§ Logistic Regression with a Neural Network mindset

1. Packages

numpy is the fundamental package for scientific computing with Python.
h5py is a common package to interact with a dataset that is stored on an H5 file.
matplotlib is a famous library to plot graphs in Python.
PIL and scipy are used here to test your model with your own picture at the end.

2. Overview of the Problem set

Problem Statement: You are given a dataset ("data.h5") containing:
- a training set of m_train images labeled as cat (y=1) or non-cat (y=0)
- a test set of m_test images labeled as cat or non-cat
- each image is of shape (num_px, num_px, 3) where 3 is for the 3 channels (RGB).

Thus, each image is square (height = num_px) and (width = num_px).

① train_set_X_orig, train_set_y, test_set_X_orig, test_set_y, classes = load_dataset()

m_train = train_set_X_orig.shape[0] → number of training examples (209)
m_test = test_set_X_orig.shape[0] → number of testing examples (50)
num_px = train_set_X_orig.shape[1] → height / width of each images (64)

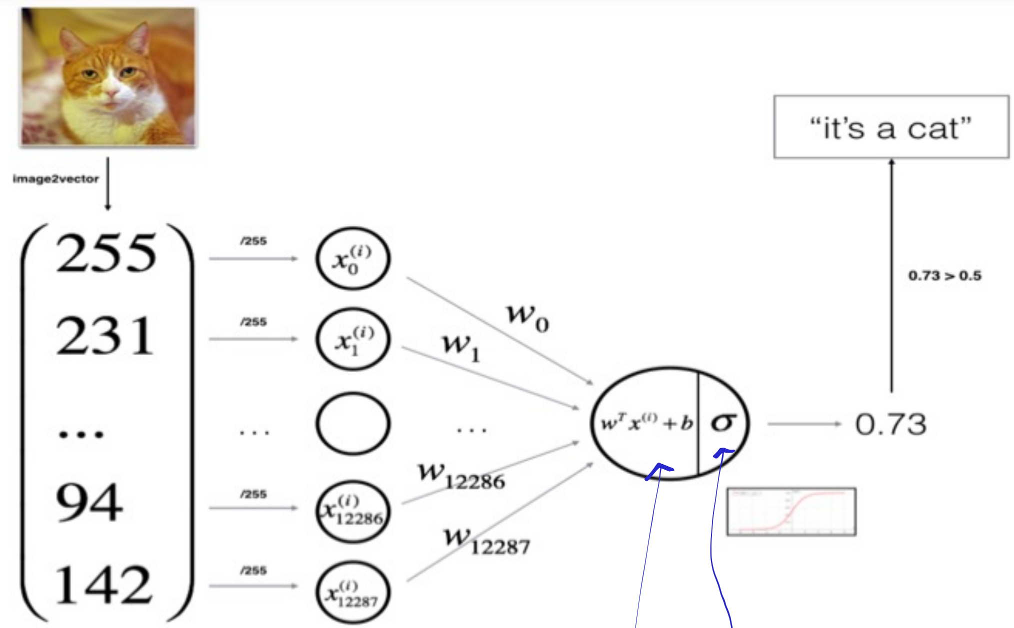
train_set_X shape : (209, 64, 64, 3) → 3 channels → RGB
number of training examples   height / width / depth
train_set_y shape : (1, 209) → 209 labels
test_set_X shape : (50, 64, 64, 3)
test_set_y shape : (1, 50) → numbers of test examples

② flatten
train_set_X_flatten = train_set_X_orig.reshape(train_set_X_orig.shape[0], -1).T
test_set_X_flatten = test_set_X_orig.reshape(test_set_X_orig.shape[0], -1).T
train_set_X_flatten shape : (12288, 209)   64×64×3 = 12288
test_set_X_flatten shape : (12288, 50)

③ standardize → center and standardize dataset
train_set_X = train_set_X_flatten / 255 → RGB, value from 0~255
test_set_X = test_set_X_flatten / 255

3. General Architecture of learning algorithm

design algorithm to distinguish cat / non-cat images



Mathematical expression of the algorithm:

For one example $x^{(i)}$:

$$z^{(i)} = w^T x^{(i)} + b$$
$$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)})$$
$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1-y^{(i)}) \log(1-a^{(i)})$$

The cost is then computed by summing over all training examples:

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

Key steps: In this exercise, you will carry out the following steps:

```
- Initialize the parameters of the model
- Learn the parameters for the model by minimizing the cost
- Use the learned parameters to make predictions (on the test set)
- Analyse the results and conclude
```

4. Building the parts of 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)

① helper functions

s = 1/(1+ np.exp(-z)) → $sigmoid(z) = \frac{1}{1+e^{-z}}$

② initializing parame

w: np.zeros(shape=(dim, 1))

b = 0

③ Forward and Backward propagation

Hints:

Forward Propagation:

- You get X
- You compute $A = \sigma(w^T X + b) = (a^{(0)}, a^{(1)}, \ldots, a^{(m-1)}, a^{(m)})$
- You calculate the cost function: $J = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) \log(1-a^{(i)})$

Here are the two formulas you will be using:

$$\frac{\partial J}{\partial w} = \frac{1}{m} X (A - Y)^T \tag{7}$$
$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) \tag{8}$$

A = sigmoid(np.dot(w.T, X)+b) → $A=\sigma(\underbrace{\underbrace{w^T}_{1\times 12288}\underbrace{X}_{12288\times 209}}_{1\times 209}+b) \rightarrow 1\times 209$

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

→ $J=-\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(a^{(i)})+(1-y^{(i)})\log(1-a^{(i)})$

dw = 1/m*np.dot(X,(A-Y).T) → $\frac{dJ}{dw}=\frac{1}{m}\underbrace{\underbrace{X}_{12288\times 209}\underbrace{(A-Y)^T}_{209\times 1}}_{12288\times 1}$

db = 1/m*np.sum(A-Y)

→ $\frac{dJ}{dw}=\frac{1}{m}\sum_{i=1}^{m}(a^{(i)}-y^{(i)})$

④ Optimization

for i in range(num_iterations):

w = w - learning_rate * dw

b = b - learning_rate * db

→ $\theta=\theta-\alpha d\theta$

⑤ Prediction

$\hat{Y}=A=\sigma(w^TX+b)$ → $>0.5 \rightarrow \hat{Y}=1$; $<0.5 \rightarrow \hat{Y}=0$

A = sigmoid(np.dot(w.T, X)+b)

for i in range(A.shape[1]):

Y_prediction[0,i] = 1 if A[0,i] > 0.5 else 0

5. Merge all functions into a model

(1) Initialize parameters

w, b = initialize_with_zeros(X_train.shape[0])

(2) Gradient descent

parameters, grads, costs = optimize(w, b, X_train, Y_train, num_iteration, learning_rate, print_cost)

w = parameters["w"]

b = parameters["b"]

retrieve parameters

(3) predict test/train set examples

```
Y_prediction_test = predict(w, b, X_test)

Y_prediction_train = predict(w, b, X_train)
```

(4) print train / test errors

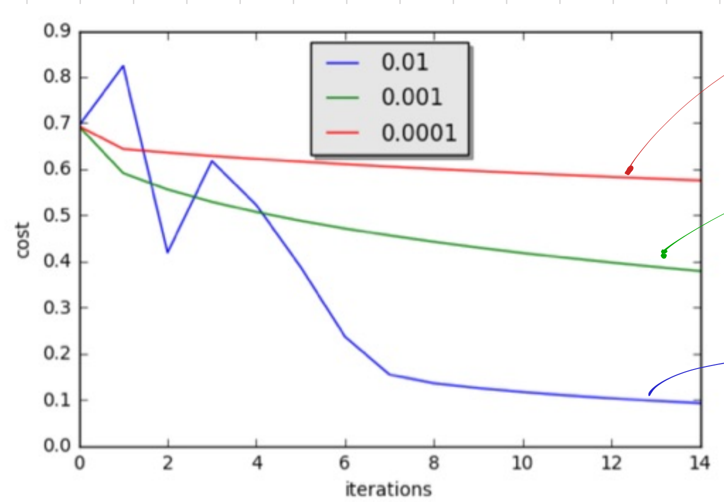
```
print("train accuracy: {}%".format(100 - np.mean(np.abs(Y_prediction_train
                                                   - Y_train)) * 100))
```

6. Choice of learning rate

Choice of learning rate

Reminder: In order for Gradient Descent to work you must choose the learning rate wisely. The learning rate α determines how rapidly we update the parameters. 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.

Let's compare the learning curve of our model with several choices of learning rates. Run the cell below. This should take about 1 minute. Feel free also to try different values than the three we have initialized the `learning_rates` variable to contain, and see what happens.



train: 68.42%
test: 38.0%

train: 88.995%
test: 64.0%

train: 99.52%
test: 68.0%

Interpretation:

- Different learning rates give different costs and thus different predictions results.
- If the learning rate is too large (0.01), the cost may oscillate up and down. It may even diverge (though in this example, using 0.01 still eventually ends up at a good value for the cost).
- A lower cost doesn't mean a better model. You have to check if there is possibly overfitting. It happens when the training accuracy is a lot higher than the test accuracy.
- In deep learning, we usually recommend that you:
  - Choose the learning rate that better minimizes the cost function.
  - If your model overfits, use other techniques to reduce overfitting. (We'll talk about this in later videos.)