# Ecperiment4: Neural Network Based on Numpy

Time: 2022/3/29
Location: Science_Building_119
Name: 易弘睿
Number: 20186103

## Part I: Review

MNIST里包含各种手写数字图片以及每张图片对应的标签。每张图片都经过了大小归一化和居中处理。处理后的数据是一个单通道的黑白图片。首先对网络中的参数做初始化，然后解压训练集的标签和图像，再对训练集中的数据执行100次网络的训练，接着解压测试集的标签和图像，最后使用测试集数据查看网络的学习效果，并打印网络在测试集上的正确率。

## Part II: Introduction

对初始代码的修改主要如下：
1. 对代码按照不同部分功能进行命名；
2. 对代码进行每行注释；
3. 对代码进行调整和优化。

## Part III: Annotation

### 1. 数据库的导入

In [1]:

```
import struct      # 导入struct库，用于数据的解压
import numpy as np # 导入numpy库
```

### 2. 参数及函数的定义

In [2]:

```
# 定义网络的学习率为0.001
learn_rate = 0.001
```

In [3]:

```
def get_data():                                                    # 此函数用于训练数据的解压，返回one-hot
    # 导入训练图像的标签
    with open('train-labels.idx1-ubyte', 'rb') as lbpath:
        # 解压数据
        magic, n = struct.unpack('>II', lbpath.read(8))
        # 获取标签，如果第一个标签为5则表示第一张照片中的数字为5
        labels = np.fromfile(lbpath, dtype=np.uint8)

    # 导入训练图像
    with open('train-images.idx3-ubyte', 'rb') as imgpath:
        # 解压图像
        magic, num, rows, cols = struct.unpack('>IIII', imgpath.read(16))
        # 获得图像并将照片的形状从28*28调整为784
        images = np.fromfile(imgpath, dtype=np.uint8).reshape(len(labels), 784)

    # 对标签做one-hot编码
    # 如果第一张图的标签为5，则他的one-hot编码为[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
    # 初始化标签编码均为0
    list_labels = np.zeros([60000, 10])
    for index in range(60000):
        # 完成第index个数据的标签编码
        list_labels[index][labels[index]] = 1

    # 返回one-hot编码的训练标签和训练图像
    return list_labels, images
```

In [4]:

```
def parameter_initialization():                                    # 此函数用于网络中的参数初始化，返回两层网

    # 由于网络中使用了sigmod函数作为激活函数，所以对权重做了一个0.001倍的缩小
    # 由于第一层网络输入的feature_size为784并且希望将feature_size调整为60
    # 所以，第一层网络的weights的形状为[60, 784]，bias的形状为[60, 1]
    w1 = 0.001*np.random.rand(60, 784)
    b1 = 0.001*np.random.randn(60, 1)

    # 由于第二层网络的输入的feature_size为60并且希望将feature_size调整为10
    # 所以，第二层网络的weights的形状为[10, 60]，bias的形状为[10, 1]
    w2 = 0.001*np.random.rand(10, 60)
    b2 = 0.001*np.random.randn(10, 1)

    # 返回两层网络的weights和bias
    return w1, w2, b1, b2
```

In [5]:

```
def sigmoid(z):
    # 此函数用于制作sigmod函数
    return 1 / (1 + np.exp(-z))
```

In [6]:

```
def relu(x):                                                       # 此函数用于制作relu函数
    return np.maximum(0,x)
```

In [7]:

```
def relu_backward(next_dz,z):                                    # 此函数用于反向更新参数时，relu函数的更新
    return np.where(np.greater(z, 0), next_dz, 0)
```

In [8]:

```
def buildmode(images, labels, w1, w2, b1, b2):                   # 此函数为主函数，执行训练过程

    # 对输入的images和labels做转置的操作
    images = images.T
    labels = labels.T
    # 定义一个batch的数量为250
    batch_size = 250

    # 对每个batch做前向传递的操作
    for batch in range(int(images.shape[-1]/batch_size)):
        # 寻找每个batch的初始index
        start = batch*batch_size
        # 分割出该batch的images数据和标签
        batchImage = images[:,start:start+batch_size]
        batchlabel = labels[:,start:start+batch_size]
        # 数据过第一层网络，调整形状为[60, 250]
        z1 = np.dot(w1,batchImage) + b1
        # 数据过relu激活函数
        a1 = relu(z1)
        # 数据过第二层网络，调整形状为[10, 250]
        z2 = np.dot(w2, a1)+b2
        # 数据过sigmod激活函数
        a2 = sigmoid(z2)
        # 使用交叉熵作为损失函数，计算该batch的损失值
        loss = -batchlabel*np.log(a2)
        # 计算真实值和标签的差距，便于后续的参数更新
        dz2 = a2 - batchlabel
        # 由于这是一整个batch的梯度，所以下面所有的梯度都做了一个除batch_size的操作
        # 根据公式$x(\hat{y}-y)$，计算第二层weights的梯度
        dw2 = np.dot(dz2,a1.T)/batch_size
        # 根据公式$(\hat{y}-y)$，计算第二层bias的梯度
        db2 = np.sum(dz2,axis=1,keepdims=True)/batch_size
        # 类似于第二层网络的参数更新，计算第一层网络中的weights和bias的梯度
        da1 = np.dot(w2.T,dz2)
        # 由于是relu函数，所以当数值大于0时，梯度保持不变，数据小于等于0时，梯度为0
        dz1 = relu_backward(da1,z1)
        dw1 = np.dot(dz1,batchImage.T)/batch_size
        db1 = np.sum(dz1,axis=1,keepdims=True)/batch_size

        # 根据先前制定好的学习率和计算出来的梯度执行反向更新的操作
        w1 = w1 - learn_rate * dw1
        w2 = w2 - learn_rate * dw2
        b1 = b1 - learn_rate * db1
        b2 = b2 - learn_rate * db2

        # 每过10个batch就执行一次数据的打印，反馈网络当前的损失值
        if batch % 10 == 0:
            print('第{a}batch训练的当前的loss值为{b}'.format(a=batch, b=np.sum(loss)/batch_size))

    # 返回经过1个episode训练后的网络参数
    return w1, w2,  b1, b2
```

In [9]:

```
def get_test_data():                                    # 此函数用于测试数据的解压，返回one-hot
    # 导入测试图像的标签
    with open('t10k-labels.idx1-ubyte', 'rb') as lbpath2:
        # 解压标签数据
        magic, n = struct.unpack('>II', lbpath2.read(8))
        # 获取标签，如果第一个标签为5则表示第一张照片中的数字为5
        labels_test = np.fromfile(lbpath2, dtype=np.uint8)

    # 导入测试图像
    with open('t10k-images.idx3-ubyte', 'rb') as imgpath2:
        # 解压图像数据
        magic, num, rows, cols = struct.unpack('>IIII', imgpath2.read(16))
        # 获得图像并将照片的形状从28*28调整为784
        images_test = np.fromfile(imgpath2, dtype=np.uint8).reshape(len(labels_test), 784)

    # 对标签做one-hot编码
    # 如果第一张图的标签为5，则他的one-hot编码为[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
    # 初始化标签编码均为0
    list_labels_test = np.zeros([10000, 10])
    for index in range(10000):
        # 完成第index个数据的标签编码
        list_labels_test[index][labels_test[index]] = 1

    return list_labels_test, images_test
```

In [10]:

```
def test(labels_test, images_test, w1, w2, b1, b2):          # 此函数用于测试测试集中的识别准确率
    # 定义变量truesample，统计训练中识别正确的图像
    truesample = 0
    for i in range(len(images_test)):
        # 调整测试数据的形状
        image = images_test[i].reshape(784,-1)
        label = labels_test[i].reshape(10,-1)
        # 过第一层网络
        a1 = relu(np.dot(w1, image)+b1)
        # 过第二层网络
        a2 = sigmoid(np.dot(w2, a1)+b2)
        # 选择概率最高的作为网络给出的判断
        # 比如网络的输出为[0.7, 0.2, 0.1]，那么网络会输出标签为0
        # 如果网络的判断与标签一致，则truesample的数量加1
        if np.argmax(a2) == np.argmax(label):
            truesample = truesample + 1
        # 计算测试集中正确的sample的概率
    return truesample / len(labels_test)
```

### 3. 网络的训练及测试

In [11]:

```
# 对网络中的参数做初始化
w1, w2, b1, b2 = parameter_initialization()
# 解压训练集的标签和图像
labels, images = get_data()
# 对训练集中的数据执行100次网络的训练
for i1 in range(100):
    w1, w2, b1, b2 = buildmode(images, labels, w1, w2, b1, b2)
# 解压测试集的标签和图像
labels_test, images_test = get_test_data()
# 使用测试集数据查看网络的学习效果
last = test(labels_test, images_test, w1, w2, b1, b2)
# 打印网络在测试集上的正确率
print('测试正确率为{c}'.format(c=last))
```

```
第60batch训练的当前的loss值为0.02363675843952702
第70batch训练的当前的loss值为0.013059099936823951
第80batch训练的当前的loss值为0.026610237302743023
第90batch训练的当前的loss值为0.017200771902023695
第100batch训练的当前的loss值为0.02340397489400452
第110batch训练的当前的loss值为0.03588560710752591
第120batch训练的当前的loss值为0.01155212579402571
第130batch训练的当前的loss值为0.021417993104271768
第140batch训练的当前的loss值为0.020523558256231464
第150batch训练的当前的loss值为0.0167508383484445
第160batch训练的当前的loss值为0.01064895155588577
第170batch训练的当前的loss值为0.03908485792970039
第180batch训练的当前的loss值为0.03273145197030868
第190batch训练的当前的loss值为0.027524903431915725
第200batch训练的当前的loss值为0.017340440240899688
第210batch训练的当前的loss值为0.02494703914309521
第220batch训练的当前的loss值为0.017514437735175966
第230batch训练的当前的loss值为0.019599387687792662
测试正确率为0.9754
```