[Week 2 - PA 1 - Python Basics with Numpy](https://nbviewer.jupyter.org/github/amanchadha/coursera-deep-learning-specialization/blob/master/C1%20-%20Neural%20Networks%20and%20Deep%20Learning/Week%202/Python%20Basics%20with%20Numpy/Python_Basics_With_Numpy_v3a.ipynb)

Chào mừng bạn đến với nhiệm vụ đầu tiên của bạn. Bài tập này cung cấp cho bạn một giới thiệu ngắn gọn về Python. Ngay cả khi bạn đã sử dụng Python trước đây, điều này sẽ giúp bạn làm quen với các chức năng mà chúng tôi cần.

**Hướng dẫn:**

* Bạn sẽ sử dụng Python 3.
* **Tránh** sử dụng vòng lặp **for** và vòng lặp **while**, trừ khi bạn được yêu cầu rõ ràng làm như vậy.
* Không sửa đổi the (# GRADED FUNCTION [function name]) comment trong một số ô. Công việc của bạn sẽ không được chấm điểm nếu bạn thay đổi điều này. Mỗi ô chứa chú thích đó chỉ nên chứa một hàm.
* Sau khi mã hóa hàm của bạn, hãy chạy ô ngay bên dưới nó để kiểm tra xem kết quả của bạn có chính xác hay không.

**Sau nhiệm vụ này, bạn sẽ:**

* Có thể sử dụng iPython Notebooks
* Có thể sử dụng các hàm numpy và các phép toán vectơ / ma trận numpy
* Hiểu khái niệm "broadcasting"
* Có thể vectơ hóa mã

Bắt đầu nào!

Giới thiệu về iPython Notebooks

* Chạy các ô của bạn bằng SHIFT + ENTER (hoặc "Chạy ô")
* Chỉ viết mã trong các khu vực được chỉ định bằng Python 3
* Không sửa đổi mã bên ngoài các khu vực được chỉ định

1. Xây dựng các chức năng cơ bản với numpy

Những gì bạn cần nhớ:

* np.exp (x) hoạt động với bất kỳ np.array x nào và áp dụng hàm mũ cho mọi tọa độ
* hàm sigmoid và độ dốc của nó
* image2vector thường được sử dụng trong học sâu
* np.reshape được sử dụng rộng rãi. Trong tương lai, bạn sẽ thấy rằng việc giữ thẳng kích thước ma trận / vectơ sẽ giúp loại bỏ rất nhiều lỗi.
* numpy có các chức năng tích hợp hiệu quả
* boardcasting hình cực kỳ hữu ích

1.1. Hàm sigmoid, np.exp ()

1.2. Sigmoid Gradient (Đạo hàm)

1.3. Định hình lại mảng

1.4. Chuẩn hóa các hàng

1.5. Broadcasting và hàm softmax

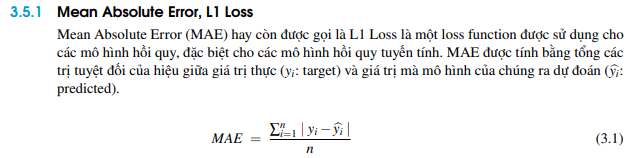
2. Vectơ hóa

2.1. Thực hiện các hàm mất L1 và L2

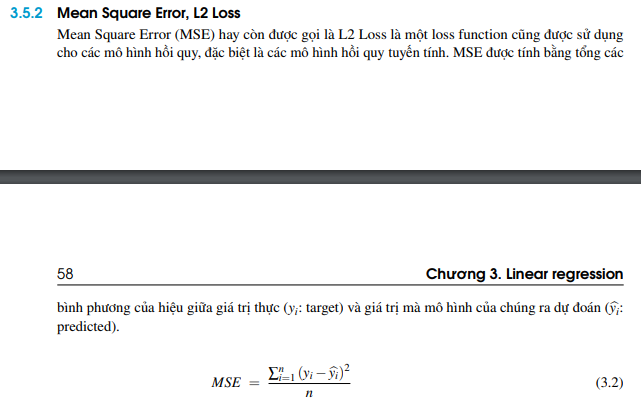
2.2. Theory – Phổ cập kiến thức L1 và L2

Nguồn: Sách DL nntuan – Trang 57 và 58

a. L1 - Lỗi trị đối trung bình



b. L2- Lỗi bình phương trung bình



c. So sánh

- outlier: điểm dị biệt

- Nếu có nhiều điểm dị biệt thì đương nhiên là MAE (L1) sẽ tốt hơn MSE (L2), MSE bình phương nên sẽ làm error bị phóng đại.

- Tuy nhiên là do MSE dễ tính đạo hàm hơn, nên là người ta hay sử dụng MSE hơn.

[Week 2 - PA 2 - Logistic Regression with a Neural Network mindset](https://nbviewer.jupyter.org/github/amanchadha/coursera-deep-learning-specialization/blob/master/C1%20-%20Neural%20Networks%20and%20Deep%20Learning/Week%202/Logistic%20Regression%20as%20a%20Neural%20Network/Logistic_Regression_with_a_Neural_Network_mindset_v6a.ipynb)

1 - Packages

2 - Overview of the Problem set

- Tiền xử lý – Các bước phổ biến để tiền xử lý một tập dữ liệu mới là:

* Tìm ra các kích thước và hình dạng của vấn đề (m\_train, m\_test, num\_px, ...)
* **Reshape** dataset, để mỗi ví dụ bây giờ là một vectơ có kích thước (num\_px \* num\_px \* 3, 1)
* **Chuẩn hóa** dữ liệu

3 - General Architecture of the learning algorithm

- Các bước chính: Trong bài tập này, ta sẽ thực hiện các bước sau đây:

* Khởi tạo các tham số của model
* Học các tham số cho model bằng cách tối thiểu hóa hàm cost
* Sử dụng các tham số đã học được để đưa ra dự đoán trên test set
* Phân tích kết quả và kết luận

4 - Building the parts of our algorithm

The main steps for building a Neural Network are:

1. Define the model structure (such as number of input features)
2. Initialize the model's parameters
3. Loop:
   * Calculate current loss (forward propagation)
   * Calculate current gradient (backward propagation)
   * Update parameters (gradient descent)

You often build 1-3 separately and integrate them into one function we call model().

4.1 - Helper functions

def sigmoid(z):

    s = 1/(1+np.exp(-z))

    return s

4.2 - Initializing parameters

def initialize\_with\_zeros(dim):

    w = np.zeros((dim,1))

    b = 0

    return w, b

4.3 - Forward and Backward propagation

def propagate(w, b, X, Y):

    m = X.shape[1]

    A = sigmoid(np.dot(w.T,X) + b)

    cost = np.sum(((- np.log(A))\*Y + (-np.log(1-A))\*(1-Y)))/m

    dw = (np.dot(X,(A-Y).T))/m

    db = (np.sum(A-Y))/m

    assert(dw.shape == w.shape)

    assert(db.dtype == float)

    cost = np.squeeze(cost)

    assert(cost.shape == ())

    grads = {"dw": dw,

             "db": db}

    return grads, cost

4.4 – Optimization

def optimize(w, b, X, Y, num\_iterations, learning\_rate, print\_cost = False):

    costs = []

    for i in range(num\_iterations):

        grads, cost = propagate(w, b, X, Y)

        dw = grads["dw"]

        db = grads["db"]

        w = w - (learning\_rate\*dw)

        b = b - (learning\_rate\*db)

        if i % 100 == 0:

            costs.append(cost)

        if print\_cost and i % 100 == 0:

            print ("Cost after iteration %i: %f" %(i, cost))

    params = {"w": w,

              "b": b}

    grads = {"dw": dw,

             "db": db}

    return params, grads, costs

4.5 – Prediction (optional)

def predict(w, b, X):

    '''

    Predict whether the label is 0 or 1 using learned logistic regression parameters (w, b)

    Arguments:

    w -- weights, a numpy array of size (num\_px \* num\_px \* 3, 1)

    b -- bias, a scalar

    X -- data of size (num\_px \* num\_px \* 3, number of examples)

    Returns:

    Y\_prediction -- a numpy array (vector) containing all predictions (0/1) for the examples in X

    '''

    m = X.shape[1]

    Y\_prediction = np.zeros((1,m))

    w = w.reshape(X.shape[0], 1)

    A = sigmoid(np.dot(w.T,X) + b)

    #### WORKING SOLUTION 1: USING IF ELSE ####

    #for i in range(A.shape[1]):

        ## Convert probabilities A[0,i] to actual predictions p[0,i]

        ### START CODE HERE ### (≈ 4 lines of code)

        #if (A[0,i] >= 0.5):

        #    Y\_prediction[0, i] = 1

        #else:

        #    Y\_prediction[0, i] = 0

        ### END CODE HERE ###

    #### WORKING SOLUTION 2: ONE LINE ####

    #for i in range(A.shape[1]):

        ## Convert probabilities A[0,i] to actual predictions p[0,i]

        ### START CODE HERE ### (≈ 4 lines of code)

        #Y\_prediction[0, i] = 1 if A[0,i] >=0.5 else 0

        ### END CODE HERE ###

    #### WORKING SOLUTION 3: VECTORISED IMPLEMENTATION ####

    Y\_prediction = (A >= 0.5) \* 1.0

    assert(Y\_prediction.shape == (1, m))

    return Y\_prediction

5 - Merge all functions into a model

def model(X\_train, Y\_train, X\_test, Y\_test, num\_iterations = 2000, learning\_rate = 0.5, print\_cost = False):

    # initialize parameters with zeros (≈ 1 line of code)

    w, b = initialize\_with\_zeros(X\_train.shape[0])

    # Gradient descent (≈ 1 line of code)

    parameters, grads, costs = optimize(w, b, X\_train, Y\_train, num\_iterations, learning\_rate, print\_cost)

    # Retrieve parameters w and b from dictionary "parameters"

    w = parameters["w"]

    b = parameters["b"]

    # Predict test/train set examples (≈ 2 lines of code)

    Y\_prediction\_test = predict(w, b, X\_test)

    Y\_prediction\_train = predict(w, b, X\_train)

    # Print train/test Errors

    print("train accuracy: {} %".format(100 - np.mean(np.abs(Y\_prediction\_train - Y\_train)) \* 100))

    print("test accuracy: {} %".format(100 - np.mean(np.abs(Y\_prediction\_test - Y\_test)) \* 100))

    d = {"costs": costs,

         "Y\_prediction\_test": Y\_prediction\_test,

         "Y\_prediction\_train" : Y\_prediction\_train,

         "w" : w,

         "b" : b,

         "learning\_rate" : learning\_rate,

         "num\_iterations": num\_iterations}

    return d

6 - Further analysis (optional/ungraded exercise)

Hai cái dưới này nhiều hàm numpy với matplotlib lạ nên để sau hẵng học nhé.

7 - Test with your own image (optional/ungraded exercise)

[Week 3 - PA 3 - Planar data classification with one hidden layer](https://nbviewer.jupyter.org/github/amanchadha/coursera-deep-learning-specialization/blob/master/C1%20-%20Neural%20Networks%20and%20Deep%20Learning/Week%203/Planar%20data%20classification%20with%20one%20hidden%20layer/Planar_data_classification_with_onehidden_layer_v6c.ipynb)

- Sau buổi học này, tôi nắm được quy trình làm 1 bài phân loại dữ liệu phẳng với neural network 1 lớp ẩn.

+ Khởi tạo

+ Vòng for

* forward
* cost
* backward
* update

+ Predict

- Trong lúc code, tôi đã thấu hiểu phần nào về các vấn đề vectorization, đạo hàm, số chiều, bản chất, 1/m của backpropagation.

1 – Packages

Có sklearn mới.

2 – Dataset

- X có shape (2, 400)

- Y có shape (1, 400)

- red (y = 0)

- blue (y = 1)

3 - Simple Logistic Regression

- Sử dụng HQL chỉ để thấy rằng nó không phù hợp với tập dữ liệu (không phân tách tuyến tính) này. Kết quả dự đoán cho khá thấp.

- Ta kỳ vọng ở NN mà ta sắp implement.

4 - Neural Network model

4.1 - Defining the neural network structure

def layer\_sizes(X, Y):

    n\_x = X.shape[0]

    n\_h = 4

    n\_y = Y.shape[0]

    return (n\_x, n\_h, n\_y)

4.2 - Initialize the model's parameters

def initialize\_parameters(n\_x, n\_h, n\_y):

    np.random.seed(2)

    W1 = np.random.randn(n\_h,n\_x) \* 0.01

    b1 = np.zeros((n\_h,1))

    W2 = np.random.randn(n\_y,n\_h) \* 0.01

    b2 = np.zeros((n\_y,1))

    parameters = {"W1": W1,

                  "b1": b1,

                  "W2": W2,

                  "b2": b2}

    return parameters

4.3 - The Loop

4.4 - Integrate parts 4.1, 4.2 and 4.3 in nn\_model()

4.5 - Predictions

4.6 - Tuning hidden layer size (optional/ungraded exercise)

5 - Performance on other datasets

[Week 4 - PA 4 - Building your Deep Neural Network: Step by Step](https://nbviewer.jupyter.org/github/amanchadha/coursera-deep-learning-specialization/blob/master/C1%20-%20Neural%20Networks%20and%20Deep%20Learning/Week%204/Building%20your%20Deep%20Neural%20Network%20-%20Step%20by%20Step/Building_your_Deep_Neural_Network_Step_by_Step_v8a.ipynb)

- Bài tập này: Xây dựng mạng sâu.

- Bài tập sau: Áp dụng vào phân loại hình ảnh.

- Thành quả:

* Sử dụng ReLU
* Xây dựng được mạng thần kinh sâu
* Triển khai được 1 lớp mạng thần kinh dễ sử dụng

1 - Packages

2 - Outline of the Assignment

3 - Initialization

3.1 - 2-layer Neural Network

3.2 - L-layer Neural Network

4 - Forward propagation module

4.1 - Linear Forward

4.2 - Linear-Activation Forward

4.3 - L-Layer Model

5 - Cost function

6 - Backward propagation module

6.1 - Linear backward

6.2 - Linear-Activation backward

6.3 - L-Model Backward

6.4 - Update Parameters

7 - Conclusion

[Week 4 - PA 5 - Deep Neural Network for Image Classification: Application](https://nbviewer.jupyter.org/github/amanchadha/coursera-deep-learning-specialization/blob/master/C1%20-%20Neural%20Networks%20and%20Deep%20Learning/Week%204/Deep%20Neural%20Network%20Application_%20Image%20Classification/Deep%20Neural%20Network%20-%20Application%20v8.ipynb)

1 - Packages

2 - Dataset

3 - Architecture of your model

3.1 - 2-layer neural network

3.2 - L-layer deep neural network

3.3 - General methodology

4 - Two-layer neural network

5 - L-layer Neural Network

6 - Results Analysis

7 - Test with your own image (optional/ungraded exercise)