Otimização da Função de Perda



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
# ... (trunctated: 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%

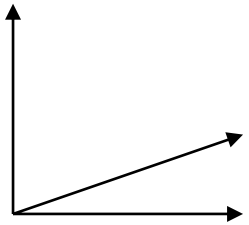
Método 1 – Busca Randômica

Toma-se vários passos aleatórios e mede-se a perda. Em seguira, 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

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

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

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

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

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

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

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

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;…]
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;…]
perda 1,25322

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;…]
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;…]
perda 1,25322

gradiente dW:

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

(1,25322 – 1,25347)/0,0001
= -2,5

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

?;
?;…]

Método 2 – Seguindo a Inclinação

W corrente:	**W + h** (2ª dim)**:**	**gradiente dW:**
[0,34;	[0,34;	[-2,5;
-1,11;	-1,11 + **0,0001**;	?;
0,78;	0,78;	?;
0,12;	0,12;	?;
0,55;	0,55;	?;
2,81;	2,81;	?;
-3,1;	-3,1;	?;
-1,5;	-1,5;	?;
0,33;…]	0,33;…]	?;…]
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 (2ª dim)**:**

[0,34;
-1,11 + **0,0001**;
0,78;
0,12;
0,55;
2,81;
-3,1;
-1,5;
0,33;…]
perda 1,25353

gradiente dW:

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

(1,25353 – 1,25347)/0,0001
= 0,6

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

?;…]

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 (3ª 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;…]
perda 1,25347

W + h (3ª 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;
0;
?;
?;…]

(1,25347 – 1,25347)/0,0001
= 0

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

Método 2 – Seguindo a Inclinação

Avaliação numérica do gradiente

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

```
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) # evalute 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

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

- **Aproximada**
- **Muito lenta**

```python
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) # evalute 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:

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

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

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

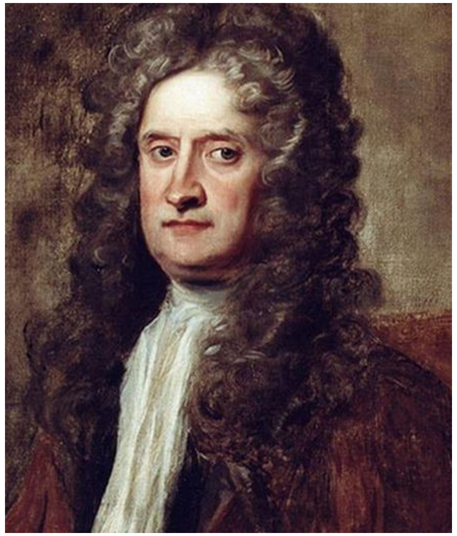
Deseja-se

$$\nabla_W L = \ldots$$

Solução

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

dW = ...
(alguma função dos dados e de W)

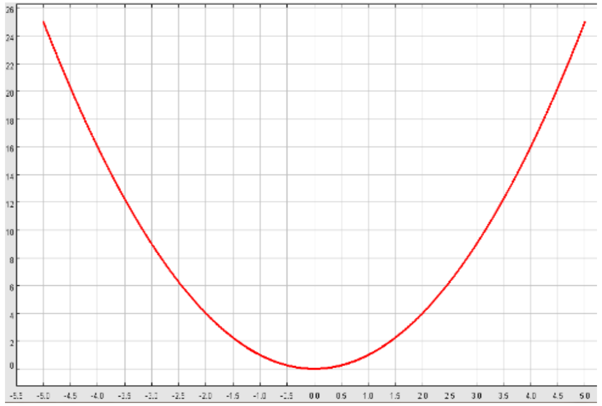
gradiente dW:

[-2,5;
0,6;
0;
0,2;
0,7;
-0,5;
1,1;
1,3;
-2,1;…]

Funções de Perda

Perda Quadrática

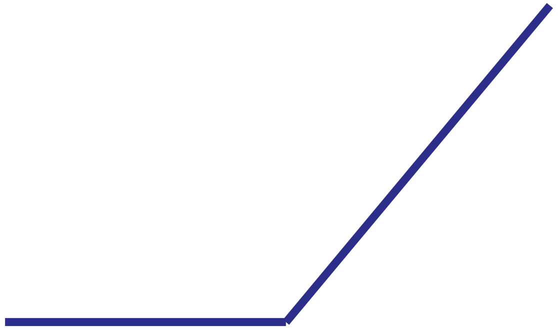
$$L = \left(y_i - f(x_i)\right)^2$$



Perda de Articulação

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

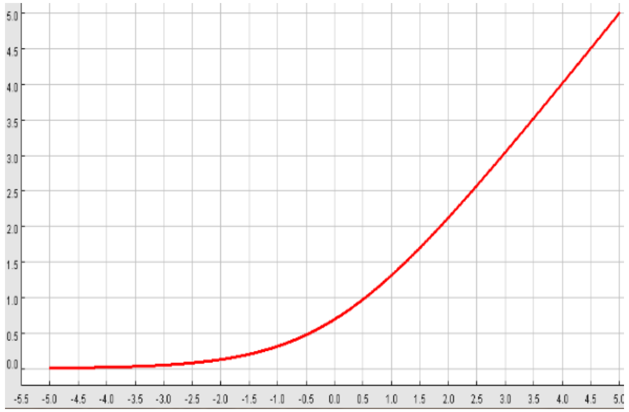
$$L = \max(0,1 - y_i f(x_i))$$



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.

$\Rightarrow$ Isto é chamado de **verificação de gradiente**