Study Report

Shuo Xu

July 16, 2018

Attempts on neural networks

After completing the first three weeks of cuosera, I implemented a two-layer neural network that mimicked the code of the exercises, with a recognition rate of 98
Since the code for the exercises is perfect enough, I actually paid less effort. But if I write it myself, I don't think I can do modular programming. I should learn this part of the code.

My code

```python
#!/usr/bin/python
#coding=utf-8

import numpy as np
def sigmoid(x,flag=False):
    if(flag):
        return x*(1-x)
    return 1/(1+np.exp(-x))

def load_dataset(name = 'UCI.txt'):
    file = open(name)
    lines = file.readlines()
    rows = len(lines)
    X = np.zeros((30, rows))
    Y = np.zeros((1, rows))

    row = 0
    for line in lines:
        line = line.split('\t')
        if(line[0] == 'M'):
            Y[:,row] = 1
        else:
            Y[:,row] = 0

        X[:,row] = line[1:]
        row += 1

    return X,Y

def layer_sizes(X, Y):
    n_x = len(X)
    n_y = len(Y)
    return (n_x, n_y)

def initialize_parameters(n_x, n_y, n_h):
    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

```

```python
def forward_propagation(X, parameters):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1,X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2,A1) + b2
    A2 = sigmoid(Z2)    # tanh & sigmoid function can be used

    #assert(A2.shape == (1, X.shape[1]))
    #should use "assert", but didn't think of it myself

    # cache:
    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2}

    return A2, cache

def compute_cost(A2, Y, parameters):
    log = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), 1 - Y)
    # multiply is very import
    cost = -np.sum(log) / Y.shape[1]
    cost = np.squeeze(cost)

    #assert(isinstance(cost, float))

    return cost

def backward_propagation(parameters, cache, X, Y):
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    A1 = cache["A1"]
    A2 = cache["A2"]

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2,A1.T) / Y.shape[1]
    db2 = np.sum(dZ2,axis = 1,keepdims = True) / Y.shape[1]
    dZ1 = np.multiply(np.dot(W2.T, dZ2) , 1 - np.power(A1, 2))
    dW1 = np.dot(dZ1, X.T) / Y.shape[1]
    db1 = np.sum(dZ1, axis = 1, keepdims = True) / Y.shape[1]
    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2}

    return grads

def update_parameters(parameters, grads, learning_rate):
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']
    dW1 = grads['dW1']
    db1 = grads['db1']
```

```python
    dW2 = grads['dW2']
    db2 = grads['db2']

    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters

def nn_model(X, Y, n_h,learning_rate, num_iterations = 10000 , print_cost=False):
     np.random.seed(1)
     n_x, n_y = layer_sizes(X, Y)

     parameters = initialize_parameters(n_x, n_y, n_h)
     W1 = parameters['W1']
     b1 = parameters['b1']
     W2 = parameters['W2']
     b2 = parameters['b2']

     for i in range(0, num_iterations):
         A2, cache = forward_propagation(X, parameters)
         cost = compute_cost(A2, Y, parameters)
         grads = backward_propagation(parameters, cache, X, Y)
         parameters = update_parameters(parameters, grads, learning_rate)

         if print_cost and i % 1000 == 0:
             print ("Cost after iteration %i: %f" %(i, cost))

     return parameters

def predict(parameters, X):
    A2, cache = forward_propagation(X, parameters)
    predictions = np.array( [1 if x >0.5 else 0 for x in A2.reshape(-1,1)] ).reshape(A2.shape)
    return predictions

#name = input("input the file name for training:")
X,Y = load_dataset('UCI.txt')
parameters = nn_model(X, Y, 4, 1.5, 500,  True)
#name = input("input the file name for predict:")
X,Y = load_dataset('UCI_test.txt')
predictions = predict(parameters, X)
print ('Accuracy: %d' % float((np.dot(Y,predictions.T) + np.dot(1-Y,1-predictions.T))/float(Y.size)*100) + '%')
```

I still don't know some details in the code. And I don't have a good way to determine the learning rate and the number of loops, but the method of determining the learning rate will be mentioned in the following video.

Leetcode

Palindrome Linked List

Description

Given a singly linked list, determine if it is a palindrome.
Example 1:
Input: 1->2 Output: false
Example 2:
Input: 1->2->2->1 Output: true

Solution

The problem is not difficult, create a vector and store the elements of the linked list into it, and then judge whether the corresponding elements are the same from the front and back ends of the vector, move the iterator to the middle at the same time

code:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        if(!head)
            return true;
        vector<int> line;
        while(head){
            line.push_back(head -> val);
            head = head -> next;
        }
        auto l = line.begin(), r = line.end()-1;
        while(l < r){
            if(*l != *r)
                return false;
            l++; r--;
        }
        return true;
    }
};
```

Summary And Thinking

Maybe I should watch the deeplearning video on cousera earlier.