**REPORT EXERCISE COURSERA 1**

**Python Basics with Numpy (Optional [assignment](https://www.coursera.org/learn/neural-networks-deep-learning/programming/isoAV/python-basics-with-numpy/lab?path=%2Fnotebooks%2Frelease%2FW2A1%2FPython_Basics_with_Numpy.ipynb))**

1. **Exercise and knowledge:**
2. **Building basic functions with numpy:**
   1. ***Sigmoid function:*** np.exp () or (math.exp() )

Sigmoid1 = 1 / (1 + math.exp(-x)) or

Sigmoid2 = 1 / (1 + np.exp(-x))

Note:

* Dùng để tính xác suất giá trị đầu ra nằm trong khoảng từ 0 tới 1 thường dùng với binary classification
* Calculation Sigmoid function we can use library” Numpy” or “math”.
* But in deep learning we rarely use “math” library because the inputs of the functions are matrices and vectors so we need use “Numpy” library. If the inputs of the functions are real numbers, we can use math or Numpy to calculation Sigmoid functions.
* ( bởi vì dùng math khi tính hàm mũ (exponential) exp() thì phải dùng với for để lặp qua từng phần tử rùi mới tính được. Còn khi dùng numpy thì cho phép thực hiện phép toán trên toàn bộ mảng mà không cần dùng vòng lặp)
* Example: Sigmoid1(1) = 0.731058

Sigmoid2(np.array[1, 2, 3]) = [2.71828, 738905, 20.08553]

* 1. ***Sigmoid Gradient:***



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

Sigmoid\_derivative = Sigmoid \* (1 - Sigmoid)

Note:

* Để tối ưu hàm mất mát ta cần truyển ngược( đạo hàm) để tối ưu. Để update các thông số.
  1. ***Reshaping arrays:***

np.shape(): use to get the shape (dimension) of a matrix/vector X

np.reshape (): use to reshape X into some other dimension

Ex:

X = np.array([1, 2, 3, 4, 5, 6]) --> x.shape --> (6, 0) mảng 1 chiều

X\_reshape = x.reshape((2, 3)) --> x\_reshape.shape ---> [[1, 2, 3], [4, 5, 6]] ---> (2, 3) mảng 2 chiều

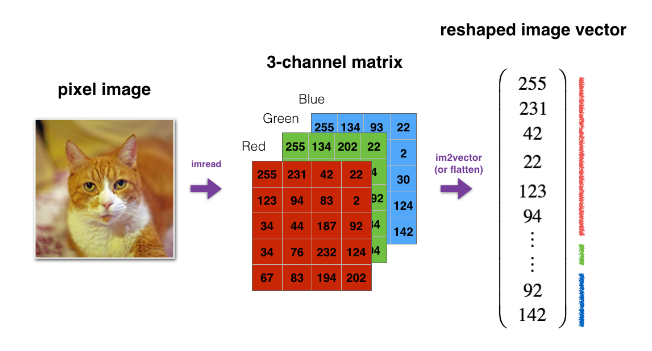
X\_reshape\_3D = x.reshape((2, 3, 3)) ---> [[[1, 2, 3]] [[4, 5, 6]]]

V = image.reshape(-1, 1) : number -1 : tự động tính toán số lượng phần tử trong chiều

Ex: (100, 200, 3) ---> reshape(-1, 1) = (60000, 1)

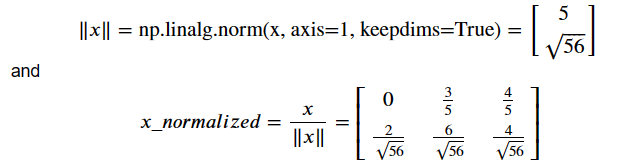
(10, 10) ---> reshape(-1, 1) = (100, 1)

Or v = image.reshape((image.shape[0] \* image.shape[1] \* image.shape[2], 1))



Note:

* In Computer vision, an image is represented by 3D array of shape( length, height, depth = 3)
* But in deep learning, when we read an image as the input, we need convert it to a vector of shape (length \* height \* 3, 1) (reshape 3D array into a 1D vector)
  1. ***Normalizing rows:***



Chuẩn hóa dữ liệu giúp hiệu suất tốt hơn vì độ dốc giảm dần, hội tụ nhanh hơn sau khi chuẩn hóa

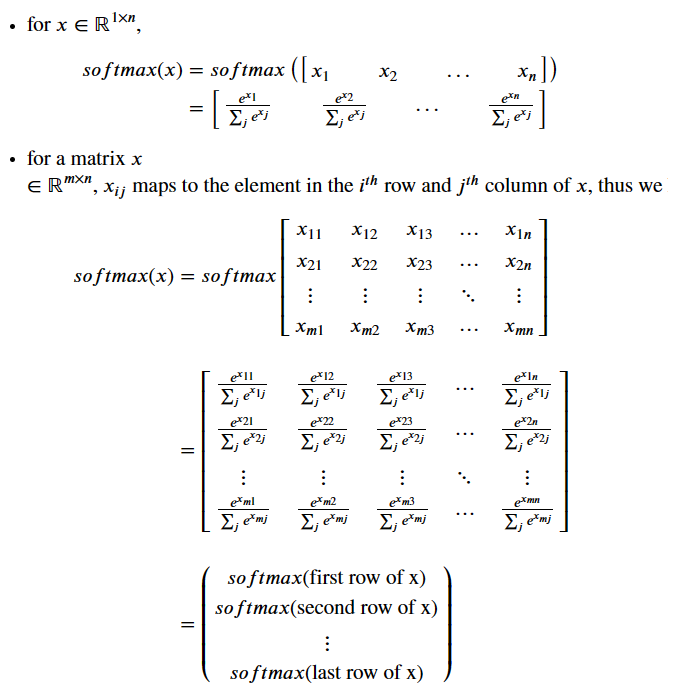
x\_norm = np.linalg.norm(x, ord = 2, axis = 1, keepdims = True)

# ord là giá trị norm mình chọn này là 2 là norm 2, axis chỉ định tính norm theo hàng output là norm của mỗi hàng(cột = 0), keepdims: giữ nguyên kích thước ma trận cho dù có bị giảm chiều hay ko

x = x / x\_norm

**NOTE:** ma trận x (n, m) sau khi tính x-norm thì ma trận còn dạng (n, 1) vì tính norm theo hàng vd: x = [[1, 2, 3], [4, 5, 6]] ---> x-norm = [[3.123], [4.567]] và nếu vậy thì chia ko được nhưng numpy sẽ nhân mảng này để phù hợp với kích thước để có thể chia (**gọi là Broadcasting** ) x\_norm\_ = [[3.123, 3.123, 3.123], [4.567, 4.567, 4.567]]

* 1. ***Softmax:***



x\_exp = np.exp(x)

x\_sum = np.sum(x\_exp, axis = 1, keepdims = True)

s = x\_exp / x\_sum

Note: if x\_exp and x\_sum.shape so x\_sum.shape() have (2, 1) but x\_exp.shape() have (2, 5)

1. ***Vectorization:*** vector hóa: giúp hiệu quả về mặt tính toán. Thông qua một số thuật toán như : dot/outer/ elementwise

|  |  |  |  |
| --- | --- | --- | --- |
| **Non-vectorized** | | **Vectorized** | |
|  |  |  |  |
| **Dot product** | Dùng vòng lập để tính từng phần tử của vector. Thời gian chạy không nhanh | **Vectorized dot product** | Dùng hàm np.dot() tính tích vô hướng 2 vector  Thời gian thấp |
| **Outer product** | Dùng vòng lặp lòng nhau tính từng phần tử | **Vectorized outer product** | Dùng hàm np.outer() tính tích ngoài của hai vector |
| **Elementwise mulltiplication** | Sử dụng vòng lặp để tính toán các phần tử của vector và nhân chúng lại | **Vectorized Elementwise mulltiplication** | Dùng hàm np.multiple() tính toán nhân từng phần tử của 2 vector |
| **General dot product** | Sử dụng 2 vòng lặp lồng nhau để tính toán từng phần tử của vector kết quả | **Vectorized general dot product** | Dùng hàm np.dot() tính tích vô hướng của ma trận và vector |
| * Dùng vòng lặp để tính toán nhưng thời gian chạy chậm do phải duyệt và tính toán qua từng phần tử trong vector nên thời gian chậm hơn. * Nếu dữ liệu nhỏ ta có thể thực hiện vòng lặp để thực hiện việc tính toán | | * Dùng hàm của thư viện Numpy nên thời gian chạy nhanh hơn. Giúp tôi ưu hơn khi thực hiện những bài toán trong deeplearning. * Nếu dữ liệu nhiều đòi hỏi tính toán nhanh thì thực hiện những tính toán bằng vectorized sẽ nhanh và tối ưu hơn. | |

1. L1 loss: dùng để đánh giá hiệu suất của mô hình. Loss càng lớn thì predict bị overfitting.

Trong deep learning có thể dùng Gradient descent để train và minimize cost.

loss = np.sum(np.abs(yhat - y)). # abs() tính trị tuyệt đối

1. L2 loss: loss = np.sum(np.dot((yhat - y), (yhat - y)))
2. **Notion and important knowledge:**

**Remember:**

* np.exp(x) hoạt động với mọi np.array(x) và áp dụng cho hàm mũ
* Sigmoid function and gradient dùng để tính xác xuất và dùng cho những loại bài phân loại 2 lớp (Binary classification)
* Np.reshape() giúp giữ thẳng các kích thước matran/vector
* Vectorization thì quan trọng trong deeplearning giúp tính toán tối ưu, hiệu quả, nhanh.
* Dùng L1, L2 để tính toán mất mát của mô hình để tránh bị overfitting.