模式识别与统计学习实验七:CNN卷积神经网络实验报告

郭宇航 2016100104014

一　实验原理

1.1　CNN模型概述

前一次实验进行了基本神经网络模型(DNN)的搭建，最基本的神经网络是全连接的网络，在此基础上如果有意识的去掉一些边的权重，变为非全连接网络，其中一种很重要的类别就是Convolutional Neural Network（卷积神经网络），这种网络是基于全连接网络改进而来，广泛应用于图片识别，NLP等领域，主要原因就在于其在提取图片的特征上有很好的表现。通过一张图我们可以看到一个CNN的基本构造（图1）。 CNN网络模型的输入为图片构成的像素点矩阵，对此矩阵进行多通道

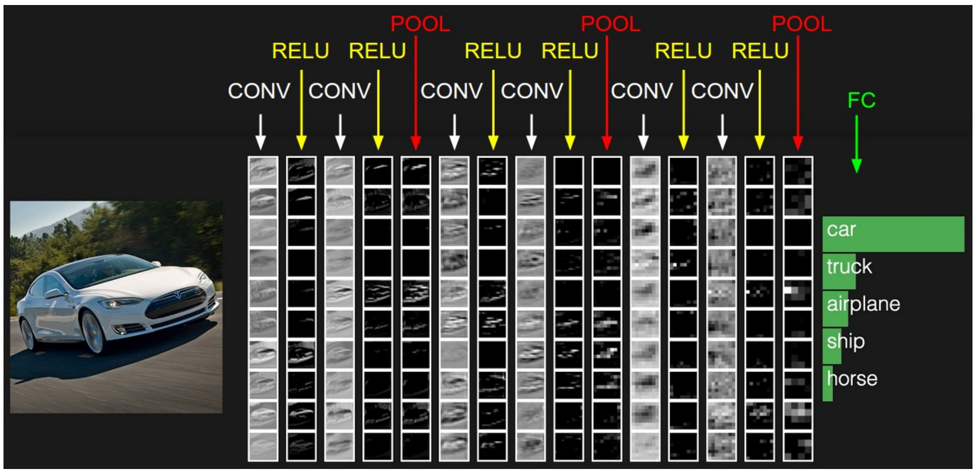


图 1: CNN网络基本构造图

的卷积操作并通过激活函数激活即可以得到我们的第一层的FeatureMap，第二层对FeatureMap进行池化(pooling)，最后将提取的特征flatten为列向量输入一个全连接层网络，最终得到图片的输出分类。上述过程即为卷积神经网络的实现过程。下面给出每一层的前向后向传播过程。为了更好的表示这些过程，现在对其中使用的符号进行定义：

- w_{jk}^l表示从$(l-1)$层的第k个神经元到l层的第j个神经元的连接权重。
- b_j^l表示第l层的第j个神经元的偏置。
- z_j^l表示第l层的第j个神经元的带权输入。
- a_j^l表示第l层的第j个神经元的激活值。
- $\sigma()$表示激活函数：$a^l = \sigma(z^l)$.

1.2 CNN前向传播

首先给出一个简单的卷积网络模型（图2）并基于此模型讨论网络的前向传播过程。

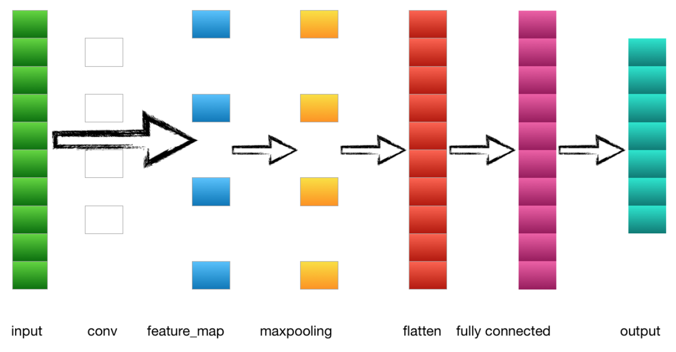


图 2: 简单的三层卷积网络模型

1.2.1 卷积层

假设以一张图片为例，卷积层的输入为图片构成的像素点矩阵，共有k个卷积核，对图片和卷积核分别进行卷积操作，卷积操作的原理(以二维卷积为例)表示为：

$$s(i,j)=(X*W)(i,j)=\sum_m\sum_n x(i+m,j+n)w(m,n) \tag{1}$$

其中$*$表示对应的点乘运算，W表示卷积核，X表示一张输入的图片。这里需要注意一些细节，假设输入图片的尺寸为$m\times m$，k个尺寸为$n\times n$的卷积核，假设希望保证图片边缘的能够被卷积核提取，可以考虑在输入周围添加一圈0，记padding的层数为a，同时卷积每一移动的像素距离(stride)为s，我们可以计算得到输出的feature map的尺寸为：

$$output_{size}=(m-n+2a)/s+1 \tag{2}$$

共计k个尺寸为$output_{size}$的feature map.通过下图3可以更好的直观理解：

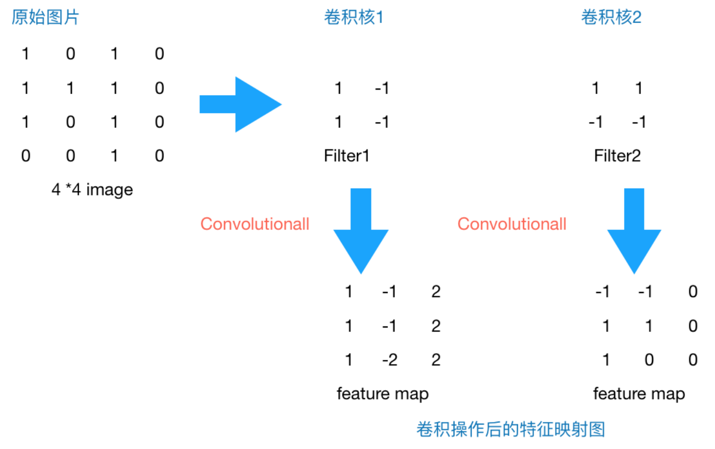


图 3: CNN卷积层前向传播

进一步可以一般表示为：

$$a^l = \sigma(z^l) = \sigma(\sum_{k=1}^{M}) = \sigma(\sum_{k=1}^{M} a_k^{l-1} * W_k^l + b^l) \tag{3}$$

1.2.2 池化层

关于池化层的作用，表面上看来池化是为了压缩feature map的尺寸，但事实上pooling的作用最本质上在于对特征的进一步的提取和筛选，卷积层表面看上去仅仅是在做简单的运算，但事实上是在进行图片的局部信息特征的静态性的提取，所谓静态性的特征提取主要是指可能由于图片的尺寸，图片内部物体的分布的问题，导致在面对同一个事物的时候，在像素点上直观表示出来的不太一样，因此利用平移不变性，使用卷积可以把图片间共同的形状，轨迹等低频和高频的轮廓进行提取。回到pooling，池化很大程度上就是在对卷积出来的特征最筛选，减少一下局部极其微小的无关参数的影响，增强一些主要特征的识别聚合度。pooling的主要类别有最大池化和平均池化两种，这里以最大池化为例—(图4)： 池化层的处理逻辑上比较简单，对于输入的feature map进行缩小化，

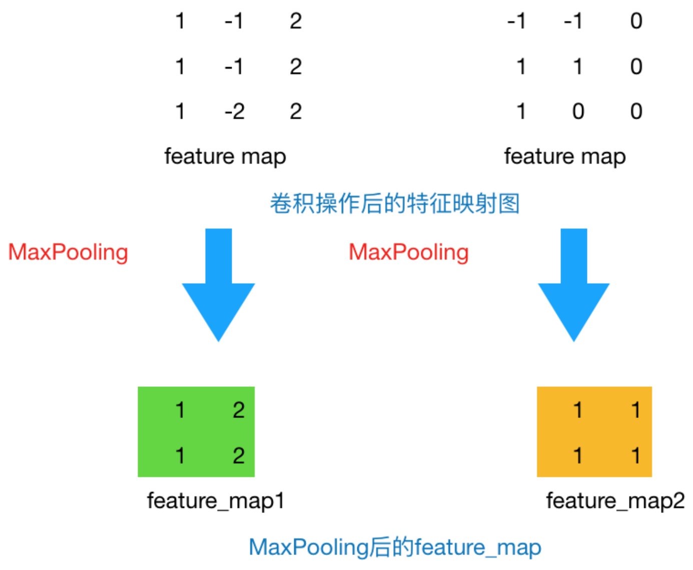


图 4: CNN池化层前向传播

假设输入的feature map尺寸为$k \times N \times N$，池化区域的大小为$p \times p$，池化操作为在这个$p \times p$的区域内选取一个最大的值则通过池化层得到的输出矩阵维度为$k \times \frac{N}{k} \times \frac{N}{k}$.

1.2.3 全连接层

全连接层的模型构造即为普通DNN模型的结构，前向传播的结果可以简单的表示为：

$$a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l) \tag{4}$$

这里可以加多层的全连接，最后一层为数量为输出的类别标签的数量，输出层可以考虑使用一层softmax函数进行激活。

1.3 CNN反向传播

回顾一下DNN的后向传播算法，我们有这样四个表达式：

$$\begin{cases} \delta^L = \nabla_{a^L} C \odot \sigma'(z^L) \\ \delta^l = (W^{l+1})^T \delta^{l+1} \odot \sigma'(z^l) \\ \nabla_{b^l} C = \delta^l \\ \nabla_{W^l} C = a^{l-1}\delta^l \end{cases} \tag{5}$$

而考虑CNN的反向传播时，其基本思想与DNN一致，只有在卷积层和池化层的误差求解的过程中，由于前向传播的方式不同于全连接层，因此其反向传播的误差计算也有一些差别，下面分别每一层对应的反向传播的误差和对权重以及偏置求导的详细说明。

1.3.1 由第一层全连接到池化层的反向传播误差

由于全连接层的反向传播函数在DNN中已经给出了解释这里就不加以赘述，下面主要论述从全连接到池化层的误差的求解问题，假设现在求解得到了第一层全连接的误差值，记做δ^{l+1}，则传至池化层的时候的误差应该表示为：

$$\delta^l = (W^{l+1})^T \delta^{l+1} \odot \sigma'(z^l) = (W^{l+1})^T \delta^{l+1} \odot I \tag{6}$$

由于池化层到全连接层之间是直接将特征矩阵flatten，没有经过激活函数，因此激活函数可以写为单位矩阵与$(W^{l+1})^T \delta^{l+1}$进行对应位置点乘，然后进行flatten的反向操作，将这个误差reshape为池化层正向传播的输出值的尺寸。

1.3.2 由池化层到卷积层的反向传播误差

想要求从池化层到卷积层的反向传播误差，首先理解前向传播的原理，池化进行压缩尺寸压缩数据的操作，实际上进行的是下采样(downsampling)那么反向传播的时候需要进行的是下采样的反向操作上采样(upsampling)以一个简单的例子作为说明，假设池化区域大小为2×2，δ^l的第a个矩阵为：

$$\begin{bmatrix} 2 & 8 \\ 4 & 6 \end{bmatrix}$$

若为max_pooling操作，则在进行前向传播时需要记录下每个池化区域取最大值的位置，假设这里的四个为分别为左上，左下，右上和右下，则在进行upsampling还原时，我们得到的矩阵可以写为：

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

如果进行的是average_pooling，则矩阵可以写为：

$$\begin{bmatrix} 0.5 & 0.5 & 2 & 2 \\ 0.5 & 0.5 & 2 & 2 \\ 1 & 1 & 1.5 & 1.5 \\ 1 & 1 & 1.5 & 1.5 \end{bmatrix}$$

对每个通道都进行upsampling，我们就可以计算卷积层的误差值，其中upsampling完成了池化误差矩阵放大与误差重新分配的逻辑。对于上一层卷积的误差可以写为：

$$\delta^{l-1} = \text{upsampling}(\delta^l) \odot \sigma'(z^{l-1}) \tag{7}$$

1.3.3 多层卷积的反向传播误差

假设CNN网络进行了多次的卷积操作，则对于卷积层来说，如果卷积层的误差为δ^l，计算卷积上一层的误差。仍然使用一个简单的例子：假设$l-1$层的输出是3×3的矩阵，第l层的卷积核为2×2的矩阵，stride为1，则输出z^l是一个2×2的矩阵，b^l对中间的计算没有影响暂时不考虑，基于上述的假设则有：

$$a^{l-1} \odot W^l = z^l \tag{8}$$

进一步列出具体的表达式：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \odot \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$$

由卷积的定义很容易得到：

$$\begin{aligned} z_{11} &= a_{11}w_{11} + a_{12}w_{12} + a_{21}w_{21} + a_{22}w_{22} \\ z_{12} &= a_{12}w_{11} + a_{13}w_{12} + a_{22}w_{21} + a_{23}w_{22} \\ z_{21} &= a_{21}w_{11} + a_{22}w_{12} + a_{31}w_{21} + a_{32}w_{22} \\ z_{22} &= a_{22}w_{11} + a_{23}w_{12} + a_{32}w_{21} + a_{33}w_{22} \end{aligned} \tag{9}$$

接着模拟反向的求导过程：

$$\nabla a^{l-1} = \frac{\partial C}{\partial a^{l-1}} = (\frac{\partial z^l}{\partial a^{l-1}})^T \frac{\partial C}{\partial z^l} = (\frac{\partial z^l}{\partial a^{l-1}}^{T})\delta^l \tag{10}$$

假设反向传播得到的误差组成的矩阵为：

$$\begin{bmatrix} \delta_{11} & \delta_{12} \\ \delta_{21} & \delta_{22} \end{bmatrix}$$

则对每个∇a^{l-1}中的标量的梯度可以使用另一种反向的卷积的形式的表示：

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & 0 \\ 0 & \delta_{21} & \delta_{22} & 0 \end{bmatrix} * \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix} = \begin{bmatrix} \nabla a_{11} & \nabla a_{12} & \nabla a_{13} \\ \nabla a_{21} & \nabla a_{22} & \nabla a_{23} \\ \nabla a_{31} & \nabla a_{32} & \nabla a_{33} \end{bmatrix}$$

因此写成矩阵的形式可以表示为：

$$\delta^{l-1} = \delta^l * \text{rot180}(W^l) \odot \sigma'(z^{l-1}) \tag{11}$$

对比全连接层的误差递推公式会发现有非常相似的地方：

$$\delta^l = (W^{l+1})^T \delta^{l+1} \odot \sigma'(z^l) \tag{12}$$

1.3.4 由卷积层误差求解梯度值

对于池化层，其只进行了简单的下采样过程，不需要求解权重和偏置的梯度，全连接层求解的方法已经在前面的DNN中给出了这里也不再加以赘述。主要讨论卷积层的梯度的求解方法。

前向卷积有：

$$z^l = a^{l-1} * W^l + b \tag{13}$$

对权重W^l求偏导数不难得到：

$$\frac{\partial C}{\partial W^l} = a^{l-1} * \delta^l = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} * \begin{bmatrix} \delta_{11} & \delta_{12} \\ \delta_{21} & \delta_{22} \end{bmatrix} \tag{14}$$

而对于偏置的求导可以表示为：

$$\frac{\partial C}{\partial b^l} = \text{average}(\delta^l) \tag{15}$$

二 Python代码实现

仍然使用MNIST数据集，本次实验的网络的基本构造如下图5： 模型一共为五层，第一层为

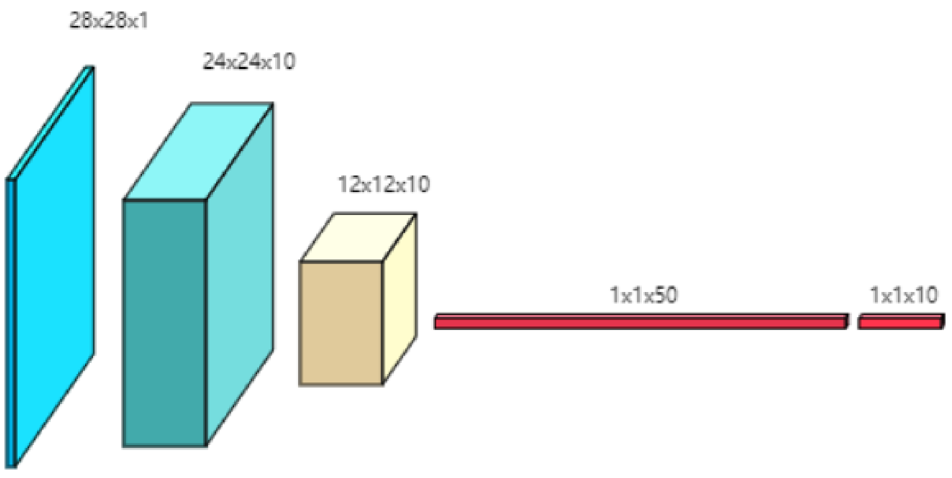


图 5: CNN真实实验模型

输入层，输入图片尺寸为：28 × 28，第二层为卷积层，10个尺寸为5 × 5的卷积核，周围未补零，stride为1，卷积输出为10×24×24的feature map，池化层的池化区域为2×2，池化输出为10×12×12，flatten为1 × 1440的列向量，作为输入传入第四层全连接层，第四层全连接层50个神经元，最后的输出层为10个神经元。

2.1 定义CNN中三层的类

CNN网络中共涉及三类不同的层，卷积层，池化层，全连接层。为了方便后续函数的编程，这边将三层不同的网络封装为三个类，代码如下：

```
import numpy as np

class input_layer:
    def __init__(self,data_size,batch_size):
        self.output_size = [batch_size,data_size,data_size]
        self.input = None
        self.a_value = None
        self.layer_kind = 'input'

class conv_2d_layer:
    def __init__(self,input,filter_size,depth,zero_padding,stride,activation):
        self.input = None
        self.size = filter_size
        self.depth = depth
        self.zero_padding = zero_padding
        self.stride = stride
        self.activation = activation
        self.weights = np.random.rand(depth,filter_size,filter_size)
        self.bias = np.random.rand(depth,1)
        self.output_size = [input.output_size[0],self.depth,
                        int((input.output_size[1] - filter_size + 2*zero_padding)/stride + 1),
                        int((input.output_size[2] - filter_size + 2*zero_padding)/stride + 1)]
        self.error = None
        self.z_value = None
        self.a_value = None
        self.delta_weights = None
        self.layer_kind = 'conv'

class pooling_layer:
    def __init__(self,input,pooling_size,pooling_type):
        self.input = None
        self.size = pooling_size
        self.output_size = [input.output_size[0],input.output_size[1],
                            int(input.output_size[2]/pooling_size),
                            int(input.output_size[3]/pooling_size)]
        self.up_sampling = None
        self.z_value = None
        self.a_value = None
        self.type = pooling_type
        self.layer_kind = 'pool'

class full_connected_layer:
    def __init__(self,input,layer_size,activation):
        if input.layer_kind == 'conv' or input.layer_kind == 'pool':
            self.size = [layer_size,input.output_size[1]*input.output_size[2]*input.
                                                    output_size[3]]
        elif input.layer_kind == 'full':
            self.size = [layer_size,input.size[0]]
        self.weights =  np.random.randn(self.size[0],self.size[1])/10
        self.bias = np.random.randn(layer_size,1)/10
        self.input = None
        self.activation = activation
        self.z_value = None
        self.a_value = None
        self.error = None
        self.delta_weights = None
        self.layer_kind = 'full'
```

2.2 激活函数代码

沿用DNN使用到的几种激活函数：Sigmoid，RELU，Softmax.代码如下：

```
def Sigmoid_forward(weighted_input):
    return 1.0 / (1.0 + np.exp(-weighted_input))

def Sigmoid_backward(output_value):
    return np.multiply(output_value,(1-output_value))

def RELU_forward(weighted_input):
    return weighted_input * (weighted_input >= 0)

def RELU_backward(output_value):
    output_value[output_value>0] = 1
    output_value[output_value<=0] = 0
    return output_value

def softmax_forward(H_x):
    for i in H_x:
        i = np.exp(i) / np.sum(np.exp(H_x))
    return H_x
```

2.3 前向传播代码

```
import numpy as np
import random
import time
import matplotlib.pyplot as plt
import CNN_class
import activation_function
import copy

def conv_2d(conv_layer):
    conv_length = int((conv_layer.input.shape[1] - conv_layer.size + 2*conv_layer.zero_padding
                                                  ) / conv_layer.stride + 1)
    conv_width = int((conv_layer.input.shape[2] - conv_layer.size + 2*conv_layer.zero_padding)
                                                   / conv_layer.stride + 1)
    feature_map = np.zeros((conv_layer.input.shape[0],conv_layer.depth,conv_length,conv_width)
                                                  )
    for num in range(conv_layer.input.shape[0]):
        for j in range(conv_layer.depth):
            for k in range(conv_length):
                for m in range(conv_width):
                    feature_map[num,j,k,m] = np.sum(np.multiply(conv_layer.input[num,k:k+
                                                                conv_layer.size,m:m+
                                                                conv_layer.size],conv_layer
                                                                .weights[j]))
                                                  + conv_layer.bias[j,0]
    conv_layer.z_value = feature_map
    if conv_layer.activation == 'Sigmoid':
        conv_layer.a_value = activation_function.Sigmoid_forward(conv_layer.z_value)
    elif conv_layer.activation == 'RELU':
        conv_layer.a_value = activation_function.RELU_forward(conv_layer.z_value)

def pooling(pooling_layer):
    output_feature = np.zeros(pooling_layer.output_size)
    output_feature_index = np.zeros(pooling_layer.output_size)
    if pooling_layer.type == 'max':
        for k in range(pooling_layer.output_size[0]):
            for n in range(pooling_layer.output_size[1]):
                for i in range(pooling_layer.output_size[2]):
                    for j in range(pooling_layer.output_size[3]):
                        output_feature[k,n,i,j] = np.max(pooling_layer.input[k,n,2*i:2*i+
                                                                pooling_layer.size,2*j:
                                                                2*j+pooling_layer.size]
                                                                )
                        output_feature_index[k,n,i,j] = np.argmax(pooling_layer.input[k,n,2*i:
                                                                2*i+pooling_layer.size,
                                                                2*j:2*j+pooling_layer.
                                                                size])
    elif pooling_layer.type == 'average':
        for k in range(pooling_layer.output_size[0]):
            for n in range(pooling_layer.output_size[1]):
                for i in range(pooling_layer.output_size[2]):
                    for j in range(pooling_layer.output_size[3]):
                        output_feature[k,n,i,j] = np.average(pooling_layer.input[k,n,2*i:2*i+
                                                                pooling_layer.size,2*j:
                                                                2*j+pooling_layer.size]
                                                                )
        output_feature_index = None
    pooling_layer.z_value = output_feature
    pooling_layer.a_value = output_feature
    pooling_layer.up_sampling = output_feature_index!=0

def full_connected(full_connected_layer):
    full_connected_layer.z_value = np.dot(full_connected_layer.weights,full_connected_layer.
                                        input) + full_connected_layer.bias
    if full_connected_layer.activation == 'Sigmoid':
        full_connected_layer.a_value = activation_function.Sigmoid_forward(
                                            full_connected_layer.z_value)
    elif full_connected_layer.activation == 'RELU':
        full_connected_layer.a_value = activation_function.RELU_forward(full_connected_layer.
                                            z_value)


def CNN_forward_function(model,input_data):
    model[0].input = input_data
    model[0].a_value = input_data
    #print(model[0].a_value.shape)
    for i in range(1,len(model)):
        model[i].input = model[i-1].a_value
        if model[i].layer_kind == 'conv':
            conv_2d(model[i])
        if model[i].layer_kind == 'pool':
            pooling(model[i])
        if model[i].layer_kind == 'full':
            if model[i-1].layer_kind == 'conv' or model[i-1].layer_kind == 'pool':
                model[i].input = model[i].input.reshape(model[i].input.shape[0],model[i].input
                                                    .shape[1]*model[i].input.shape[
                                                    2]*model[i].input.shape[3]).T
            full_connected(model[i])
```

2.4 后向传播代码

```
import numpy as np
import random
import time
import matplotlib.pyplot as plt
import CNN_class
import activation_function
import copy

def conv_2d(conv_layer):
    conv_length = int((conv_layer.input.shape[1] - conv_layer.size + 2*conv_layer.zero_padding
                                                    ) / conv_layer.stride + 1)
    conv_width = int((conv_layer.input.shape[2] - conv_layer.size + 2*conv_layer.zero_padding)
                                                     / conv_layer.stride + 1)
    feature_map = np.zeros((conv_layer.input.shape[0],conv_layer.depth,conv_length,conv_width)
                                                    )
    for num in range(conv_layer.input.shape[0]):
        for j in range(conv_layer.depth):
            for k in range(conv_length):
                for m in range(conv_width):
                    feature_map[num,j,k,m] = np.sum(np.multiply(conv_layer.input[num,k:k+
                                                                conv_layer.size,m:m+
                                                                conv_layer.size],conv_layer
                                                                .weights[j]))
                                                + conv_layer.bias[j,0]
    conv_layer.z_value = feature_map
    if conv_layer.activation == 'Sigmoid':
        conv_layer.a_value = activation_function.Sigmoid_forward(conv_layer.z_value)
    elif conv_layer.activation == 'RELU':
        conv_layer.a_value = activation_function.RELU_forward(conv_layer.z_value)

def pooling(pooling_layer):
    output_feature = np.zeros(pooling_layer.output_size)
    output_feature_index = np.zeros(pooling_layer.output_size)
    if pooling_layer.type == 'max':
        for k in range(pooling_layer.output_size[0]):
            for n in range(pooling_layer.output_size[1]):
                for i in range(pooling_layer.output_size[2]):
                    for j in range(pooling_layer.output_size[3]):
                        output_feature[k,n,i,j] = np.max(pooling_layer.input[k,n,2*i:2*i+
                                                            pooling_layer.size,2*j:
                                                            2*j+pooling_layer.size]
                                                            )
                        output_feature_index[k,n,i,j] = np.argmax(pooling_layer.input[k,n,2*i:
                                                            2*i+pooling_layer.size,
                                                            2*j:2*j+pooling_layer.
                                                            size])
    elif pooling_layer.type == 'average':
        for k in range(pooling_layer.output_size[0]):
            for n in range(pooling_layer.output_size[1]):
                for i in range(pooling_layer.output_size[2]):
                    for j in range(pooling_layer.output_size[3]):
                        output_feature[k,n,i,j] = np.average(pooling_layer.input[k,n,2*i:2*i+
                                                            pooling_layer.size,2*j:
                                                            2*j+pooling_layer.size]
                                                            )
        output_feature_index = None
    pooling_layer.z_value = output_feature
    pooling_layer.a_value = output_feature
    pooling_layer.up_sampling = output_feature_index!=0

def full_connected(full_connected_layer):
    full_connected_layer.z_value = np.dot(full_connected_layer.weights,full_connected_layer.
                                                    input) + full_connected_layer.bias
    if full_connected_layer.activation == 'Sigmoid':
        full_connected_layer.a_value = activation_function.Sigmoid_forward(
                                                full_connected_layer.z_value)
    elif full_connected_layer.activation == 'RELU':
        full_connected_layer.a_value = activation_function.RELU_forward(full_connected_layer.
                                                z_value)


def CNN_forward_function(model,input_data):
    model[0].input = input_data
    model[0].a_value = input_data
    #print(model[0].a_value.shape)
    for i in range(1,len(model)):
        model[i].input = model[i-1].a_value
        if model[i].layer_kind == 'conv':
            conv_2d(model[i])
        if model[i].layer_kind == 'pool':
            pooling(model[i])
        if model[i].layer_kind == 'full':
            if model[i-1].layer_kind == 'conv' or model[i-1].layer_kind == 'pool':
                model[i].input = model[i].input.reshape(model[i].input.shape[0],model[i].input
                                                .shape[1]*model[i].input.shape[
                                                2]*model[i].input.shape[3]).T
            full_connected(model[i])
```

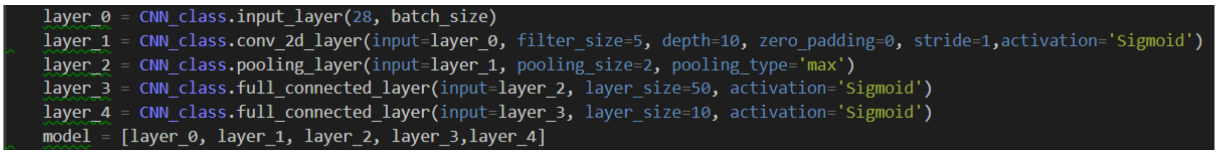
2.5 CNN主函数

```
import numpy as np
import random
import datetime
import matplotlib.pyplot as plt
import CNN_forward
import CNN_class
import CNN_backward
from tensorflow.examples.tutorials.mnist import input_data
import activation_function

def load_data(filename):
    data_set = input_data.read_data_sets(filename,one_hot=True)
    train_data,train_label,test_data,test_label = data_set.train.images,
                                                  data_set.train.labels,
                                                  data_set.test.images,
                                                  data_set.test.labels
    return train_data,train_label,test_data,test_label

def mini_batch_generate(data,size):
    mini_batches = []
    total_number = len(data)
    for i in range(0,total_number,size):
        mini_batches.append(data[i:i+size])
    return mini_batches

def train_process(model,data,label,alpha):
    CNN_forward.CNN_forward_function(model,data)
    cost = np.sum((model[-1].a_value - label)**2)
    CNN_backward.CNN_backward_function(model,label,0.01,'RSE')
    return cost
```

```
def test_process(model, test_data, test_label):
    total_num = 10000
    true_number = 0
    CNN_forward.CNN_forward_function(model, test_data)
    result = np.argmax(model[-1].a_value, axis=0).reshape(total_num, 1)
    true_label = np.array([np.argmax(y) for y in test_label.T]).reshape(total_num, 1)
    for i in range(total_num):
        if result[i] == true_label[i]:
            true_number += 1
    accuracy = true_number / total_num
    return accuracy

if __name__ == '__main__':
    start_time = datetime.datetime.now()
    train_data, train_label, test_data, test_label = load_data('MNIST')
    training_set = list(zip(train_data,train_label))
    total_number = len(training_set)
    input_number = train_data.shape[1]
    batch_size = 100
    layer_0 = CNN_class.input_layer(28, batch_size)
    layer_1 = CNN_class.conv_2d_layer(input=layer_0,filter_size=5,depth=10,zero_padding=0,
                                       stride=1,activation='Sigmoid')
    layer_2 = CNN_class.pooling_layer(input=layer_1, pooling_size=2, pooling_type='max')
    layer_3 = CNN_class.full_connected_layer(input=layer_2, layer_size=50, activation='Sigmoid
                                       ')
    layer_4 = CNN_class.full_connected_layer(input=layer_3, layer_size=10, activation='Sigmoid
                                       ')
    model = [layer_0, layer_1, layer_2, layer_3,layer_4]
    accuracy_list = []
    for iteration in range(10):
        np.random.shuffle(training_set)
        train_data_shuffle = np.array(list(zip(*training_set))[0])
        train_label_shuffle = np.array(list(zip(*training_set))[1])
        mini_batches_data = mini_batch_generate(train_data_shuffle, batch_size)
        mini_batches_label = mini_batch_generate(train_label_shuffle, batch_size)
        for i in range(50):
            data = mini_batches_data[i].reshape(batch_size,28,28)
            label = mini_batches_label[i].T
            cost = train_process(model,data,label,0.1)
        print('accuracy:',test_process(model,test_data.reshape(10000,28,28),test_label.T))
        accuracy_list.append(test_process(model,test_data.reshape(10000,28,28),test_label.T))
```

三　结果及图形展示

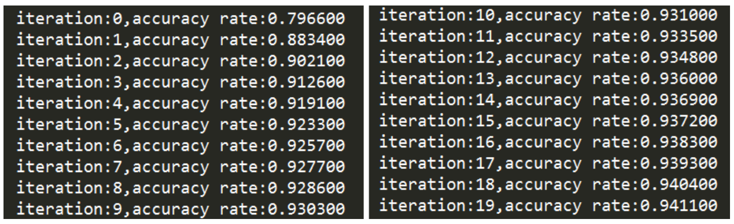
每一层模型的参数设置如下图6：学习率设置为:0.1，由于代码的优化没有做的很好，所以CNN的



```
layer_0 = CNN_class.input_layer(28, batch_size)
layer_1 = CNN_class.conv_2d_layer(input=layer_0, filter_size=5, depth=10, zero_padding=0, stride=1,activation='Sigmoid')
layer_2 = CNN_class.pooling_layer(input=layer_1, pooling_size=2, pooling_type='max')
layer_3 = CNN_class.full_connected_layer(input=layer_2, layer_size=50, activation='Sigmoid')
layer_4 = CNN_class.full_connected_layer(input=layer_3, layer_size=10, activation='Sigmoid')
model = [layer_0, layer_1, layer_2, layer_3,layer_4]
```

图 6: 测试模型的各层参数设置

代码跑的非常慢，所以这里只跑了20次迭代，给出了实验结果以及最终绘制的准确率曲线如下图7和图8所示：



```
iteration:0,accuracy rate:0.796600
iteration:1,accuracy rate:0.883400
iteration:2,accuracy rate:0.902100
iteration:3,accuracy rate:0.912600
iteration:4,accuracy rate:0.919100
iteration:5,accuracy rate:0.923300
iteration:6,accuracy rate:0.925700
iteration:7,accuracy rate:0.927700
iteration:8,accuracy rate:0.928600
iteration:9,accuracy rate:0.930300
```

```
iteration:10,accuracy rate:0.931000
iteration:11,accuracy rate:0.933500
iteration:12,accuracy rate:0.934800
iteration:13,accuracy rate:0.936000
iteration:14,accuracy rate:0.936900
iteration:15,accuracy rate:0.937200
iteration:16,accuracy rate:0.938300
iteration:17,accuracy rate:0.939300
iteration:18,accuracy rate:0.940400
iteration:19,accuracy rate:0.941100
```

图 7: 模型训练效果

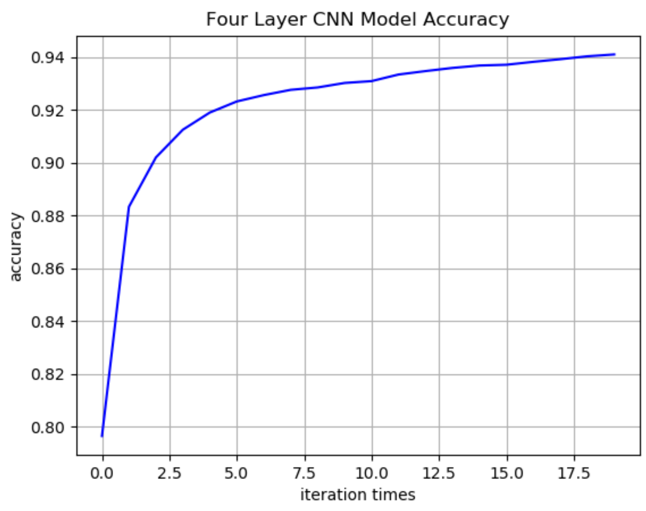


图 8: 模型训练效果

四 总结体会

这一次的CNN的代码花了很长的时间去写，主要时间花在了代码的封装和一些代码的细节运算问题上，大概三天的所有白天的时间才写完成，其中花了很长的时间去理解其反向传播的所有细节内容，另外就是在numpy的一些函数的使用上踩了很多的坑，尤其其中的矩阵的广播机制以及数据维度的压缩，因此往往出现和自己的想法不一致的情况，非常恼火。但总算将这个比较难的任务完成了，个人觉得还是很有成就感的。但是其实还是暴露出来了很多的问题，尤其是在代码的效率上，在实际运行CNN的过程中，我发现大量的时间都花在做卷积上，因为其中的运算量很大，然而我的代码在这个地方大量的使用了循环遍历，很大程度上导致代码跑的非常慢，大概做一次迭代将6万张图片全部放入模型训练需要大概半小时，因此在最终的结果上我只给出了20次迭代的结果。接下来考虑如何在卷积以及反向传播的部分代码上做一些改进提高一些效率。最后的结果显示并未达到预期的CNN传说的99%的正确率，其主要问题可能还是在于我的迭代次数太少以及代码的参数设置存在一些问题。