Weekly Work Report

Ruyue Han

VISION@OUC

July 22, 2018

1 Deep Learning

1.1 Vectorization

1.1.1 why using vectorization

for-loops and while-loops computation wast time,especily you have to run millions of data. vectoring data and using vector computation can quickly get the result .And vector computation is almost 300 times faster than for-loops computation.

1.1.2 vectorizing Logistic Regression

There are m samples ,each sample has n features,using vectoring :

$$X = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(m)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{n\times m} \tag{1}$$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(m)} \end{bmatrix}_{1\times m} \tag{2}$$

$$W = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}_{n\times 1} \tag{3}$$

because one sample :

$$z^{(i)} = W^T x^{(i)} + b \tag{4}$$

$$a^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}} \tag{5}$$

so m sample :

$$Z = W^T X + b \tag{6}$$

$$\hat{Y} = A = \begin{bmatrix} a^{(1)} & a^{(2)} & a^{(3)} & \dots & a^{(m)} \end{bmatrix}_{1\times m} = \sigma(Z) \tag{7}$$

vectorizing logistic regressions gradient computation:

$$dZ = \begin{bmatrix} dz^{(1)} & dz^{(2)} & \dots & dz^{(m)} \end{bmatrix}_{1\times m} = \begin{bmatrix} a^{(1)} & a^{(2)} & \dots & a^{(m)} \end{bmatrix}_{1\times m} - \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m)} \end{bmatrix}_{1\times m} = A - Y \tag{8}$$

$$dW = \begin{bmatrix} dw_1 \\ dw_2 \\ \vdots \\ dw_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(m)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{n\times m} \begin{bmatrix} dz^{(1)} \\ dz^{(2)} \\ \dots \\ dz^{(m)} \end{bmatrix}_{m\times 1} = X dZ^T \tag{9}$$

average dW and db:

$$dW = \frac{X dZ^T}{m} \tag{10}$$

$$db = \frac{dz^{(1)} + dz^{(2)} + \cdots + dz^{(m)}}{m} \tag{11}$$

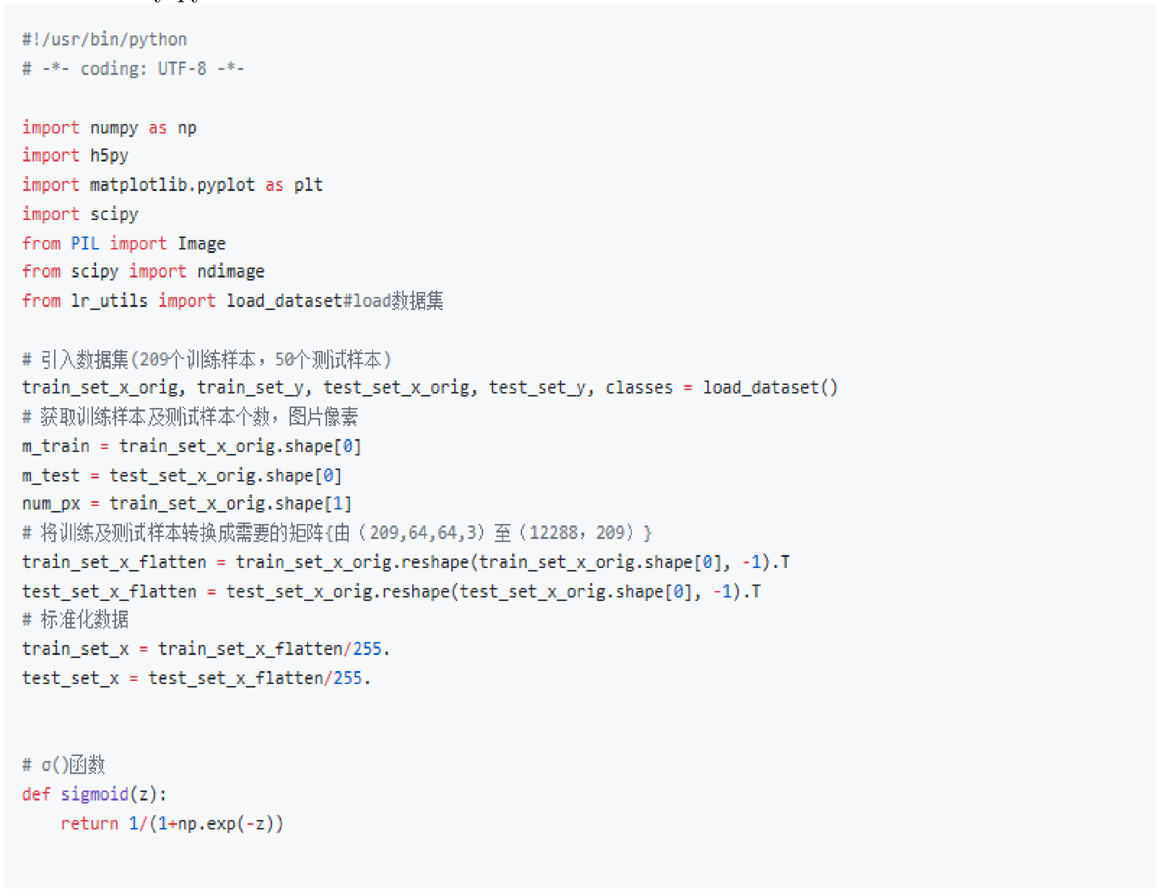
finally:

$$W = W - \alpha * dW \tag{12}$$

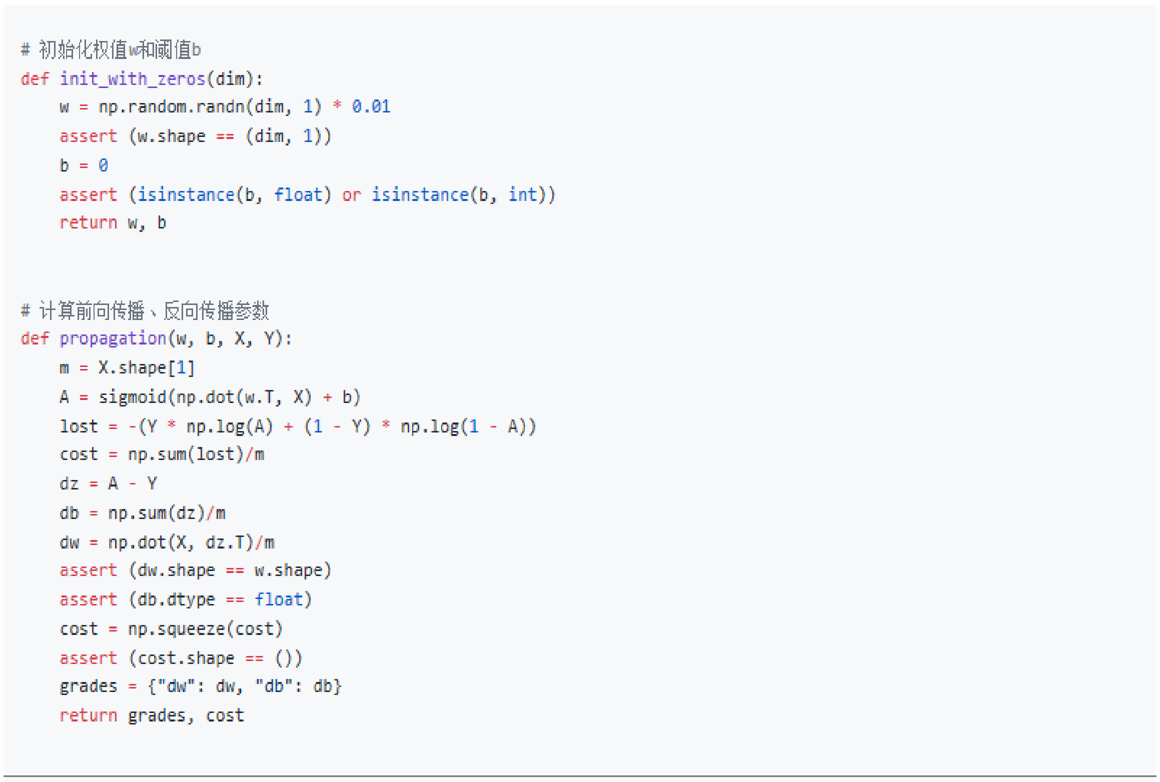
$$b = b - \alpha * db \tag{13}$$

1.1.3 building recognizing cats model

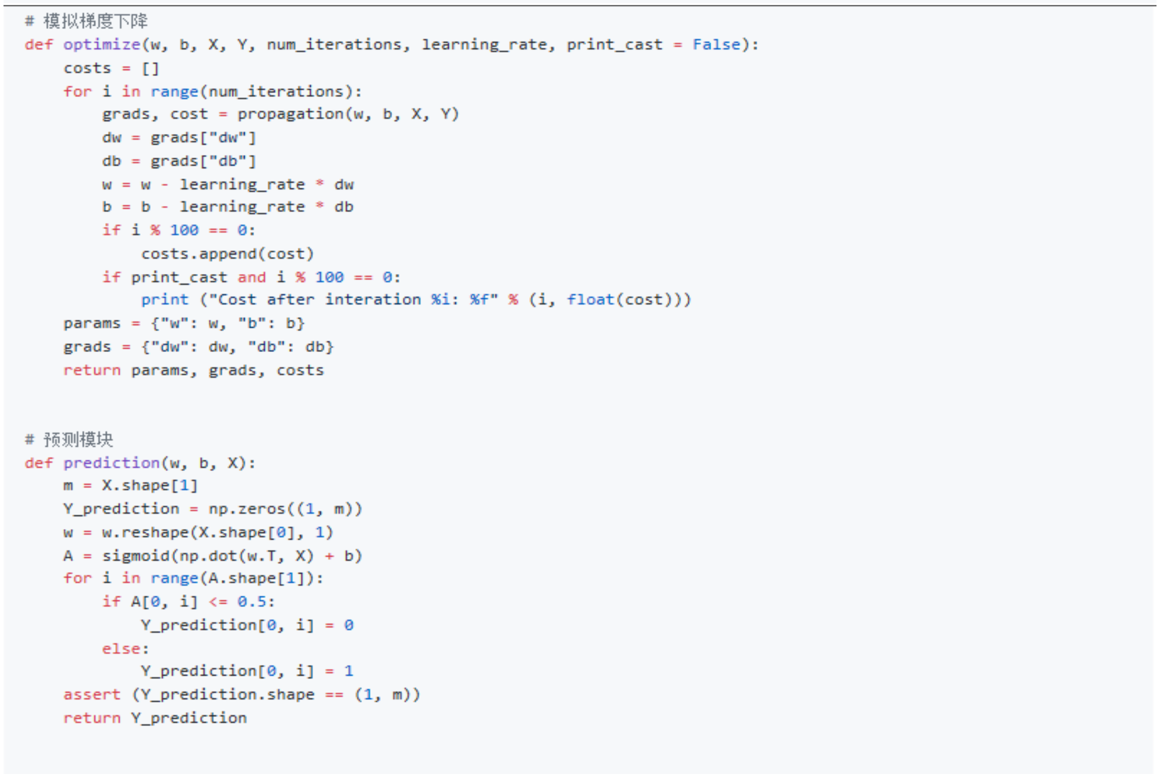
Using python languge build a logistic regression classifier to recogize cats with a Neural Network mind-set.Here is my python code:



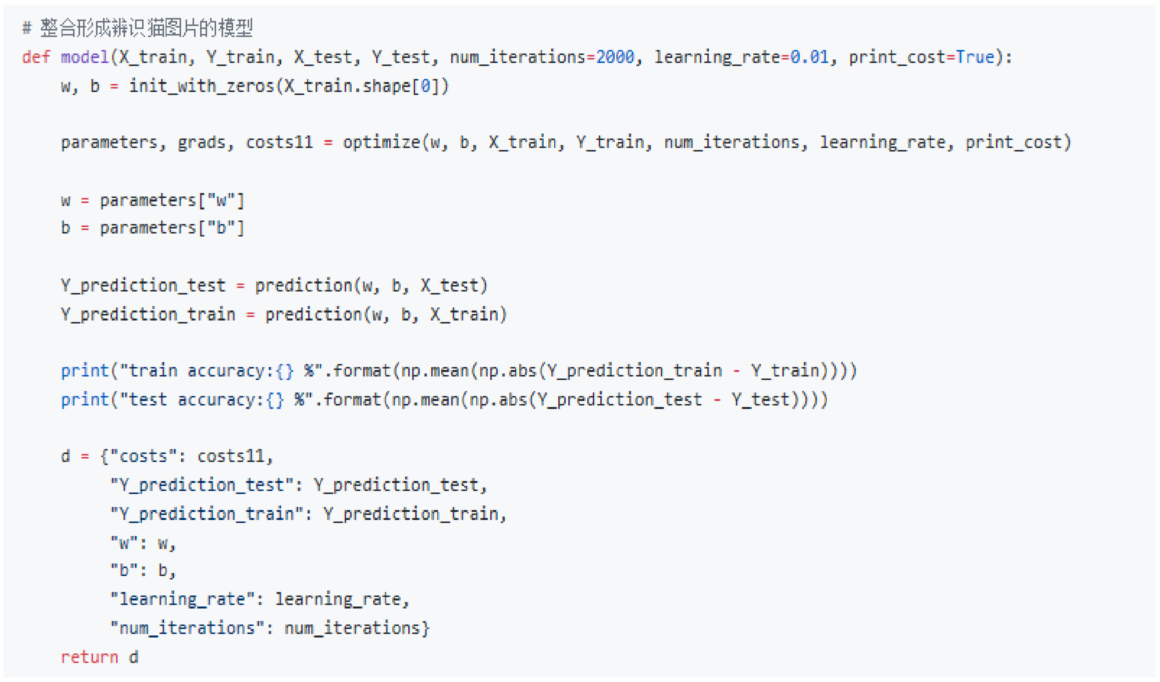
```python
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
from lr_utils import load_dataset#load数据集

# 引入数据集(209个训练样本，50个测试样本)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
# 获取训练样本及测试样本个数，图片像素
m_train = train_set_x_orig.shape[0]
m_test = test_set_x_orig.shape[0]
num_px = train_set_x_orig.shape[1]
# 将训练及测试样本转换成需要的矩阵{由（209,64,64,3）至（12288，209）}
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
# 标准化数据
train_set_x = train_set_x_flatten/255.
test_set_x = test_set_x_flatten/255.


# σ()函数
def sigmoid(z):
    return 1/(1+np.exp(-z))
```



```python
# 初始化权值w和阈值b
def init_with_zeros(dim):
    w = np.random.randn(dim, 1) * 0.01
    assert (w.shape == (dim, 1))
    b = 0
    assert (isinstance(b, float) or isinstance(b, int))
    return w, b


# 计算前向传播、反向传播参数
def propagation(w, b, X, Y):
    m = X.shape[1]
    A = sigmoid(np.dot(w.T, X) + b)
    lost = -(Y * np.log(A) + (1 - Y) * np.log(1 - A))
    cost = np.sum(lost)/m
    dz = A - Y
    db = np.sum(dz)/m
    dw = np.dot(X, dz.T)/m
    assert (dw.shape == w.shape)
    assert (db.dtype == float)
    cost = np.squeeze(cost)
    assert (cost.shape == ())
    grades = {"dw": dw, "db": db}
    return grades, cost
```



```python
# 模拟梯度下降
def optimize(w, b, X, Y, num_iterations, learning_rate, print_cast = False):
    costs = []
    for i in range(num_iterations):
        grads, cost = propagation(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_cast and i % 100 == 0:
            print ("Cost after interation %i: %f" % (i, float(cost)))
    params = {"w": w, "b": b}
    grads = {"dw": dw, "db": db}
    return params, grads, costs


# 预测模块
def prediction(w, b, 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)
    for i in range(A.shape[1]):
        if A[0, i] <= 0.5:
            Y_prediction[0, i] = 0
        else:
            Y_prediction[0, i] = 1
    assert (Y_prediction.shape == (1, m))
    return Y_prediction
```



```python
# 整合形成辨识猫图片的模型
def model(X_train, Y_train, X_test, Y_test, num_iterations=2000, learning_rate=0.01, print_cost=True):
    w, b = init_with_zeros(X_train.shape[0])

    parameters, grads, costs11 = optimize(w, b, X_train, Y_train, num_iterations, learning_rate, print_cost)

    w = parameters["w"]
    b = parameters["b"]

    Y_prediction_test = prediction(w, b, X_test)
    Y_prediction_train = prediction(w, b, X_train)

    print("train accuracy:{} %".format(np.mean(np.abs(Y_prediction_train - Y_train))))
    print("test accuracy:{} %".format(np.mean(np.abs(Y_prediction_test - Y_test))))

    d = {"costs": costs11,
         "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
```

From this encoding,I learn to build the general architeture of a learning algorithm,including:1.initializing parameters. 2.calculating the cost function and its gradient. 3.using an optimization algorithm.

1.2 One hidden layer Neural Network

1.2.1 Neural Network Representation

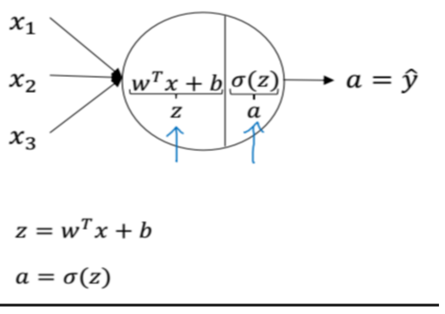


Figure 1: single neuron node

The main functions of the neuron node are calculating z and a,like Fig.1

$$z = w.Tx + b \tag{14}$$
$$a = \hat{y} = \sigma(z) \tag{15}$$

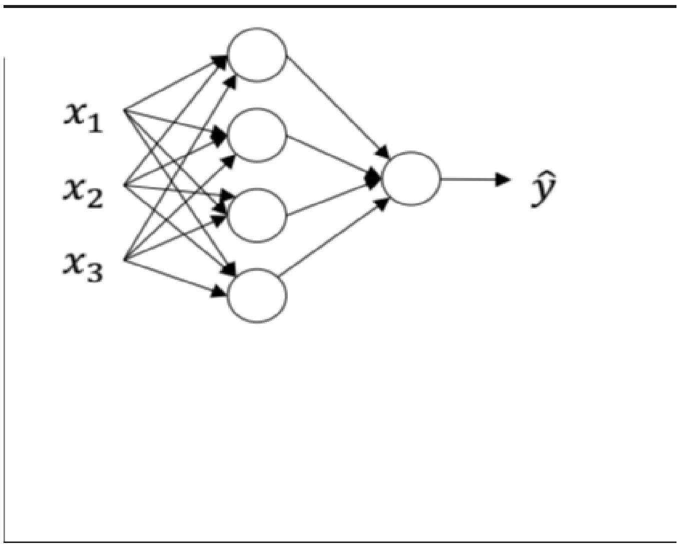


Figure 2: One hidden layer Neural Network

The Fig.2 is One hidden layer Neural Network.We can see that Neural Network is formed by a lot of single neuron stacking together.Neural Network can be divided into input layer,hidden layer and output layer,three sections.

1.2.2 vectorizing in one hidden neural network

We using one hidden neural nework as the Fig.2 show,and we assume firstly there is only one sample,as we have known the vectorizion of one neural note and one sample from session vectorizing Logistic Regression,and we also know neural network is stacking by lots of singal neuron,so we can get the results of vectorizion in forward propagation.

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ \cdots & w_3^{[1]T} & \cdots \\ \cdots & w_4^{[1]T} & \cdots \end{bmatrix}_{4\times 3} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3\times 1} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}_{4\times 1} \tag{16}$$

$$a^{[1]} = \hat{y^{[1]}} = g^{[1]} \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} \tag{17}$$

so after vectorizing:

$$z^{[1]} = W^{[1]}x + b^{[1]} \tag{18}$$
$$a^{[1]} = g^{[1]}(z^{[1]}) \tag{19}$$
$$z^{[2]} = W^{[2]}x + b^{[2]} \tag{20}$$
$$a^{[2]} = g^{[2]}(z^{[2]}) \tag{21}$$

$g^{[1]}()$ function is activate function,such as $\sigma()$,tanh(),ReLU,leaky ReLU and so on,as Fig.3 show:

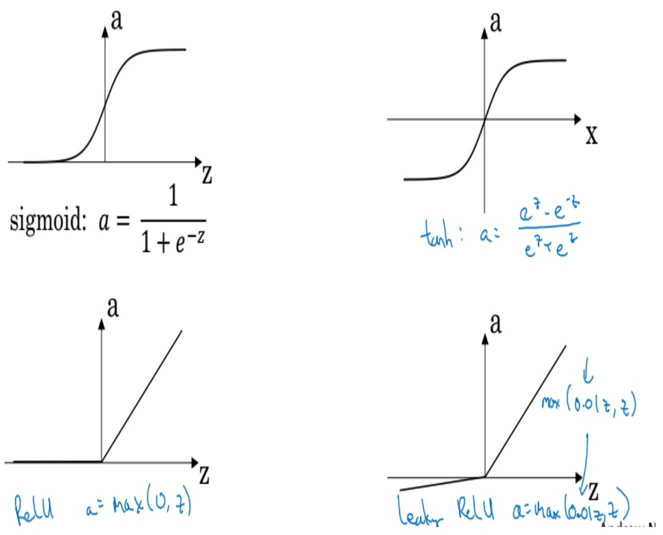


Figure 3: One hidden layer Neural Network

Form Fig.4,we can get the results of vectorizion in backword propagation:

$$dz^{[2]} = a^{[2]} - y \tag{22}$$
$$dW^{[2]} = dz^{[2]} a^{[1]T} \tag{23}$$
$$db^{[2]} = dz^{[2]} \tag{24}$$
$$dz^{[1]} = W^{[2]} dz^{[2]} * g^{[1]'}(z^{[1]}) \tag{25}$$
$$dW^{[1]} = dz^{[1]} x^T \tag{26}$$
$$db^{[1]} = dz^{[1]} \tag{27}$$

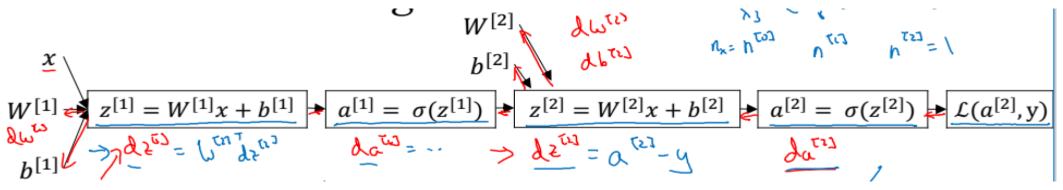

x
$W^{[1]}$
$dW^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$dz^{[1]} = W^{[2]T} dz^{[2]}$
$a^{[1]} = \sigma(z^{[1]})$
$da^{[1]} = \dots$
$W^{[2]}$
$b^{[2]}$
$dW^{[2]}$
$db^{[2]}$
$z^{[2]} = W^{[2]}x + b^{[2]}$
$dz^{[2]} = a^{[2]} - y$
$n_x = n^{[0]}$
$n^{[1]}$
$n^{[2]} = 1$
$a^{[2]} = \sigma(z^{[2]})$
$da^{[2]}$
$\mathcal{L}(a^{[2]}, y)$

Figure 4: One hidden layer Neural Network

Vectorizing across m examples:

$$\begin{aligned} Z^{[1]} &= W^{[1]} X + b^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \\ Z^{[2]} &= W^{[2]} X + b^{[2]} \\ A^{[2]} &= g^{[2]}(Z^{[2]}) \end{aligned} \tag{28}$$

$$Z^{[1]} = W^{[1]} X + b^{[1]} \tag{29}$$
$$A^{[1]} = g^{[1]}(Z^{[1]}) \tag{30}$$
$$Z^{[2]} = W^{[2]} X + b^{[2]} \tag{31}$$
$$A^{[2]} = g^{[2]}(Z^{[2]}) \tag{32}$$

$$dZ^{[2]} = A^{[2]} - Y \tag{33}$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T} \tag{34}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True) \tag{35}$$

$$dz^{[1]} = W^{[2]} dZ^{[2]} * g^{[1]'}(Z^{[1]}) \tag{36}$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T \tag{37}$$

$$db^{[1]} = dZ^{[1]} \tag{38}$$

Finally,When we train neural network, it is important to initialize the weights randomly. Main reasion is that all neuron node will be become linear calculation when the weights are same,that means the result of millions of neuron node'work is same as one neuron node. Also,hidden layer's active function need use un-linear function ,such as ReLU,tanh except sigmoid function.

2 Progress in this week

In this week,my main job is installing,configuring, operating linux system,github,latex,markdown editer and so on;learning python encoding and deep learning ;also encoding a small identifing cats picture model using python.

Step 1 learning linux,github,latex,markdown

Step 2 Learning python and encoding on leetcode

Step 3 Learning the second and third week course of neural networks and deep learning

3 Plan

Objective: Finish deep learning course **Deadline:** 2018-8-18

2018.07.23—2018.07.29 Finish the fourth class of deep neural networks courses and improving deep neural networks courses.

2018.07.302018.08.05 Finish structuring machine learning projects courses.

2018.08.062018.08.12 Finish convolutional neural networks courses.

2018.08.132018.08.18 Finish sequence models courses.

References