TD 1: Deep Neural Network

1. Objective

In this TD, we will begin with neural network. The problem is discussed in problem section where we will apply neural network to solve a binary classification problem (cat or not cat). A demo implementation is shown in Review section where we will implement a Logistic Regression. In the Exercises section, you will create the functions by yourself to build a network with one hidden layer to solve the same problem in Review section.

After this exercise, you will:

- Have a basic knowledge about Numpy functions, operations for matrix, vectors, …
- Use vectorization to avoid loop functions (for, while, ...)
- Be able to build a neural network (logistic regression, one hidden layer)
- Know how to initialize the parameters, calculate the cost function, gradient descent
- Train a network and predict result from a trained model

2. Software and libraries:

Software and libraries require:

- Python
- Jupyter (iPython notebook)
- Numpy
- Matplotlib
- H5py

3. Problem

You are provided a dataset ("data.h5"), which containing:

- A training set of m training labeled images (as cat (y=1) or not cat (y=0))
- A test set of n images labeled as cat or not cat
- Each image is presented in RGB color mode in the shape of (**width, height, 3**). Thus, each image is square (width = height)

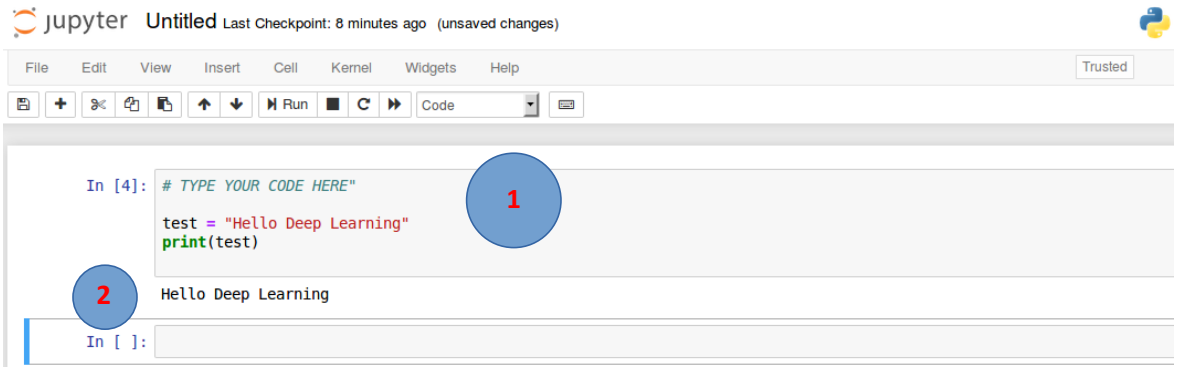
Problem: Implement a neural network to discover an image contains cat or non with two scenarios:

- Use Logistic Regression (already in Review section)
- Use one hidden layer neural network (your turn)

4. Review:

4.1. Jupyter Notebook environments

Open a terminal and type "jupyter notebook" to open the Jupyter notebook. The user interface of Jupyter Notebook likes images below. In this environment, we implement our code in region (1). After running, the results will be appeared in region (2) (even we have any error in our code).



4.2. Basic Numpy functions with activation functions

Question 1: Using Numpy to implement the activation functions: **sigmoid(*x*), tanh(x), relu(x);** where x can be a number, a vector, or a matrix.

Reminder:

- $sigmoid(x) = \frac{1}{1+e^{-x}}$
- $tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

- $ReLU(x) = max(0, x)$

```
import numpy as np

def sigmoid(x):
    s = 1/(1 + np.exp(-x))
    return s

def tanh(x):
    s = (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))
    return s

def relu(x):
    s = np.maximum(0,x)
    return s
```

Question 2: Using numpy to implement the derivative of activation functions: ***sigmoid, tanh, relu***.

Reminder:

- $sigmoid\_grad(x) = \sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$
- $tanh\_grad(x) = 1 - (\sigma(x))^2$
- $relu\_grad(x) = \begin{cases} 0 & if\ x \leq 0 \\ 1 & if\ x > 0 \end{cases}$

```
def sigmoid_grad(x):
    s = sigmoid(x)
    s_grad = s * (1-s)
    return s_grad

def tanh_grad(x):
    s = tanh(x)
    return 1 - (s * s)

def relu_grad(x):
    x[x>0] = 1
    x[x<=0] = 0
    return x
```

4.3. Data preprocess

Question 3.1: Implement ***load_data()*** function to load data.

```
import h5py
train_path = 'data/train_catvnoncat.h5'
test_path = 'data/test_catvnoncat.h5'

def load_data(train_path, test_path):
    train_dataset = h5py.File(train_path,'r')
    train_set_X = np.array(train_dataset['train_set_x'][:])
    train_set_Y = np.array(train_dataset['train_set_y'][:])

    test_dataset = h5py.File(test_path,'r')
    test_set_X = np.array(test_dataset['test_set_x'][:])
    test_set_Y = np.array(test_dataset['test_set_y'][:])

    classes = np.array(test_dataset['list_classes'][:])
    return train_set_X, train_set_Y, test_set_X, test_set_Y, classes


train_set_X, train_set_Y, test_set_X, test_set_Y, classes = load_data(train_path, test_path)
print (train_set_X.shape)
```

Question 3.2: Implement ***reshape_data***() to reshape the loaded data into single vector (from (width, height, 3) to (width * height * 3,1)).

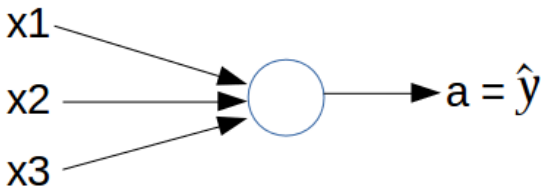
```
def reshape_data(x_dataset, y_dataset):
    x_dataset_reshape = x_dataset.reshape((x_dataset.shape[1] * x_dataset.shape[2] * x_dataset.shape[3],x_dataset.shape[0]))
    y_dataset_reshape = y_dataset.reshape((1,y_dataset.shape[0]))
    return x_dataset_reshape, y_dataset_reshape

# Run the function to check the errors
train_set_X_reshape, train_set_Y_reshape = reshape_data(train_set_X, train_set_Y)
test_set_X_reshape, test_set_Y_reshape = reshape_data(test_set_X,test_set_Y)
print('Train x dataset: ' + (str(train_set_X_reshape.shape)))
  print('Train y dataset: ' + (str(train_set_Y_reshape.shape)))
  print('Test x dataset: ' + (str(test_set_X_reshape.shape)))
```

```
12. print('Test y dataset: ' + (str(test_set_Y_reshape.shape)))
```

4.4. Solve problem with a Logistic Regression model

In this part, you will build a Logistic Regression using a Neural Network.



Mathematical expression of the algorithms:

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = sigmoid\left(z^{(i)}\right)$$

$$L\left(a^{(i)}, y^{(i)}\right) = -y^{(i)} log\left(a^{(i)}\right) - \left(1 - y^{(i)}\right) log\left(1 - a^{(i)}\right)$$

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

$$J = \frac{1}{m} \sum_{i=0}^{m} L\left(a^{(i)} - y^{(i)}\right)$$

The main steps for building a Neural Network are:

a. Define the model structure (such as number of input features)
b. Initialize the model's parameters
c. Loop:
   - Calculate forward propagation
   - Calculate backward propagation
d. Update parameters (gradient descent)

Question 4: Implement ***initialize_parameters_with_zeros()*** function to initialize w parameter and b parameter with zero value.

```
def initialize_parameters_with_zeros(dims):
    w = np.zeros((dims,1))
    b = 0
    return w, b
```

Question 5: Implement ***propagate()*** function to compute the results for forward and backward propagation.

Hints:

Forward Propagation

- Get X
- Compute $A = \sigma(w^T X + b) = (a^1, a^2, \dots, a^m)$
- Calculate the cost function: $J = \frac{-1}{m}\sum_{i=1}^{m}\left(a^{(i)} - y^{(i)}\right)$

Backward Propagation

- $\frac{dJ}{dw} = \frac{1}{m}X(A-Y)^T$
- $\frac{dJ}{db} = \frac{-1}{m}\sum_{i=1}^{m}\left(a^{(i)} - y^{(i)}\right)$

```
def propagate(X, Y, w, b):
    m = X.shape[1]
    A = sigmoid(np.dot(w.T,X) + b)
    cost = -np.sum(Y * np.log(A) + (1 - Y) * np.log(1- A))/m

    dw = np.dot(X,(A-Y).T)/m
    db = np.sum(A - Y)/m
    cost = np.squeeze(cost)

       dicts = {'dw':dw, 'db':db}
    return dicts, cost
```

Question 6: Implement ***optimize***() function to update the parameters (w, b)

```
def optimize(X,Y, w, b, num_iterations, learning_rate):
    costs = []
    for i in range(num_iterations):
        grads, cost = propagate(X,Y,w,b)
        dw = grads['dw']
        db = grads['db']
        w = w - learning_rate * dw
        b = b - learning_rate * db
```

```
        if i % 100 == 0:
                costs.append(cost)
        params = {'w':w, 'b':b}
        grads = {'dw':dw, 'db':db}
    return params, grads, costs
```

Question 7: Implement ***predict***() function to predict the label for an image x. The prediction includes two steps:

Hints:

- Compute $\hat{Y} = A = \sigma(w^T X + b)$
- Convert the entries of an output into 0 (if activation < 0.5) or 1(if activation > 0.5), then store the predictions in a vector Y_prediction.

```
def predict(X_test, w, b):
    m = X_test.shape[1]
    y_prediction = np.zeros((1,m))
    w = w.reshape(X_test.shape[0],1)
    A = sigmoid(np.dot(w.T, X_test) + b)
    for i in range(A.shape[1]):
        y_prediction[0,i] = 0 if A[0,i] <= 0.5 else 1

    return y_prediction
```

Question 8: Implement ***compute_network***() to merge together all functions.

```
def compute_network(X_train, Y_train, X_test, Y_test, num_iterations
 = 2000, learning_rate = 0.5):
    w,b = initialize_parameters_with_zeros(X_train.shape[0])
    params, grads, costs = optimize(X_train, Y_train, w, b, num_iter
ations, learning_rate)
    w = params['w']
    b = params['b']

    y_prediction_train = predict(X_train,w,b)
    y_prediction_test = predict(X_test, w, b)
    print("train accuracy: {} %".format(100 - np.mean(np.abs(y_predi
ction_train - Y_train)) * 100))
        print("test accuracy: {} %".format(100 - np.mean(np.abs(y_pre
diction_test - Y_test)) * 100))

        d = {"costs": costs,
             "Y_prediction_test": y_prediction_test,
```

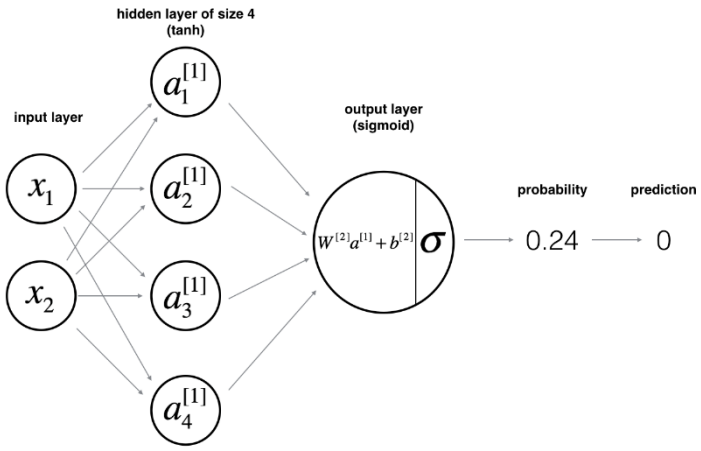
```
            "Y_prediction_train" : y_prediction_train,
            "w" : w,
            "b" : b,
            "learning_rate" : learning_rate,
            "num_iterations": num_iterations}
    return d
```

Question 9: Train the network with different learning rates (i.e. 0.1, 0.01, 0.001) and comparing the results.

```
learning_rates = np.array([0.1, 0.01, 0.001])
for lrate in learning_rates:
    compute_network(train_set_X_reshape, train_set_Y_reshape, test_set_X_reshape, test_set_Y_reshape, num_iterations = 2000, learning_rate = lrate)
```

5. Exercises

In this section, you will implement a neural network with one hidden layer to solve the same problem with Review section. Your model is as following:



Mathematical expression of the algorithms for one example $x^{(i)}$:

$$z^{[1](i)} = w^{[1]}x^{(i)} + b^{[1](i)}$$

$$a^{[1](i)} = tanh\left(z^{[1](i)}\right)$$

$$z^{[2](i)} = w^{[2]}a^{[1](i)} + b^{[2](i)}$$

$$\hat{y}^{(i)} = a^{[2](i)} = sigmoid\left(z^{[1](i)}\right)$$

$$y_{prediction}^{(i)} = f(x) = \begin{cases} 1, if a^{[2](i)} < 0 \\ 0, otherwise \end{cases}$$

$$L\left(a^{[2](i)}, y^{(i)}\right) = -y^{(i)} log\left(a^{[2](i)}\right) - \left(1 - y^{(i)}\right) log\left(1 - a^{[2](i)}\right)$$

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

$$J = \frac{1}{m} \sum_{i=0}^{m} L\left(a^{(i)} - y^{(i)}\right)$$

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 forward propagation
   - Calculate backward propagation
   - Update parameters (gradient descent)

Exercise 1: Define three variables: *n_x*, *n_h*, *n_y* are the size of input layer, the size of hidden layer and the size of the output layer, respective. In this exercise, we hardly set the hidden layer size to be **4** (n_h=4)**.**

Exercise 2: Implement ***initialize_parameters()*** function. You have to initialize w and b parameters as random matrix using **numpy** library (not zeros).

Exercise 3: Implement the ***forward_propagation()*** function.

Hints:

- Look the mathematical representation of the classifier
- You can use **sigmoid**() function that you have implemented and np.tanh() function to implement tanh activation function.
- The steps you have to implement are:
  - Retrieve the parameter from your initialization.
  - Implement forward propagation. Compute $Z^{[1]}, A^{[1]}, Z^{[2]}, A^{[2]}$
  - Keep the values of parameters for backward propagation

Exercise 4: Implement the ***compute_cost()*** function to compute the cost value J.

Hints: There are many ways to implement the cross-entropy loss. You can use either np.multiply() and then np.sum() or directly np.dot()

Exercise 5: Implement the ***backward_propagation()*** function

Exercise 6: Implement the ***update_rule()*** function to update the parameters of your network. This functions have to use (dW1, db1, dW2, db2) in order to update the value of (W1, b1, W2, b2).

Hints: The general rule to update the parameter is $\theta = \theta - \alpha \frac{dJ}{d\theta}$, where α is the learning rate and θ presents for a parameter (W or b).

Exercise 7: Merge together the functions that you have implemented for your network into ***nn_one_hidden_model()*** function.

Exercise 8: Implement the ***prediction()*** function to use your model to predict the test examples. Then, comparing the result with the result of Logistic Regression.

Hints: Your prediction should be set to 1 or 0 depending on the value of activation.

$$prediction = y_{prediction} = 1\{activation > 0.5\}$$

Exercise 9: Change the size of hidden layer and re-run the model (i.e. 5, 10, 20, 50). Then, comparing the results of model among different number of hidden units.