**REPORT EXERCISE COURSERA 2**

**LOGISTIC REGRESSION WITH A NEURAL NETWORK MINDSET ([assignment](https://www.coursera.org/learn/neural-networks-deep-learning/programming/thQd4/logistic-regression-with-a-neural-network-mindset))**

1. **Exercise and knowledge:**
2. **Overview data:**

If you want to see label in test set image when you predict:

In dataset need have classes it class of image test, and use function: (“np.squeeze()”, if image is byte transform character use “decode(“utf-8”)”

* **np.squeeze():** it remove dimension have size = 1

Ex: 1-dimensional array: [1, 2, 3] ; have size (3,)

1. dimensional array: [[1, 2, 3], [4, 5, 6]]: have size(2, 3)
2. dimensional array: [[1, 2, 3, 4]]: size (1, 4)

3-dimensional array: [[1], [2], [3]]: size(1, 3, 1)

Or size (1, 3, 1, 4)

When use np.squeeze() --> size(1, 3, 1, 4) = (3, 4)

Size(1, 4) = (4, )

Size(1, 3, 1) = (3)

Ex: train\_set = np.array([[0, 1, 2, 3, 4]])

Index = 2

Label = train\_set[:, index] ---> array([2])-->size (2,)

Label = np.squeeze(label) ---> 2 -->tìm kiếm dễ dàng

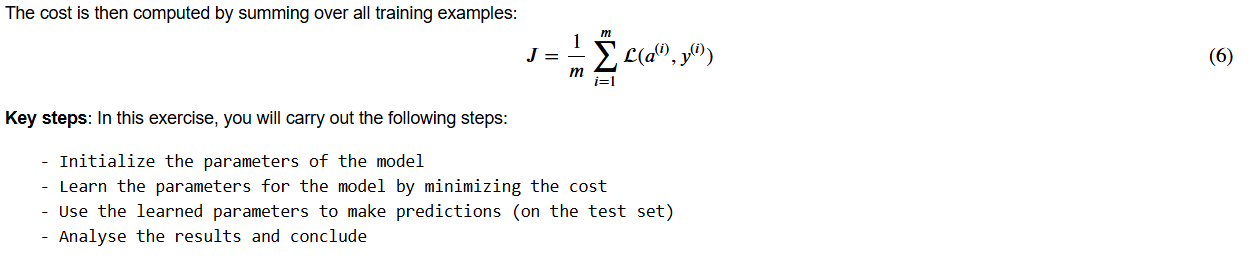
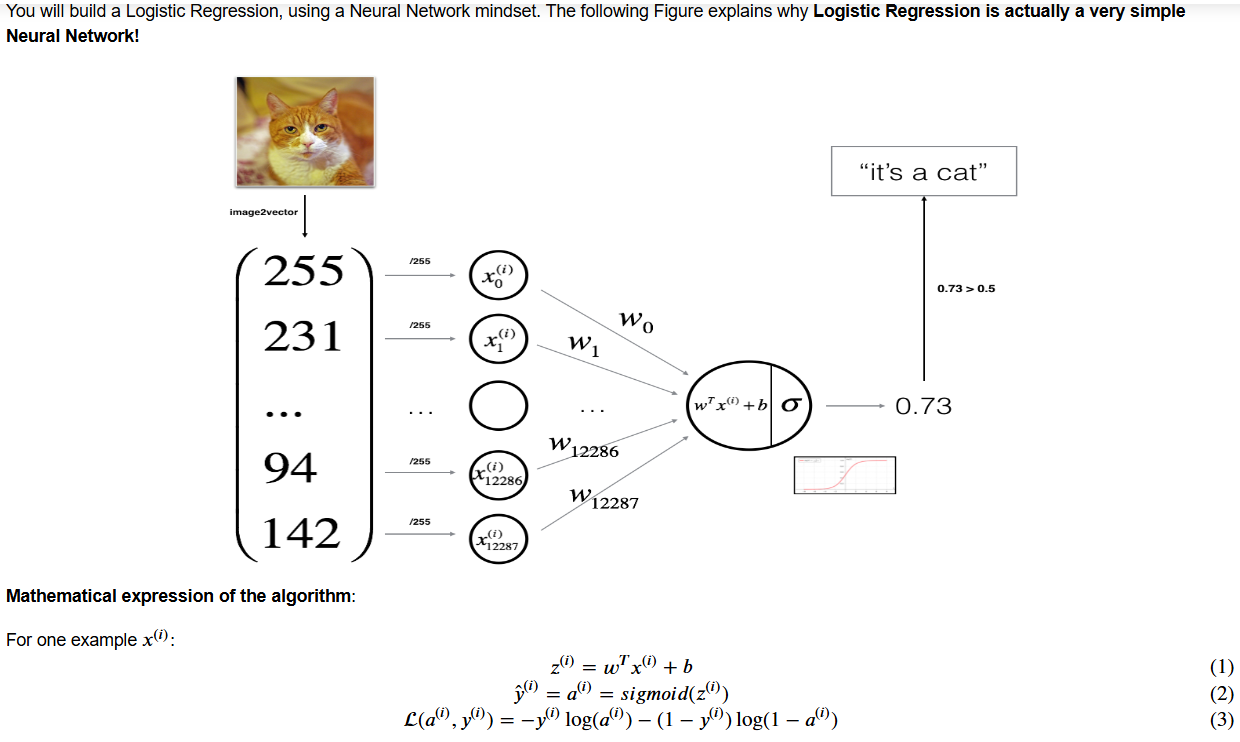
* Matrix image of shape (a, b, c, d) : a: number of samples, b, c, d: number pixel in image (height, width, number of color channels)
* **Help simplify the processing and training of machine learning mode**ls that convert image data from multi-dimensional to 2-dimensional: using np.reshape(train\_set.shape[0], -1) --> (a, b \* c \* d)

Shape[0]: collect number of sample image in train\_set

-1: auto calculation ( height \* width \* number of color channel)

* **Preprocess the image by dividing by 255:**convert pixel values from interval [0, 255] to interval [0, 1]. Helps the computer learn effectively because the input value is within a fixed range [0, 1] and is small

1. **Building the parts of out algorithm:**

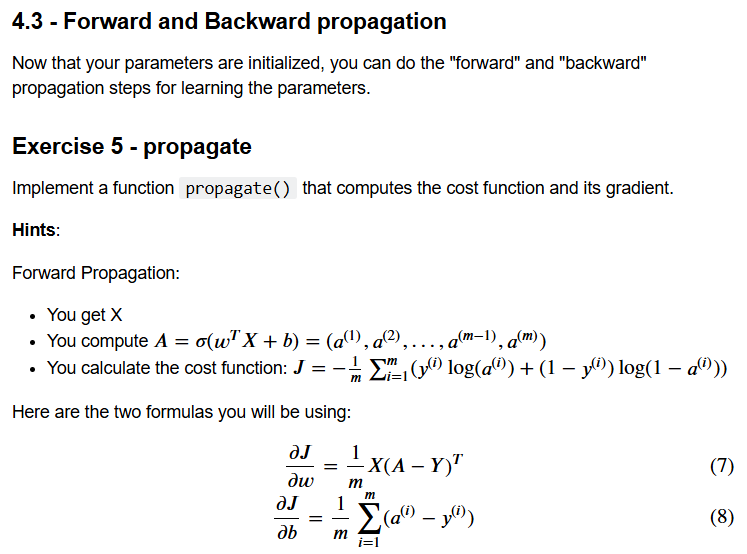


1. **Sigmoid: s = 1 / (1 + np.exp(-z))**
2. **Initializing parameters:** giá trịkhởi tạo giá trị ban đầu = 0 giúp bắt đầu quá trình huấn luyện 1 điểm xuất phát đơn giản.

w = np.zeros((dim, 1)): khởi tạo vector có số chiều dim, và tất cả các phần tử = 0

B = 0

1. **Forward and Backward propagation:**



* 1. **Forward propagation:**
     1. **Activation: prediction**

Z = np.dot(w.T, X) + b : phép tính tuyến tính

Sigmoid = sigmoid(z): tính sigmoid để tính giá trị dự đoán.

* + 1. **Cost: tính toán chi phí:**

Cost = -1 / m \* np.sum(Y \* np.log(A) + (1 - Y) \* np.log(1 - A)) : Y: nhãn thực tế, A giá trị predict

* 1. **Backward propagation: propage**
     1. **Đạo hàm của hàm chi phí:**

dw = 1 / m \* np.dot(X, (A - Y).T): đạo hàm theo vector trọng số w

db = 1 / m \* np.sum(A - Y): đạo hàm theo giá trị bais

**Note:**

* Đạo hàm của hàm chi phí để:
  + Tối ưu các tham số: qua mỗi lần lặp(epoch) các tham số được điều chỉnh để giảm giá trị của hàm chi phí---> cải thiện chất lượng model
  + Learning rate: điều chỉnh để thuật toán không bị quá chậm hay quá nhanh trong quá trình học
  + Nếu đạo hàm gần = 0, mô hình đã đến 1 điểm tối ưu

cost = np.squeeze(np.array(cost)): do hàm cost tính ra là 1 số nhưng vẫn để vào array để đảm bảo cost là 1 số vô hướng

Vd: cost = np.array([42]) shape = (1,) print(cost) = [42]

Cost = np.sqeeze(cost) -->42 shape ()

* 1. **Optimize:** cập nhật trọng số w, b để giảm thiểu hàm mất mát (cost function)
     1. Num\_iterations: số lần lặp của thuật toán gradient descent
     2. Learning\_rate: tốc độ học thuật, điều khiển mức độ thay đổi của w, b trong mỗi lần cập nhật.
     3. Print\_cost: cờ boolean để bật tắt in giá trị hàm mất mát sau mỗi lần lặp.
     4. np.deepcopy(): tạo bản sao của w, b ban đầu
     5. w = w - learning\_rate \* dw

b = b - learning\_rate \* db

Cập nhật hệ số, theo quy ước di chuyển ngược lại so với w, b ban đầu và nhân với learning\_rate để giảm thiểu hàm chi phí sinh ra cho mô hình học.

* + 1. Qua đó sẽ ghi lại cost sau 100 lần để có thể vẽ biểu đồ để xem lượng mất mát có giảm hay không

1. **Remember and notion:**
   1. Figure out the dimension and shapes of the problem(m\_train, m\_test, num\_px,..)
   2. Reshape the datasets such that each example is a vector size (number sample, height \* width \* number of color channels)
   3. “standardize” data : chuẩn hóa dữ liệu
   4. Initialize (w,b)
   5. Optimize the loss iteratively to learn parameters (w,b):
   6. Computing the cost and its gradient
   7. Updating the parameters using gradient descent
   8. Use the learned (w,b) to predict the labels for a given set of examples
   9. If the learning rate is too large we may "overshoot" the optimal value. Similarly, if it is too small we will need too many iterations to converge to the best values. That's why it is crucial to use a well-tuned learning rate.