斯坦福大学公开课 Machine Learning 学习笔记

第五章 神经网络(Neural Networks)——学习网络权值

上一章学习了神经网络的基本概念和前向传播等内容，本章学习神经网络的重点——反向传播算法，最后是通过梯度下降算法训练神经网络。

代价函数和反向传播（Back Propagation）

- 代价函数

衡量神经网络算法的好坏同样使用代价函数，具有 sigmoid 单元的神经网络的代价函数与逻辑回归的代价函数很相似，下面是正则化的神经网络代价函数。

：

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_i^{(k)}\log\left(h_\theta\left(x^{(i)}\right)_k\right) + \left(1-y_i^{(k)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)_k\right)\right] +$$

$$\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(\Theta_{ji}^{(i)}\right)^2 \qquad \text{(公式 1)}$$

其中，

K 与分类的类别数 n 有关，当 n=2 时 K=1，当 n>=3 时 K=n，同时$h_\theta(X) \in R^K$。

$h_\theta\left(x^{(i)}\right)_k$是第 k 个输出，即输出向量中的第 k 个元素。

L 是神经网络的总层数。

S_l是第 l 层的单元个数，不包含偏置单元。

- 反向传播算法

由之前几章的内容可知，使用梯度下降算法训练代价函数达到全局最小值（$\min_{\Theta} J(\theta)$）需要计算两个公式的值：$J(\theta)$和$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)$。只有计算出了$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)$才能使用梯度下降调整每个$\theta_{ij}^{(l)}$一点，但是按照之前计算$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)$的方法是行不通的，因为处于隐藏层中的$\theta_{ij}^{(l)}$是不直接向最终的结果$J(\theta)$作出贡献的，也就是说$J(\theta)$公式中除了正则化项以外是不含有隐藏层中的$\theta_{ij}^{(l)}$变量的，所以我们不能直接使用偏微分公式求$J(\theta)$对隐藏层中的$\theta_{ij}^{(l)}$的偏微分。

这个时候就需要反向传播算法，将输出层造成的误差一层一层传递回隐藏层，从而计算出隐藏层权值的偏微分。下面先直接给出反向传播算法过程，后面小节会继续学习反向传播算法的原理，算法中使用的符号在上一章内容中已说明。

反向传播算法：

选择数据集$\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\ldots,(x^{(m)},y^{(m)})\}$。

将所有的$\Delta_{i,j}^{(l)}$都初始化为零。

遍历每个训练集样本 t=1 到 m，执行以下过程：

1. $a^{(1)}=x^{(t)}$
2. 使用前向传播计算每一层的$a^{(l)}$
3. 使用$y^{(t)}$计算输出层的误差$\delta^{(L)}=a^{(L)}-y^{(t)}$
4. 使用以下公式反向计算每层隐藏层的误差$\delta^{(L-1)}$，$\delta^{(L-2)}$，…，$\delta^{(2)}$:

$$\delta^{(l)}=\left(\left(\Theta^{(l)}\right)^{T}\delta^{(l+1)}\right).*\ g'(z^{(l)})$$，其中$g'\left(z^{(l)}\right)=a^{(l)}.*(1-a^{(l)})$，即 sigmod 函数

g(z)对 z 的导数。

5. 更新Δ矩阵：调整每个$\Delta_{i,j}^{(l)}=\Delta_{i,j}^{(l)}+a_j^{(l)}\delta_i^{(l+1)}$，或者向量式的计算方式为：$\Delta^{(l)}=\Delta^{(l)}+\delta^{(l+1)}(a^{(l)})^{T}$

遍历所有的训练样本后：

$\mathrm{D}_{i,j}^{(l)}=\frac{1}{m}(\Delta_{i,j}^{(l)}+\lambda\Theta_{i,j}^{(l)})$，如果$j\neq 0$，即非偏置单元

$\mathrm{D}_{i,j}^{(l)}=\frac{1}{m}\Delta_{i,j}^{(l)}$，如果$j=0$，偏置单元

则，$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)=\mathrm{D}_{i,j}^{(l)}$

注意，算法中的.*符号是指 MatLab 的.*，这个 Python 的 Numpy 中的 dot 运算不同，关于其中的不同可通过网络搜索。

计算出每个$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)$之后，就可以使用梯度下降更新每个：$\theta_{ij}^{(l)}=\theta_{ij}^{(l)}-\alpha\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)=\theta_{ij}^{(l)}-\mathrm{D}_{i,j}^{(l)}$，直到代价函数达到最小值。

其中，算法中的$\delta^{(l)}$表示 l 层中所有单元的误差的向量，$\delta_i^{(l)}$表示 l 层 i 单元输出$a_i^{(l)}$的误差。

上面的算法直接写出来，很难让人信服，为什么根据误差可以计算出$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)$，以及为什么根据误差调整后的权重可以使误差函数的值接近最小？这两个问题会在接下来的小节得到答案。

反向传播的数学原理

由于自己感觉课堂中讲解的反向传播算法的直观感受并不那么直观，而且并不能让我在数学上信服，所以这里我将不记录课堂中对反向传播的讲解，而是从数学推导出偏微分的方式体会反向传播算法，同时给出了上一小节中关于为什么根据误差调整后的权重可以使误差函数的值接近最小的数学解答。

下面推导图 1 神经网络结构中$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)$的值，图中其实画出了简单的前向传播的过程。此处有必要再说明下推导中使用到的符号：

$a_i^{(j)}$:表示第 j 层的第 i 个激励值；

$\theta_{ij}^{(l)}$:表示第 l 层的激励$a_j^{(l)}$对下一层(l+1 层)的激励$a_i^{(l+1)}$计算作用的权重；

$\theta^{(l)}$:表示第 l 层对第 l+1 层所有权重的矩阵；

$a^{(l)}$:表示第 l 层的所有激励的矩阵；

$z_i^{(l)}$:表示第 l 层第 i 个单元的输入；

$z^{(l)}$:表示第 l 层所有单元输入的矩阵；

$\bar{y}_i$:表示输出层第 i 个单元的输出，图 1 中只有一个，即$a_1^{(4)}$；

K :表示输出层单元个数，图 1 中 K=1。

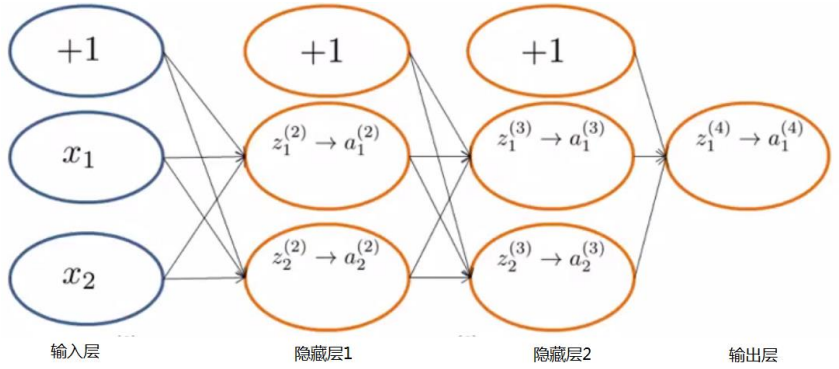

图 1 四层神经网络

先推导输入层中的偏微分$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$，即 l=1：

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = \frac{\partial J(\theta)}{\partial z_i^{(2)}} \cdot \frac{\partial z_i^{(2)}}{\partial \theta_{ij}^{(l)}} \quad \text{（公式 2）}$$

其中，对于输入层来说，$z_i^{(2)} = \theta_{i0}^{(1)} + \theta_{i1}^{(1)} x_1 + \theta_{i2}^{(1)} x_2 = \sum_{j=0}^{2}(x_j \cdot \theta_{ij}^{(l)})$，所以：

$$\frac{\partial z_i^{(2)}}{\partial \theta_{ij}^{(l)}} = x_j \quad \text{（公式 3）}$$

将公式 3 带入公式 2 得：

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = x_j \cdot \frac{\partial J(\theta)}{\partial z_i^{(2)}} \quad \text{（公式 4）}$$

下面只需要求出$\frac{\partial J(\theta)}{\partial z_i^{(2)}}$即可。由于$z_i^{(2)}$对隐藏层 2 的所有单元输入都有影响，所以：

$$\frac{\partial J(\theta)}{\partial z_i^{(2)}} = \sum_{j=1}^{2}(\frac{\partial J(\theta)}{\partial z_j^{(3)}} \cdot \frac{\partial z_j^{(3)}}{\partial z_i^{(2)}}) \quad \text{（公式 5）}$$

由于$z_j^{(3)} = \sum_{i=0}^{2}\left(g(z_i^{(2)}) \cdot \theta_{ji}^{(2)}\right)$，所以：

$$\frac{\partial z_j^{(3)}}{\partial z_i^{(2)}} = \frac{\partial z_j^{(3)}}{g(z_i^{(2)})} \cdot \frac{g(z_i^{(2)})}{z_i^{(2)}} = \theta_{ji}^{(2)} \cdot g'(z_i^{(2)}) \quad \text{（公式 6）}$$

将公式 6 代入到公式 5 得：

$$\frac{\partial J(\theta)}{\partial z_i^{(2)}} = \sum_{j=1}^{2}(\frac{\partial J(\theta)}{\partial z_j^{(3)}} \cdot \theta_{ji}^{(2)} \cdot g'(z_i^{(2)})) = g'(z_i^{(2)}) \sum_{j=1}^{2}(\frac{\partial J(\theta)}{\partial z_j^{(3)}} \cdot \theta_{ji}^{(2)}) \quad \text{（公式 7）}$$

现在记：

$$\delta_i^{(2)} = \frac{\partial J(\theta)}{\partial z_i^{(2)}} \quad \text{（公式 8）}$$

则：

$$\delta_i^{(2)} = g'(z_i^{(2)}) \sum_{j=1}^{2}(\delta_j^{(3)} \cdot \theta_{ji}^{(2)}) \quad \text{（公式 9）}$$

依据同样的方式，可以推导出相同的隐藏层 2 的公式：

$$\delta_i^{(3)} = g'(z_i^{(3)}) \sum_{j=1}^{2} (\delta_j^{(4)} \cdot \theta_{ji}^{(3)}) \quad \text{（公式 10）}$$

最后推导出输出层的：

$$\delta_i^{(4)} = \frac{\partial J(\theta)}{\partial z_i^{(4)}} = \frac{\partial \sum_{j=1}^{K} (\frac{1}{2}((\bar{y}_j - y_j)^2))}{\partial z_i^{(4)}} = (\bar{y}_i - y_i) \frac{\partial \bar{y}_i}{\partial z_i^{(4)}} \quad \text{（公式 10）}$$

令$e_i = (\bar{y}_i - y_i)$，则公式 10 可变为：

$$\delta_i^{(4)} = e_i \cdot g'(z_i^{(4)}) \quad \text{（公式 11）}$$

从公式 11 可看出，其实反向传播的就是误差e，但又不是纯粹地传播误差，而是沿着每个单元的输入值的偏微分方向。

虽然推导$\delta_i^{(1)}$是从输入层向输出层推导，但是计算得从输出层向输入层反向计算，因为经过正向传播，我们最先知道的就是$\delta_i^{(4)}$。下面我们就可以得出各层的误差函数对权重的偏微分$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(1)}}$：

隐藏层 2：

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(3)}} = \frac{\partial J(\theta)}{\partial z_i^{(4)}} \cdot \frac{\partial z_i^{(4)}}{\partial \theta_{ij}^{(3)}} = \delta_i^{(4)} \cdot g\left(z_j^{(3)}\right) = \delta_i^{(4)} \cdot a_j^{(3)} \quad \text{（公式 11）}$$

隐藏层 1：

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(2)}} = \frac{\partial J(\theta)}{\partial z_i^{(3)}} \cdot \frac{\partial z_i^{(3)}}{\partial \theta_{ij}^{(2)}} = \delta_i^{(3)} \cdot g(z_j^{(2)}) = \delta_i^{(3)} \cdot a_j^{(2)} \quad \text{（公式 11）}$$

最后是输入层：

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(1)}} = \delta_i^{(2)} \cdot x_j \quad \text{（公式 12）}$$

从以上公式可看出反向传播的意义：由总的误差值（$\delta_i^{(4)}$）反向逐层推算出各层的权值对总误差做了多少贡献。

反向传播实践

（略）

梯度检查（Gradient Checking）——数值估算（Numerical Evaluation）

从之前的学习我们可以看出，实现反向传播算法是比较困难的，而且容易出错，那么就需要有种能够验证所实现的算法是否正确的方式。由于反向传播算法是用来计算$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(1)}}$的，所以我

们可以通过验证计算所得的结果是否为$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$。这一点可以通过数值估算来验证。如下图 2，红线表示J(θ)在$\theta = \theta_0$处的切线，其斜率为$J(\theta_0)$的导数$\frac{d}{d\theta}J(\theta_0)$；而绿色粗线的斜率可表示为$\frac{J(\theta_0+\epsilon)-J(\theta_0-\epsilon)}{2\epsilon}$，从图 2 可以形象地看出，当ϵ特别小（$10^{-4}$）的时候，这两个斜率是非常接近的，所以我们可以使用像$\frac{J(\theta_0+\epsilon)-J(\theta_0-\epsilon)}{2\epsilon}$一样的方式来验证我们所求的$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$是否正确，因为$\frac{J(\theta_0+\epsilon)-J(\theta_0-\epsilon)}{2\epsilon}$是很容易求解的。

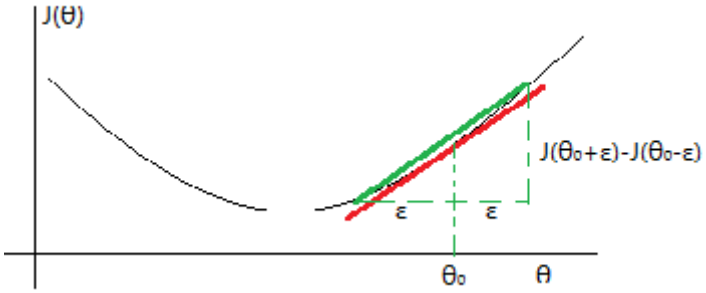

图 2 代价函数的估计

设参数向量$\theta \in R^n$为所有$\theta_{ij}^{(l)}$的依次展开形式，即$\theta = [\Theta^{(1)} \quad \Theta^{(2)} \quad \Theta^{(3)} \quad \dots \quad \Theta^{(m)}]$，m 为网络层数。从而使用下面的循环方式验证$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$值是否正确：

$$\frac{\partial J(\theta)}{\partial \theta_1} \approx \frac{J(\theta_1+\epsilon,\theta_2,\theta_2,\dots,\theta_n)-J(\theta_1-\epsilon,\theta_2,\theta_2,\dots,\theta_n)}{2\epsilon}$$

$$\frac{\partial J(\theta)}{\partial \theta_2} \approx \frac{J(\theta_1,\theta_2+\epsilon,\theta_2,\dots,\theta_n)-J(\theta_1,\theta_2-\epsilon,\theta_2,\dots,\theta_n)}{2\epsilon}$$

……

$$\frac{\partial J(\theta)}{\partial \theta_n} \approx \frac{J(\theta_1,\theta_2,\theta_2,\dots,\theta_n+\epsilon)-J(\theta_1,\theta_2,\theta_2,\dots,\theta_n-\epsilon)}{2\epsilon}$$

每次我们只控制一个权值变化从而求出其近似的偏微分数，与我们使用反向传播算法的结果进行比较。其中，对于变化后的$\theta_i+\epsilon$和$\theta_i-\epsilon$的代价函数值需要使用前向传播算法进行计算。从而，可以看出梯度验证是很耗时的。所以，在算法实现中，当我们使用梯度检查反向传播算法的实现是正确的以后就应该关闭梯度检查。

随机初始化权值向量（Random Initialization）

当运行梯度下降算法或者更高级优化算法时，需要给每个权重一个初始值，然后循环多次执行前向传播和反向传播计算$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$，然后使代价函数值下降一点，直到降到最小。但是，应该将权值θ初始化为多少呢？

每个$\theta_{ij}^{(1)}=0$如何？这意味着最后的逻辑回归单元只会得到一种特征，因为所有的逻辑回归单元都一样，这样便阻止神经网络学习更有价值的信息。所以，一般采用随机方式初始化所有的权值。

总结

在训练神经网络之前要做的第一件事是搭建网络的框架（Connectivity Pattern Between Neurons），如之前所述，神经网络的框架（结构）主要由输入层及输入层单元数、输出层及输出层单元数、隐藏层层数及每层单元数组成，下面是简单的介绍如何确定这些值的方法：
输入层单元数：固定的特征值数，一般可再增加一个偏置单元。
输出层：分两类只有一个输出单元，分类大于两类的单元数为类别数。
隐藏层层数：一般情况下为 1，但是如果要分类的特征比较复杂，可以合理增加。
隐藏层单元数：一般是越多越好，但是也要考虑计算复杂度。
下面总结出整个神经网络算法训练过程：

1. 随机初始化权值；
2. 实现前向传播算法并计算每个x^i对应的估计值$\bar{y}_i$；
3. 实现代价函数并计算所有估计输出的代价$J(\theta)$，如果$J(\theta)$达到最小或者达到可接受的范围，则算法结束，否则继续执行 4；
4. 实现并计算代价函数对所有权重的偏微分$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$：

   遍历所有 m 组训练样例：

   对于样例$(x^{(i)},y^{(i)})$执行前向传播算法（计算每层的激励值$a^{(l)}$）和反向传播算法(计算每层的误差$\delta^{(l)}$)，累积误差：$\Delta_{i,j}^{(l)}=\Delta_{i,j}^{(l)}+a_j^{(l)}\delta_i^{(l+1)}$

   根据$\Delta_{i,j}^{(l)}$计算偏微分$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$；
5. 使用梯度检查方法检查偏微分计算的正确性；
6. 在偏微分计算正确的基础上，使用梯度下降算法或者高级优化算法调整权值后跳转到第 2 步。

下面是使用神经网络算法识别人脸照片是否佩戴太阳镜的 python 实现代码，图片数据来自 Tom M. Mitchell 的《机器学习》中《人工神经网络》部分的课后题，数据链接:http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/faces.html (2016)。本算法使用 80%的数据作为训练数据集，20%的数据作为测试数据集，其中训练数据集的识别率为 98%，测试集的识别率为 88%，后续未再调校算法从而获取更高的测试集准确率，下面是具体的代码：
首先是格式化数据：

```
#!coding:utf-8
'''
该文件将图片文件的内容转换成对应的结果数据
如 an2i_left_angry_open.pgm 会转换为三种结果：
方向：left 表情：angry 是否佩戴眼镜：否
其中，方向统计放在 direction.txt 中，1 为 left, 2 为 right,3 为 straight，4 为 up
表情统计放在 expression.txt 中，1 为 angry,2 为 happy,3 为 neutral，4 为 sad
是否佩戴眼镜放在 sunglass.txt 中，0 为未佩戴，1 为佩戴
```

```
'''
import os, random

faceDir = 'faces'
testPercent = 0.2

def findDirection(strFileName):
    if strFileName.find('_left_') >= 0:
        return 1
    elif strFileName.find('_right_') >= 0:
        return 2
    elif strFileName.find('_straight_') >= 0:
        return 3
    elif strFileName.find('_up_') >= 0:
        return 4
    return -1

def findExpression(strFileName):
    if strFileName.find('_angry_') >= 0:
        return 1
    elif strFileName.find('_happy_') >= 0:
        return 2
    elif strFileName.find('_neutral_') >= 0:
        return 3
    elif strFileName.find('_sad_') >= 0:
        return 4
    return -1

def findSunglass(strFileName):
    if strFileName.find('_sunglasses') >= 0:
        return 1
    elif strFileName.find('_open') >= 0:
        return 0
    return -1

expressions = []
directions = []
sunglasss = []

faceSubDirList = [faceDir+'/'+d for d in os.listdir(faceDir)]
for direct in faceSubDirList:
    if os.path.isdir(direct):
        fileList = [direct+'/'+file for file in    os.listdir(direct) if file.endswith('_4.pgm')]
        #print(fileList)
```

```
        for fileName in fileList:
            d = findDirection(fileName)
            if d > 0:
                expressions.append(fileName+' ' + str(d))
            e= findExpression(fileName)
            if e > 0:
                directions.append(fileName+' ' + str(e))
            s = findSunglass(fileName)
            if s >= 0:
                sunglasss.append(fileName+' ' + str(s))

#打乱数据
random.shuffle(expressions)
random.shuffle(directions)
random.shuffle(sunglasss)

#训练数据
with open('expression_train.txt', 'w') as expFile , open('direction_train.txt','w') as dirFile,\
        open('sunglass_train.txt','w') as sunFile:
    for l in expressions[int(testPercent*len(expressions)) : ]:
        expFile.write(l+'\n')
    for l in directions[int(testPercent*len(directions)) : ]:
        dirFile.write(l+'\n')
    for l in sunglasss[int(testPercent*len(sunglasss)) : ]:
        sunFile.write(l+'\n')

#测试数据
with open('expression_test.txt', 'w') as expFile , open('direction_test.txt','w') as dirFile,\
        open('sunglass_test.txt','w') as sunFile:
    for l in expressions[0 : int(testPercent*len(expressions))]:
        expFile.write(l+'\n')
    for l in directions[0 : int(testPercent*len(directions))]:
        dirFile.write(l+'\n')
    for l in sunglasss[0 : int(testPercent*len(sunglasss))]:
        sunFile.write(l+'\n')
```

读取图片数据模块：

```
import re
import numpy

def read_pgm(filename, byteorder='>'):
    """Return image data from a raw PGM file as numpy array.
    Format specification: http://netpbm.sourceforge.net/doc/pgm.html
    """
    header, width, height, maxval = (0,0,0,0)
```

```
    with open(filename, 'rb') as f:
        buffer = f.read()
    try:
        header, width, height, maxval = re.search(
            b"(^P5\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n])*"
            b"(\d+)\s(?:\s*#.*[\r\n]\s)*)", buffer).groups()
    except AttributeError:
        try:
            header, width, height, maxval = re.search(
                b"(^P2\s(?:\s*#.*[\r\n])*"
                b"(\d+)\s(?:\s*#.*[\r\n])*"
                b"(\d+)\s(?:\s*#.*[\r\n])*"
                b"(\d+)\s(?:\s*#.*[\r\n]\s)*)", buffer).groups()
        except AttributeError:
            raise ValueError("Not a raw PGM file: '%s'" % filename)
    return numpy.frombuffer(buffer,
                            dtype = 'u1' if int(maxval) < 256 else byteorder +'u2',
                            count = int(width)*int(height),
                            offset = len(header)
                            ).reshape((int(height), int(width)))

if __name__ == "__main__":
    from matplotlib import pyplot
    image = read_pgm("foo.pgm", byteorder='<')
    pyplot.imshow(image, pyplot.cm.gray)
    pyplot.show()
```

BP 神经网络实现及验证：

```
#!coding:utf-8
'''
该文件实现了可以识别人像图片是否佩戴眼镜的 BP 神经网络算法
'''
import numpy as np
from matplotlib import pyplot as plt
import pgmReader
import time
import    copy

netLayers =    4                              #网络层数，包含输入层和输出层
unitsPerLayer =    960                  #每层的神经单元数，为 0 的话使用和输入层相同的单元数
unitsofInput = 960                      #输入层单元数，由训练数据决定
unitsofOutput = 1                       #输出层单元数，由训练数据决定
lambd = 0.001                           #正则化系数
```

```
alpha = 0.1                                    #学习率
epslon = 10                                    #数值检查参数

#初始化所有权值，返回 np.array
# layers: 网络层数
# unitsInp: 输入层单元数
# unitsOut: 输出层单元数
def RandomInitWeights(layers, unitsInp, unitsOut):
    weightsList = []
    np.random.seed(12)
    for i    in range(layers - 2):
        #随机生成的权值为-5 到 5 之间的小数
        weightsList.append(np.random.rand(unitsInp + 1, unitsInp)*2-1)
    #生成最后一个隐藏层到输出层权值
    weightsList.append(np.random.rand(unitsInp + 1, unitsOut)*2-1)
    return weightsList

#sigmoid 函数
sigmoid = lambda z : 1 / (1 + np.exp(-z))

#前向传播算法
# trainX[in]: 一个训练样本值
# weightsList[in]: 所有的权值列表
# activatList[out]: 经过前向算法计算出来的每层每个单元的激励值
def ForwardPro(trainX, weightsList, activatsList):
    #每层的神经单元的激励值，第一层为输入值
    activat = np.array([1] + list(trainX.flatten()))
    activatPerLayer = [activat]
    for i in range(len(weightsList)):
        activat = sigmoid(activat.dot(weightsList[i]))
        if i < len(weightsList)-1:
            #加入偏置单元
            activat = np.array([1] + list(activat.flatten()))
        activatPerLayer.append(activat)
    activatsList.append(activatPerLayer)
    return activat[0]

#正则化的代价函数
def CostFunc(trainResult,    trainY, weightsList):
    costfunc = lambda result, y : (y * np.log(result) + (1 - y) * np.log(1 - result))
    cost =    -(np.sum(costfunc(trainResult, trainY))) / len(trainY) \
                    + np.sum([np.sum(w[1:] ** 2) for w in weightsList]) * lambd / (2 * len(trainY))
    return cost
```

```
#梯度检查
def GradientCheck(trainResult,    trainY, weightsList, numCheck, direct, trainData):
    numCheck =[np.zeros(w.shape) for w in weightsList]
    for i in range(netLayers):
        print('Layer: ', i)
        for j in range(weightsList[i].shape[0]):
            for k in range(weightsList[i].shape[1]):
                weightsList[i][j][k] += epslon
                trainResults = []
                for data in trainData:
                    r = ForwardPro(data, weightsList, [])
                    trainResults.append(r)
                costPlus = CostFunc(np.array(trainResults), actualY,    weightsList)
                trainResults.clear()
                weightsList[i][j][k] -= 2*epslon
                for data in trainData:
                    r = ForwardPro(data, weightsList, [])
                    trainResults.append(r)
                costMinus = CostFunc(np.array(trainResults), actualY,    weightsList)
                numCheck[i][j][k] = (costPlus - costMinus) / (2 * epslon)
                weightsList[i][j][k] += epslon
                print('GradientCheck:       {0}       ------       BackPro:   {1}   -----costPlus:   {2},   -----costMinus:
{3}'.format(numCheck[i][j][k], direct[i][j][k], costPlus, costMinus))

#反向传播
# weightsList[in&out]:  所有权值列表
# activatList[in]:  经过前向算法计算出来的每层每个单元的激励值
def BackPro(weightsList, activatsList, trainNum, trainYs, trainResult, trainData):
    #初始化 delta 为 0
    deltaList =[np.zeros(w.shape) for w in weightsList]
    #计算各层的误差
    for index in range(trainNum):
        delta = [None] * netLayers
        #计算输出层的误差
        delta[netLayers - 1] = trainResult[index] - trainYs[index]
        for i in range(netLayers - 2, 0, -1):
            t1 = None
            if    i == (netLayers - 2):
                t1 = weightsList[i].dot(delta[i + 1])
            else:
                t1 = weightsList[i].dot(delta[i + 1][1:])
            t1 = t1.reshape(t1.size,1 )
            t2 = activatsList[index][i] * (1 - activatsList[index][i])
            t2 = t2.reshape(t2.size,1)
```

```
                delta[i] = t1 * t2
            #更新总误差
            #deltaList[netLayers-1] += delta[netLayers-1]
            for i in range(netLayers - 2):
                d = delta[i+1][1:].reshape(1, delta[i+1][1:].size)
                a = activatsList[index][i].reshape(activatsList[index][i].size, 1)
                deltaList[i] += a.dot(d)
        direct = []
        #求代价函数对每个权值的偏微分
        for i in range(len(weightsList)):
            #求偏微分时区分了偏置单元
            direct.append(np.vstack((deltaList[i][0,:] / trainNum, (deltaList[i][1:] + lambd * weightsList[i][1:])/trainNum)))
        numCheck = []

        #梯度检查反向传播的正确性
        #GradientCheck(trainResult,  trainYs, weightsList, numCheck, direct, trainData)

        #更新权值
        for i in range(len(weightsList)):
            weightsList[i] -= alpha * direct[i]

#获取数据
def GetImgData(fileName):
    with open(fileName) as testFile:
        trainFiles= [line.split(' ')  for line in  testFile.readlines()]
    trainData = []
    trainYs = []
    trainFileNames= []
    badData = 0
    for data in trainFiles:
        filename, trainY = data
        try:
            imgData = pgmReader.read_pgm(filename, byteorder='<').flatten()
            trainData.append(imgData/256)
            trainYs.append(int(trainY))
            trainFileNames.append(filename)
        except ValueError as ve:
            badData += 1
            continue
    return (trainData, np.array(trainYs), trainFileNames)

#测试结果
def TestWeights(weightsList, filename):
```

```
    #获取测试数据并测试算法准确性
    trainData, actualY, trainFiles = GetImgData(filename)
    correctCount = 0
    errorCount = 0
    for i in range(len(trainData)):
        r = ForwardPro(trainData[i], weightsList, [])
        if (r > 0.5 and actualY[i] == 1) or (r <= 0.5 and actualY[i] == 0):
            correctCount += 1
            if 'test' in filename:
                print(trainFiles[i], '---->', ('sunglasses' if (r > 0.5 and actualY[i] == 1) else 'open'))
        else:
            errorCount += 1
    print('CorrectCount: {0}, ErrorCount: {1}, CorrectPercent: {2:.2f}%, total: {3}'.format(correctCount, errorCount,
\ correctCount/(errorCount+correctCount) * 100, errorCount+correctCount))

if __name__ == '__main__':
    weightsList = RandomInitWeights(netLayers, unitsPerLayer, unitsofOutput)
    #获取训练数据
    trainData,actualY, _1= GetImgData('sunglass_train.txt')
    while True:
        trainResults = []
        activatList = []
        TestWeights(weightsList, 'sunglass_train.txt')
        for data in trainData:
            r = ForwardPro(data, weightsList, activatList)
            trainResults.append(r)
        cost = CostFunc(np.array(trainResults), actualY,    weightsList)
        print('cost: ',    cost)
        if cost < 0.74:
            print('训练完成！！！！ ')
            break
        #使用反向传播修改权值
        BackPro(weightsList, activatList, len(trainData), actualY, np.array(trainResults), trainData)
    TestWeights(weightsList, 'sunglass_test.txt')
```

说明：部分图片直接截自《机器学习》课程，笔记中部分内容参考了 Tom M. Mitchell 的《机器学习》。

个人理解可能存在偏差或错误，欢迎交流和批评指正： wearyoung@outlook.com