Python Básico com Numpy (exercício opcional)

José Ahirton Batista Lopes Filho - TIA 71760253

Bem-vindo ao seu primeiro exercício. Este exercício irá proporcionar uma breve introdução a Python. Mesmo que você já tenha utilizado Python antes, este exercício irá te ajudar a se familiarizar com funções que serão utilizadas ao longo deste curso.

Instruções:

- Você irá utilizar Python 3.
- Evite usar for-loops e while-loops, a menos que seja dito para você usá-los.
- Não modifique o (# FUNÇÃO de AVALIAÇÃO [nome da função]) comentada em algumas células. Seu trabalho não será avaliado se você modificar estes nomes. Cada célula contendo este tipo de comentário deve ter apenas uma função.
- Após codificar a sua função, execute a célula imediatamente abaixo da função para verificar se o resultado obtido está correto.

Após este exercício você deverá ser capaz de:

- Utilizar os notebooks do iPython.
- Utilizar funções do numpy e operações do numpy matrix/vector.
- Compreender os conceitos de "broadcasting"
- Ser capaz de vetorizar um código.

Muito bem, vamos começar!

Sobre notebooks iPython

Notebooks iPython são formas interativas de ambientes de codificação embutidos em uma página da web. Você irá utilizar notebooks iPython ao longo deste curso. Você precisa escrever código entre os comentários ### INICIE SEU CÓDIGO AQUI ### e ### TÉRMINO DO CÓDIGO ### . Após escrever seu código você pode executar a célula usando "SHIFT"+"ENTER" ou clicando no botão "Run Cell" (indicado pelo símbolo play) na barra superior do notebook.

Geralmente será especificado o número esperado de linhas de código "(≈ X linhas de código)" nos comentários para indicar a você o que é esperado. Este número é uma estimativa e não se preocupe se seu código estiver com mais ou menos linhas.

Exercício: coloque como valor para teste `"Bom dia mundo Python..."` na célula abaixo para imprimir "Bom dia mundo Python..." e execute as duas células abaixo.

In [1]:

```
### INICIE O SEU CÓDIGO AQUI ### (≈ 1 linha de código)
teste = "Bom dia mundo Python..."
### TÉRMINO DO CÓDIGO ###
```

In [2]:

```
print ("teste: " + teste)
```

```
teste: Bom dia mundo Python...
```

Saída esperada: teste: Bom dia mundo Python...

O que você precisa lembrar:

- Execute as células utilizando SHIFT+ENTER (ou "Run cell")
- Escreva os códigos na área designada utilizando apenas Python 3
- Nunca modifique o código fora da área designada

1 - Construindo funções básicas com numpy

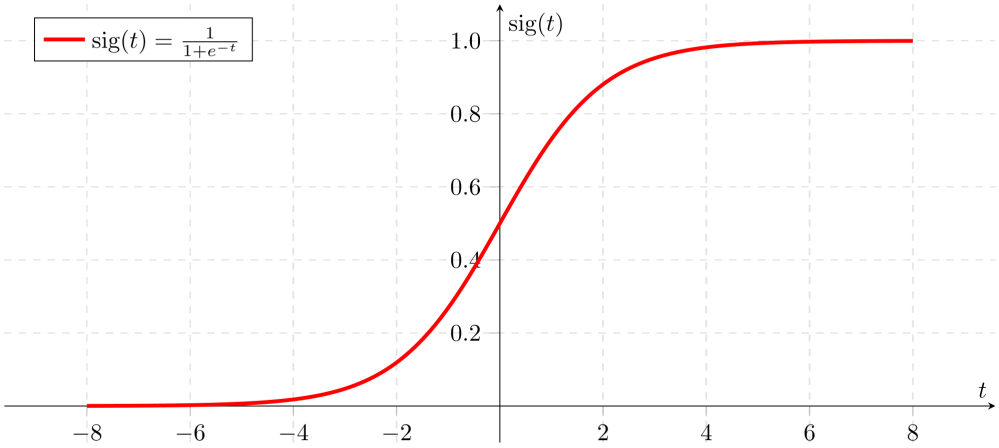
Numpy é o pacote principal do Python para computação científica. Ela é mantida por uma grande comunidade (www.numpy.org). Neste exercício você irá aprender várias funções chave do numpy, como np.exp, np.log, and np.reshape. Você precisará saber como utilizá-las em exercícios futuros.

1.1 - função sigmoid, np.exp()

Antes de utilizar np.exp(), você utilizará o math.exp() para implemetar a função sigmoid. Assim você irá compreender o porque a função np.exp() é preferível no lugar da math.exp().

Exercício: Construa uma função que retorne o sigmoid de um número real x. Utilize math.exp(x) para a função exponencial.

Lembre-se que: $sigmoid(x) = \frac{1}{1+e^{-x}}$ que também é conhecida como função logística. É uma função não linear utilizada não apenas em Aprendizado de Máquinas (Regressão logística), mas também em Deep Learning.



Para se referir a uma função que pertence a um determinado pacote você poderia fazer a chamada utilizando o nome do pacote nome_pacote.função(). Execute o código abaixo para ver um exemplo com math.exp().

In [3]:

```
# FUNÇÃO DE AVALIAÇÃO: sigmoid_basica

import math

def sigmoid_basica(x):
    """
    Determina o sigmoid de x.

    Argumentos:
    x -- Um escalar

    Retorna:
    s -- sigmoid(x)
    """

    ### INICIE O SEU CÓDIGO AQUI ### (≈ 1 linha de código)

    s= 1 / (1 + math.exp(-x))

    ### TÉRMINO DO CÓDIGO ###

    return s
```

In [4]:

```
sigmoid_basica(3)
```

Out[4]:

```
0.9525741268224334
```

Saída Esperada:

sigmoid_basica(3)	0.9525741268224334

Na verdade, raramente a biblioteca "math" é utilizada em deep learning porque as entradas das funções são números reais. Em Deep Learning as entradas são geralmente matrizes e vetores. Por isto a biblioteca numpy é mais útil.

In [5]:

```
### Uma das razões para se utilizar a biblioteca "numpy" no lugar da "math" em Deep
x = [1, 2, 3]
sigmoid_basica(x) # você verá que a execução deste comando irá dar um erro, isto oc
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-5-8b3f44223753> in <module>()
      1 ### Uma das razões para se utilizar a biblioteca "numpy" no lu
gar da "math" em Deep Learning ###
      2 x = [1, 2, 3]
----> 3 sigmoid_basica(x) # você verá que a execução deste comando irá
 dar um erro, isto ocorre porque x é um vetor.

<ipython-input-3-58371b2654a3> in sigmoid_basica(x)
     16     ### INICIE O SEU CÓDIGO AQUI ### (≈ 1 linha de código)
     17
---> 18     s= 1 / (1 + math.exp(-x))
     19
     20     ### TÉRMINO DO CÓDIGO ###

TypeError: bad operand type for unary -: 'list'
```

De fato, se $x = (x_1, x_2, \ldots, x_n)$é um vetor linha, então $np.exp(x)$ irá aplicar a função exponencial a cada elemento de x. A saída será: $np.exp(x) = (e^{x_1}, e^{x_2}, \ldots, e^{x_n})$

In [6]:

```
import numpy as np

# exemplo de np.exp
x = np.array([1, 2, 3])
print(np.exp(x)) # result is (exp(1), exp(2), exp(3))
```

```
[ 2.71828183  7.3890561  20.08553692]
```

Ainda, se x é um vetor, então uma operação em Python como $s = x + 3$ ou $s = \frac{1}{x}$ irá retornar s como um vetor do mesmo tamanho de x.

In [7]:

```
# exemplo de operação com vetor
x = np.array([1, 2, 3])
print (x + 3)
```

```
[4 5 6]
```

A qualquer momento que você precisar de mais informação sobre a função numpy.exp(), de uma olhada em documentação oficial (https://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.exp.html).

Você pode ainda criar uma nova célula no seu notebook e escrever `np.exp?` (por exemplo) para acessar a documentação rapidamente.

Exercício: Implemente a função sigmoid utilizando numpy.

Instruções: x pode ser tanto um número real, um vetor ou uma matriz. A estrutura de dados utilizada em numpy para representar estas formas (vetores, matrizes, ...) são chamadas de arrays em numpy. Por enquanto você não precisa saber mais sobre isto.

$$\text{Para } x \in \mathbb{R}^n,\ sigmoid(x) = sigmoid\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-x_1}} \\ \frac{1}{1+e^{-x_2}} \\ \dots \\ \frac{1}{1+e^{-x_n}} \end{pmatrix} \tag{1}$$

In [8]:

```
# FUNÇÃO DE AVALIAÇÃO: sigmoid

import numpy as np # isto quer dizer que você pode acessar uma função numpy usando n

def sigmoid(x):
    """
    Compute the sigmoid of x

    Arguments:
    x -- A scalar or numpy array of any size

    Return:
    s -- sigmoid(x)
    """

    ### INICIE O SEU CÓDIGO AQUI  ### (≈ 1 linha de código)

    s = 1/(1+np.exp(-x))

    ### TÉRMINO DO CÓDIGO ###

    return s
```

In [9]:

```
x = np.array([1, 2, 3])
sigmoid(x)
```

Out[9]:

```
array([0.73105858, 0.88079708, 0.95257413])
```

Saída Esperada:

sigmoid([1,2,3])	array([0.73105858, 0.88079708, 0.95257413])

1.2 - Gradiente da Sigmoid

Como foi visto na aula, é preciso determinar gradientes para poder otimizar as funções de perda quando utilizamos a backpropagation. Vamos codificar a nossa primeira função gradiente.

Exercício: Implemente a função derivativa_sigmoid() para computar o gradiente da função sigmoid com relação a sua entrada x. A formula é:

$$derivativa\_sigmoid(x) = \sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2}$$

Normalmente esta função é codificada em duas etapas:

1. Ajusta s para ser o sigmoid de x. Você pode utilizar a sua função sigmoid(x) se quiser.
2. Compute $\sigma'(x) = s(1-s)$

In [10]:

```
# FUNÇÃO DE AVALIAÇÃO: derivativa_sigmoid

def derivativa_sigmoid(x):
    """
    Determina o gradiente (também chamado de inclinação ou derivativa) da função sig
    Você pode armazenar a saída da função sigmoid em variáveis e então utilizá-las p

    Argumentos:
    x -- Um escalar ou um array numpy

    Retorna:
    ds -- O gradiente calculado.
    """

    ### INICIE O SEU CÓDIGO AQUI ### (≈ 2 linhas de código)

    sigmoid(x)
    ds = sigmoid(x) * (1 - sigmoid(x))

    ### TÉRMINO DO CÓDIGO ###

    return ds
```

In [11]:

```
x = np.array([1, 2, 3])
print ("derivativa_sigmoid(x) = " + str(derivativa_sigmoid(x)))
```

```
derivativa_sigmoid(x) = [0.19661193 0.10499359 0.04517666]
```

Saída Esperada:

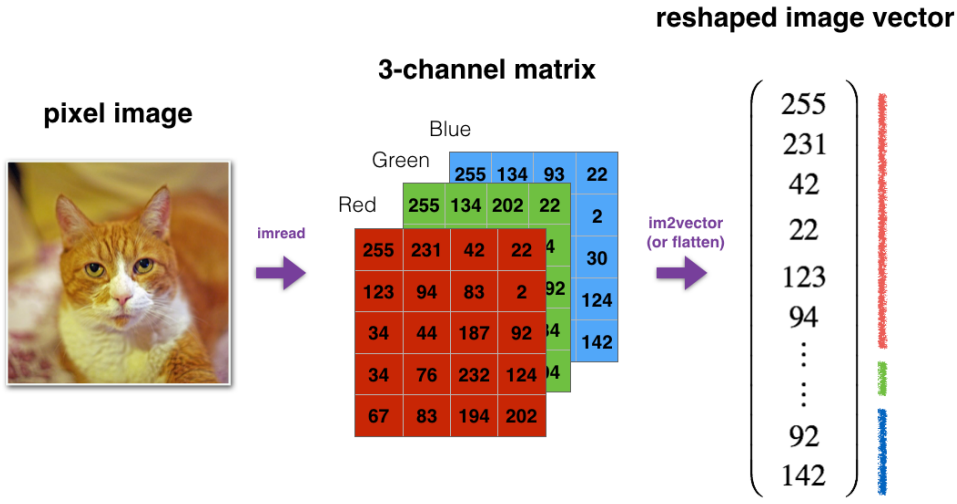
derivativa_sigmoid([1,2,3])	[0.19661193 0.10499359 0.04517666]

1.3 - Reformatando arrays

Duas funções comuns em numpy utilizadas em deep learning são np.shape (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html) e np.reshape() (https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html).

- X.shape é utilizado para obter a forma (dimensões) de uma matriz/vetor X.
- X.reshape(...) é utilizado para reformatar X em alguma outra dimensão.

Por exemplo, em ciência da computação, uma imagem é representada por um array em 3D no formato $(comprimento, altura, profundidade = 3)$. Porém, quando você lê uma imagem como uma entrada de um algoritmo você a converte pra um vetor no formato $(comprimento * altura * 3, 1)$. Em outras palavras, você "desenrola", ou reformata, o array em 3D em um vetor em 1D.



Exercício: Implemente `imagemParaVetor()` que recebe uma entrada no formato (comprimento, altura, 3) e retorna um vetor no formato (comprimento*altura*3, 1). Por exemplo, se você quiser reformatar um array v do formato (a, b, c) para o formato (a*b,c) você deve executar o comando:

```
v = v.reshape((v.shape[0]*v.shape[1], v.shape[2])) # v.shape[0] = a ; v.shape[1] = b ; v.shape[2] = c
```

- Por favor não codifique as dimensões de uma imagem como uma constante. Em vez disso, olhe as dimensões que você precisa com o comando `image.shape[0]`, etc.

In [12]:

```
# FUNÇÃO DE AVALIAÇÃO: imagemParaVetor
def imagemParaVetor(imagem):
    """
    Argumento:
    imagem -- um array numpy array no formato (comprimento, altura, profundidade)

    Retorna:
    v -- um vetor no formato (comprimento*altura*profundidade, 1)
    """

    ### INICIE O SEU CÓDIGO AQUI ### (≈ 1 linha de código)

    v = imagem.reshape((imagem.shape[0] * imagem.shape[1] * imagem.shape[2],1))

    ### TÉRMINO DO CÓDIGO ###

    return v
```

In [13]:

```
# This is a 3 by 3 by 2 array, typically images will be (num_px_x, num_px_y,3) where
imagem = np.array([[[ 0.67826139,  0.29380381],
        [ 0.90714982,  0.52835647],
        [ 0.4215251 ,  0.45017551]],

       [[ 0.92814219,  0.96677647],
        [ 0.85304703,  0.52351845],
        [ 0.19981397,  0.27417313]],

       [[ 0.60659855,  0.00533165],
        [ 0.10820313,  0.49978937],
        [ 0.34144279,  0.94630077]]])

print ("imagemParaVetor(imagem) = " + str(imagemParaVetor(imagem)))
```

```
imagemParaVetor(imagem) = [[0.67826139]
 [0.29380381]
 [0.90714982]
 [0.52835647]
 [0.4215251 ]
 [0.45017551]
 [0.92814219]
 [0.96677647]
 [0.85304703]
 [0.52351845]
 [0.19981397]
 [0.27417313]
 [0.60659855]
 [0.00533165]
 [0.10820313]
 [0.49978937]
 [0.34144279]
 [0.94630077]]
```

Saída Esperada:

imagemParaVetor(image)	[[0.67826139] [0.29380381] [0.90714982] [0.52835647] [0.4215251] [0.45017551] [0.92814219] [0.96677647] [0.85304703] [0.52351845] [0.19981397] [0.27417313] [0.60659855] [0.00533165] [0.10820313] [0.49978937] [0.34144279] [0.94630077]]

1.4 - Normalizando linhas

Uma outra técnica comum utilizada em Aprendizado de Máquinas e Deep Learning é a técnica de normalização de dados. Ela normalmente nos leva a um melhor desempenho porque o gradiente descendente converge mais rápido após a normalização. Aqui, o termo normalização significa trocar x $\frac{x}{\|x\|}$ (dividindo cada linha do vetor de x pela sua norm).

Por exemplo, se

$$x = \begin{bmatrix} 0 & 3 & 4 \\ 2 & 6 & 4 \end{bmatrix} \tag{3}$$

então

$$\|x\| = np.linalg.norm(x, axis = 1, keepdims = True) = \begin{bmatrix} 5 \\ \sqrt{56} \end{bmatrix} \tag{4}$$

e

$$x\_normalized = \frac{x}{\|x\|} = \begin{bmatrix} 0 & \frac{3}{5} & \frac{4}{5} \\ \frac{2}{\sqrt{56}} & \frac{6}{\sqrt{56}} & \frac{4}{\sqrt{56}} \end{bmatrix} \tag{5}$$

Note que você pode dividir matrizes de tamanhos diferentes e funciona sem problemas: isto é chamado de broadcasting e você vai aprender sobre isto na parte 5.

Exercício: Implemente normalizaLinhas() para normalizar as linhas de uma matriz. Após aplicar esta função para uma matriz de entrada x, cada linha de x será um vetor de comprimento unitário (ou seja comprimento 1).

In [14]:

```
# FUNÇÃO DE AVALIAÇÃO: normalizaLinhas

def normalizaLinhas(x):
    """
    Implemente a função que normaliza cada linha de uma matriz x (para ter comprimer

    Argumento:
    x -- Uma matriz numpy matrix no formato (n, m)

    Retorna:
    x -- A matriz numpy normalizada (por linha). Você tem permissão para modificar x
    """

    ### INICIE O SEU CÓDIGO AQUI  ### (≈ 2 linhas de código)
    # Compute x_norm como a norm 2 de x. Use np.linalg.norm(..., ord = 2, axis = ..

    x_norm = np.linalg.norm(x,ord = 2,axis = 1,keepdims = True)

    # Divida x pela sua norm.

    x = x / x_norm

    ### TÉRMINO DO CÓDIGO ###

    return x
```

In [15]:

```
x = np.array([
    [0, 3, 4],
    [1, 6, 4]])
print("normalizaLinhas(x) = " + str(normalizaLinhas(x)))
```

```
normalizaLinhas(x) = [[0.         0.6        0.8       ]
 [0.13736056 0.82416338 0.54944226]]
```

Saída Esperada:

normalizaLinhas(x)	[[0. 0.6 0.8] [0.13736056 0.82416338 0.54944226]]

Nota: Em normalizaLinhas(), você pode tentar imprimir os formatos de x_norm e x, e então executar a avaliação. Você verá que eles possuem formas diferentes. Isto é normal dado que x_norm pega a norm de cada linha de x. Logo x_norm possui o mesmo número de linhas mas apenas 1 coluna. Então, como isto

funciona quando você divide x pela x_norm? Isto é chamado de broadcasting e nós vamos falar sobre isto agora!

1.5 - Broadcasting e a função softmax

Um conceito importante para se entender em numpy é o "broadcasting". Ele é muito útil n desempenho de operações matemáticas entre arrays de formatos diferentes. Para detalhes mais completos sobre broadcasting leia no site oficial documentação sobre broadcasting (http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html).

Exercício: Implemente a função softmax utilizando numpy. Você pode pensar na função softmax como uma função de normalização utilizada quando o algoritmo precisa classificar entre duas ou mais classes. Você irá aprender mais sobre softmax ainda neste curso mas em módulos mais avançados.

Instruções

- para $x \in \mathbb{R}^{1\times n}$, $softmax(x) = softmax(\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}) = \begin{bmatrix} \frac{e^{x_1}}{\sum_j e^{x_j}} & \frac{e^{x_2}}{\sum_j e^{x_j}} & \cdots \end{bmatrix}$
- para uma matriz $x \in \mathbb{R}^{m\times n}$, x_{ij} mapeia para o elemento na linha i e coluna j de x, portanto tem-se:

$$softmax(x) = softmax\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix} = \begin{bmatrix} \frac{e^{x_{11}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{12}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{13}}}{\sum_j e^{x_{1j}}} & \cdots & \frac{e^{x_{1n}}}{\sum_j e^{x_{1j}}} \\ \frac{e^{x_{21}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{22}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{23}}}{\sum_j e^{x_{2j}}} & \cdots & \frac{e^{x_{2n}}}{\sum_j e^{x_{2j}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{e^{x_{m1}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m2}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m3}}}{\sum_j e^{x_{mj}}} & \cdots & \frac{e^{x_{mn}}}{\sum_j e^{x_{mj}}} \end{bmatrix} =$$

In [16]:

```
# FUNÇÃO DE AVALIAÇÃO: softmax

def softmax(x):
    """Calcula o valor da função softmax para cada linha da entrada x.

    Seu código deve funcionar para um vetor linha e também para matrizes no formato

    Argumento:
    x -- Uma matriz numpy no formato (n,m)

    Retorna:
    s -- Uma matriz numpy igual ao softmax de x, no formato (n,m)
    """

    ### INICIE O SEU CÓDIGO AQUI  ### (≈ 3 linhas de código)
    # Aplique exp() em cada elemento de x. Use np.exp(...).

    x_exp = np.exp(x)

    # Crie um vetor x_soma que some cada linha de x_exp. Use np.sum(..., axis = 1, k

    x_soma = np.sum(x_exp,axis = 1,keepdims = True)

    # Compute softmax(x) dividindo x_exp por x_soma. Ele deve automaticamente utili

    s = x_exp / x_soma


    ### TÉRMINO DO CÓDIGO ###

    return s
```

In [17]:

```
x = np.array([
    [9, 2, 5, 0, 0],
    [7, 5, 0, 0 ,0]])
print("softmax(x) = " + str(softmax(x)))
```

```
softmax(x) = [[9.80897665e-01 8.94462891e-04 1.79657674e-02 1.21052389
e-04
  1.21052389e-04]
 [8.78679856e-01 1.18916387e-01 8.01252314e-04 8.01252314e-04
  8.01252314e-04]]
```

Saída Esperada:

softmax(x)	[[9.80897665e-01 8.94462891e-04 1.79657674e-02 1.21052389e-04 1.21052389e-04] [8.78679856e-01 1.18916387e-01 8.01252314e-04 8.01252314e-04 8.01252314e-04]]

Nota:

- Se você imprimir o formato de x_exp, x_soma e s acima e executar a célula novamente, você verá que x_soma está no formato (2,1) enquanto que x_exp e s estão no formato (2,5). **x_exp/x_soma** só funciona

devido ao broadcasting do Python.

Parabéns! Agora você já consegue entender como a biblioteca numpy do Python implementa uma série de funções uteis em algoritmos de deep learning.

O que você precisa lembrar:

- np.exp(x) funciona para qualquer array np x e aplica a função exponencial para todos os elementos de x.
- a função sigmoid e seu gradiente.
- imagemParaVetor é bastante utilizada em deep learning.
- np.reshape é bastante utilizada. No futuro, você verá que manter a dimensão de sua matriz/vetor de forma correta irá ajudar a eliminar uma série de problemas de debug no código.
- numpy possui uma série de funções eficientes em sua biblioteca.
- broadcasting é bastante útil.

2) Vectorização

Em deep learning, você irá lidar com conjuntos de dados muito grandes. Logo, uma função computacional que não seja ótima pode se tornar um grande gargalo no seu algoritmo e pode tornar seu modelo um sistema muito lento. Para ter certeza que seu código é computacionalmente eficiente você deverá utilizar vetorização. Por exemplo, tente mostrar a diferença entre as seguintes implementações dos produtos dot/outer/elementwise.

In [18]:

```
import time

x1 = [9, 2, 5, 0, 0, 7, 5, 0, 0, 0, 9, 2, 5, 0, 0]
x2 = [9, 2, 2, 9, 0, 9, 2, 5, 0, 0, 9, 2, 5, 0, 0]

### IMPLEMENTAÇÃO CLÁSSICA DO PRODUTO DOT ###
tic = time.process_time()
dot = 0
for i in range(len(x1)):
    dot+= x1[i]*x2[i]
toc = time.process_time()
print ("dot = " + str(dot) + "\n ----- Tempo de Processamento = " + str(1000*(toc -

### IMPLEMENTAÇÃO CLÁSSICA DO PRODUTO OUTER ###
tic = time.process_time()
outer = np.zeros((len(x1),len(x2))) # criamos uma matriz de zeros de tamanho len(x1)
for i in range(len(x1)):
    for j in range(len(x2)):
        outer[i,j] = x1[i]*x2[j]
toc = time.process_time()
print ("outer = " + str(outer) + "\n ----- Tempo de Processamento = " + str(1000*(to

### IMPLEMENTAÇÃO CLÁSSICA DO PRODUTO ELEMENTO A ELEMENTO ###
tic = time.process_time()
mul = np.zeros(len(x1))
for i in range(len(x1)):
    mul[i] = x1[i]*x2[i]
toc = time.process_time()
print ("multiplicação elemento a elemento = " + str(mul) + "\n ----- Tempo de Proces

### IMPLEMENTAÇÃO GERAL DO PRODUTO DOT ###
W = np.random.rand(3,len(x1)) # array numpy com Random 3*len(x1)
tic = time.process_time()
gdot = np.zeros(W.shape[0])
for i in range(W.shape[0]):
    for j in range(len(x1)):
        gdot[i] += W[i,j]*x1[j]
toc = time.process_time()
print ("gdot = " + str(gdot) + "\n ----- Tempo de Processamento = " + str(1000*(toc
```

```
dot = 278
 ----- Tempo de Processamento = 0.0860000000000305ms
outer = [[ 81.  18.  18.  81.   0.  81.  18.  45.   0.   0.  81.  18.
45.   0.
    0.]
 [ 18.   4.   4.  18.   0.  18.   4.  10.   0.   0.  18.   4.  10.
0.
    0.]
 [ 45.  10.  10.  45.   0.  45.  10.  25.   0.   0.  45.  10.  25.
0.
    0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.
    0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.
    0.]
 [ 63.  14.  14.  63.   0.  63.  14.  35.   0.   0.  63.  14.  35.
```

```
0.
    0.]
 [ 45.  10.  10.  45.   0.  45.  10.  25.   0.   0.  45.  10.  25.
0.
    0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.
    0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.
    0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.
    0.]
 [ 81.  18.  18.  81.   0.  81.  18.  45.   0.   0.  81.  18.  45.
0.
    0.]
 [ 18.   4.   4.  18.   0.  18.   4.  10.   0.   0.  18.   4.  10.
0.
    0.]
 [ 45.  10.  10.  45.   0.  45.  10.  25.   0.   0.  45.  10.  25.
0.
    0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.
    0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.
    0.]]
 ----- Tempo de Processamento = 0.6720000000000059ms
multiplicação elemento a elemento = [ 81.   4.  10.   0.   0.  63.  1
0.   0.   0.   0.  81.   4.  25.   0.   0.]
 ----- Tempo de Processamento = 0.25300000000005873ms
gdot = [ 21.9738098   17.81619891  30.66566753]
 ----- Tempo de Processamento = 0.3359999999996699ms
```

In [19]:

```
x1 = [9, 2, 5, 0, 0, 7, 5, 0, 0, 0, 9, 2, 5, 0, 0]
x2 = [9, 2, 2, 9, 0, 9, 2, 5, 0, 0, 9, 2, 5, 0, 0]

### FORMA VETORIZADA DO PRODUTO DOT DE VETORES ###
tic = time.process_time()
dot = np.dot(x1,x2)
toc = time.process_time()
print ("dot = " + str(dot) + "\n ----- Tempo de Processamento = " + str(1000*(toc -

### FORMA VETORIZADA DO PRODUTO OUTER ###
tic = time.process_time()
outer = np.outer(x1,x2)
toc = time.process_time()
print ("outer = " + str(outer) + "\n ----- Tempo de Processamento = " + str(1000*(to

### FORMA VETORIZADA DE MULTIPLICAÇÃO DE ELEMENTO A ELEMENTO ###
tic = time.process_time()
mul = np.multiply(x1,x2)
toc = time.process_time()
print ("multiplicação elemento a elemento = " + str(mul) + "\n ----- Tempo de Proces

### FORMA VETORIZADA GERAL DO PRODUTO DOT ###
tic = time.process_time()
dot = np.dot(W,x1)
toc = time.process_time()
print ("gdot = " + str(dot) + "\n ----- Tempo de Processamento = " + str(1000*(toc -
```

```
dot = 278
 ----- Tempo de Processamento = 0.10399999999988196ms
outer = [[81 18 18 81  0 81 18 45  0  0 81 18 45  0  0]
 [18  4  4 18  0 18  4 10  0  0 18  4 10  0  0]
 [45 10 10 45  0 45 10 25  0  0 45 10 25  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [63 14 14 63  0 63 14 35  0  0 63 14 35  0  0]
 [45 10 10 45  0 45 10 25  0  0 45 10 25  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [81 18 18 81  0 81 18 45  0  0 81 18 45  0  0]
 [18  4  4 18  0 18  4 10  0  0 18  4 10  0  0]
 [45 10 10 45  0 45 10 25  0  0 45 10 25  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
 ----- Tempo de Processamento = 0.21800000000005149ms
multiplicação elemento a elemento = [81  4 10  0  0 63 10  0  0  0 81
4 25  0  0]
 ----- Tempo de Processamento = 0.136000000000136ms
gdot = [ 21.9738098   17.81619891  30.66566753]
 ----- Tempo de Processamento = 69.15800000000006ms
```

Como você deve ter notado, a implementação vetorizada é mais limpa e também mais eficiente (embora para o exemplo acima os tempos tenham sido semelhantes) Para vetores ou matrizes maiores a diferença em tempo de processamento é bem maior.

Nota: veja que `np.dot()` executa uma multiplicação matriz-matriz ou matriz-vetor. Esta é a diferença de `np.multiply()` e o operador `*` (que é equivalente a `.*` em Matlab/Octave), e que executa uma multiplicação elemento a elemento.

2.1 Implementando as funções de perda L1 e L2

Exercício: Implemente uma versão vetorizada, utilizando numpy, da função de perda L1. A função abs(x) (valor absoluto de x) pode ser útil.

Lembre-se:

- A função de perda é utilizada para avaliar o desempenho de um modelo. Quanto maior o valor da perda maior a diferença entre a previsão ($\hat{y}$) com relação ao valor real (y). Em deep learning, utilizam-se algoritmos de otimização como o Gradiente Descendente para treinar o modelo e minimizar o custo.
- A perda L1 é definida como:

$$L_1(\hat{y}, y) = \sum_{i=0}^{m} |y^{(i)} - \hat{y}^{(i)}| \tag{6}$$

In [20]:

```
# FUNÇÃO DE AVALIAÇÃO: L1

def L1(yhat, y):
    """
    Argumentos:
    yhat -- vetor de tamanho m (valores estimados)
    y -- vetor de tamanho m (valores reais)

    Retorna:
    perda -- o valor da função de perda L1 definido acima
    """

    ### INICIE O SEU CÓDIGO AQUI  ### (≈ 1 linha de código)

    perda = np.sum(np.abs(yhat-y),axis = 0)


    ### TÉRMINO DO CÓDIGO  ###

    return perda
```

In [21]:

```
yhat = np.array([.9, 0.2, 0.1, .4, .9])
y = np.array([1, 0, 0, 1, 1])
print("L1 = " + str(L1(yhat,y)))
```

```
L1 = 1.1
```

Saída Esperada:

L1	1.1

Exercício: Implemente uma versão vetorizada, utilizando numpy, da função de perda L2. Existem diversas formas de se implementar a função de perda L2, neste caso a função np.dot() pode ser útil. Lembre-se que, se

$x = [x_1, x_2, \ldots, x_n]$ então `np.dot(x,x)` = $\sum_{j=0}^{n} x_j^2$.

- A função de perda L2 é definida como

$$L_2(\hat{y}, y) = \sum_{i=0}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \tag{7}$$

In [22]:

```
# FUNÇÃO DE AVALIAÇÃO: L2

def L2(yhat, y):
    """
    Argumentos:
    yhat -- vetor de tamanho m (valores estimados)
    y -- vetor de tamanho m (valores reais)

    Retorna:
    perda -- o valor da função de perda L2 definido acima
    """

    ### INICIE O SEU CÓDIGO AQUI  ### (≈ 1 linha de código)

    perda = np.dot(np.abs(yhat-y),np.abs(yhat-y))

    ### TÉRMINO DO CÓDIGO ###

    return perda
```

In [23]:

```
yhat = np.array([.9, 0.2, 0.1, .4, .9])
y = np.array([1, 0, 0, 1, 1])
print("L2 = " + str(L2(yhat,y)))
```

```
L2 = 0.43
```

Saída Esperada:

L2	0.43

Parabéns, você completou este trabalho. Esperamos que este aquecimento venha a ajudá-lo em trabalhos futuros, que devem ser mais interessantes e mais desafiadores!

Lembre-se:

- Vetorização é muito importante no desempenho de algoritmos de deep learning. Ela melhora a eficiência e deixa o código mais limpo.
- Foram revisadas as funções de perda L1 e L2.
- Você se familiarizou com muitas funções numpy, como np.sum, np.dot, np.multiply, np.maximum, etc...