華東理工大學

模式识别大作业

题　目　　神经网络人脸识别

学　院　　信息科学与工程

专　业　　控制科学与工程

组　员　　孔子庆

指导教师　　赵海涛

完成日期：　2018 年 10 月 24 日

模式识别作业报告——神经网络人脸识别

组员：孔子庆

通过模式识别课程的学习，在赵海涛老师的辛勤的指导下，我对模式识别的神经网络算法有了一定的了解，并通过本次针对耶鲁大学人脸数据库 B+中的人脸图片数据进行降维，再利用降维后的人脸数据进行有监督神经网络学习进行分类器训练，最终达到人脸识别的目的。

一． 人脸识别任务及 yale B+数据库介绍

人脸识别包括人脸图像采集，人脸定位，人脸识别预处理,·身份确认及身份查找等一系列相关技术，其中人脸图像匹配与识别就是提取人脸图像的特征数据与数据库中存储的特征模板进行搜索匹配，通过设定一个阈值，当相似度超过这一阈值，则把匹配得到的结果输出。人脸识别就是将待识别的人脸特征与已得到的人脸特征模板进行比较，根据相似度对人脸的身份信息进行判断。

本次实验，采用耶鲁大学人脸数据库 B+,数据库中每张人脸数据大小为 192*168,对与每个人都有 64 张人脸图片，分别是再不同光照条件下拍摄的，为了节约计算时间，将原始数据的纬度改成 48*42，并且从每个文件夹中选出 20 张图片当做无监督学习的数据集，14 张作为有监督训练时的训练数据，10 张作为有监督训练的测试数据。

二．解决方案

神经网络总结构

该神经网络由两部分组成，第一部分为基于无监督学习的自编码器的训练完毕后的编码部分，第二部分为有监督训练单层网络，这里将运用自编码器对人脸进行数据降维，再结合有监督学习结合有标记数据进行分类训练。下面将介绍无监督学习的自编码器算法原理及程序实现与有监督训练的算法原理及人脸识别的程序实现。

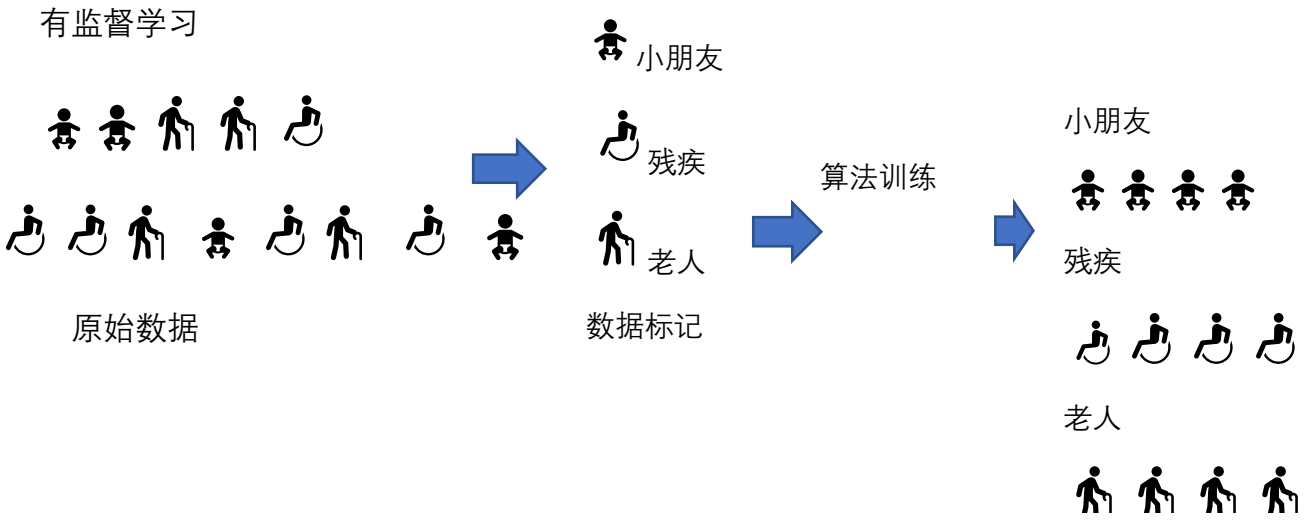
1.无监督学习与有监督学习

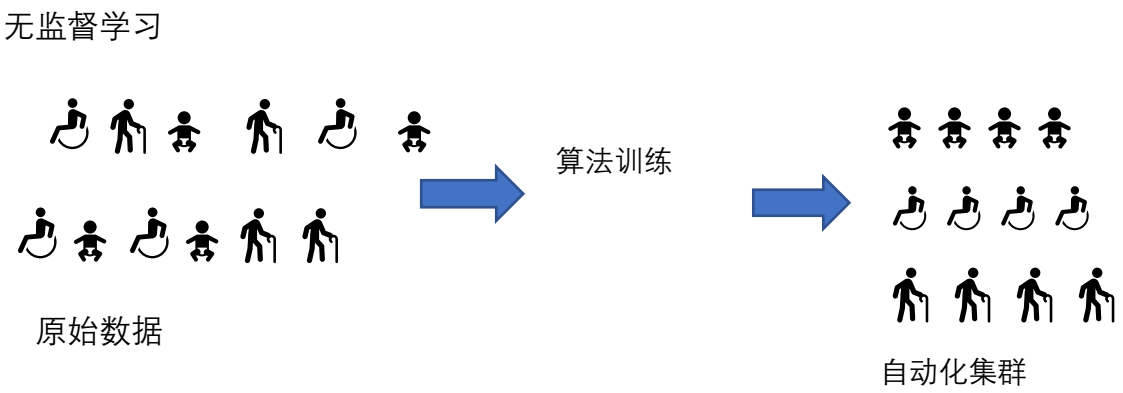
目前许多有监督学习算法，如 SVM,DNN 或者 boosting，决策树等，都是在工业界分类或决策任务上取得了不错的成果，但是这些有监督学习需要大量

带标签的数据，对数据进行上标签又是一个需要耗费人力与时间的任务，有许多数据都是不带标签的，因此我们可以利用无监督学习对其进行聚类或者特征提取，利用无监督学习得到的特征结果也可以应用到带标记数据较少的有监督学习任务中，提高其分类性能。

无监督学习的目的是利用无标记数据推断出数据内部隐藏的结构特征，与有监督学习从有标记的数据中学到一个有正确答案的模型不同，由于无监督学习的数据是未经标记的，因此在学习过程中就没有所谓误差或者知道信号去评估一个可能的解决方案。

从下图可以有监督学习与无监督学习的区别:





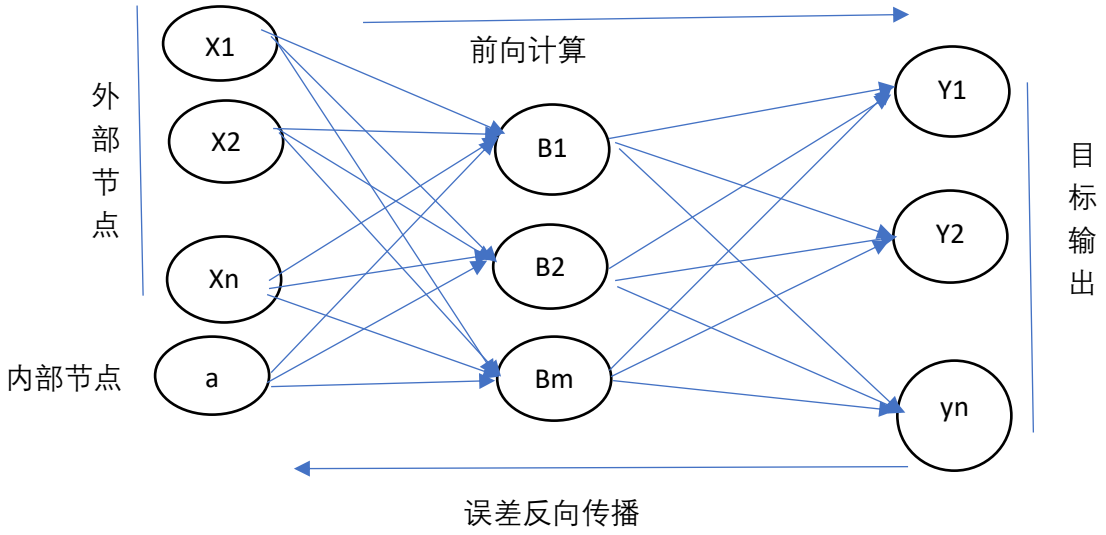
与有监督学习有明确的目标不同，无监督学习没有给定目标输出，而是去提取数据本身的静态结构特征，如在上图中，有监督学习的目的是可对输入的目标进行分类，区别这些图形是数目形状的，但是无监督学习开始训练

前，对于机器来说这些图形没有什么区别，但在经过训练后，我们能够获得这些数据的结构特征，根据这些图形各自的特点，根据图形的相似点对其进行聚类，但是，在无监督学习之后，我们也不知道这些图形分别属于什么图形，仅是根据物以类聚的概念，将其划分为几个不同的集群。

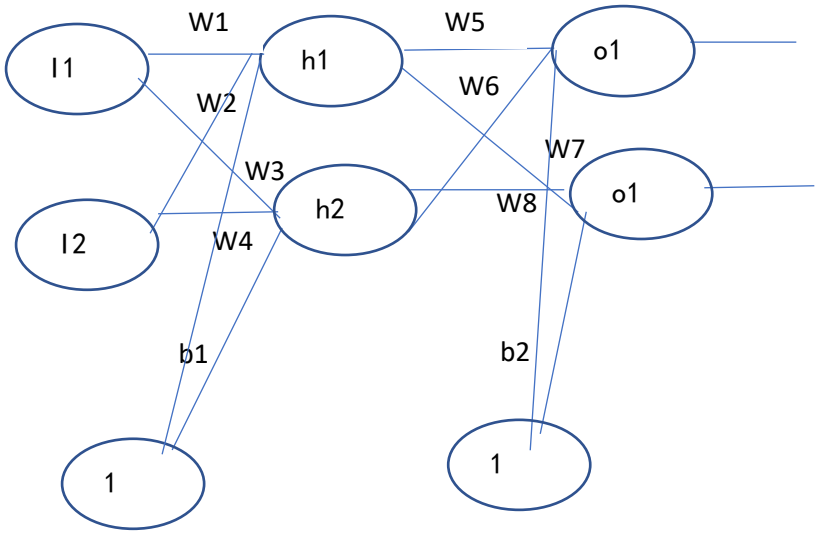
如上描述的，就是一个常见的无监督学习——数据聚类。

2.自编码器算法原理及程序实现

目前自编码器主要有数据降噪和为数据可视化而降维两个方面的应用。这里将设置的自编码器是具有一个隐藏节点的三层全连接神经网络，实现的是为数据可视化而降维这方面的应用。



下面我将构建一个简单的网络层来解释神经网络中的，如何计算前向传播与反向传播：



在这个网络层中有：

第一层输出层：里面包含神经元 i1，i，截距：b1,权重：w1,w2,w3,w4

第二层隐含层：里面包含 h1,h2，截距：b2,权重：w5,w6,w7,w8

第三层输出层：里面包含 o1,o2

我们使用 sigmoid 作为激活函数，假定我们输入数据 i1:0.02,i2:0.04 截距 b1:0.4,b2:0.7，期望的输出数据 o1:0.5,o2:0.9

未知的是权重 w1,w2,w3,w4,w5,w6,w7,w8

我们的目的是为了能得到 o1:0.5,o2:0.9 的期望的值，计算出 w1,w2,w3….w8

先假如构造一个权重 w1,w2,w3…w8 的值，通过计算获取到最佳的 w1,w2,w3…w8 的权重

权重的初始值：
w1=0.25,w2=0.25,w3=0.15,w4=0.20,w5=0.3,w6=0.35,w7=0.40,w8=0.35

2.1 前向传播

2.1.1 输入层到隐含层

NET(h1)=w1*i1+w2*i2+b1=0.25*0.02+0.25*0.04+0.4=0.415

神经元 h1 到输出 h1 的激活函数是 sigmoid

Out(h1)=1/(1+e^(-NET(h1)))=1/(1+0.660340281)=0.602286177

同理我们也可以获取 out(h2)的值

NET(h2)=w3*i3+w4*i2+b1=0.15*0.02+0.20*0.04+0.4=0.411

Out(h2)=1/(1+e^(-NET(h2)))=1/(1+0.662986932)=0.601327636

2.1.2 从隐含层到输出层

计算输出层的神经元 o1,o2 的值，计算方法和输出层到隐含层类似

NET(o1)=w5*h1+w6*h2+b2=0.4*0.602286177+0.35*0.601327636+0.7=1.091150525

Out(o2)=1/(1+e^(-NET(o2)))=1/1.316200383=0.759762733

O1:0.748598311 o2:0.759762733 距离我们期望的 o1:0.5,o2:0.9 还是有很大的距离

2.2 计算总误差

那么既然有如此大的距离，该如何计算总误差呢？

也就是说我们需要计算每个期望误差的和，有如下公式：

$$E_{\text{total}} = \sum \frac{1}{2}(\text{target} - \text{output})^2$$

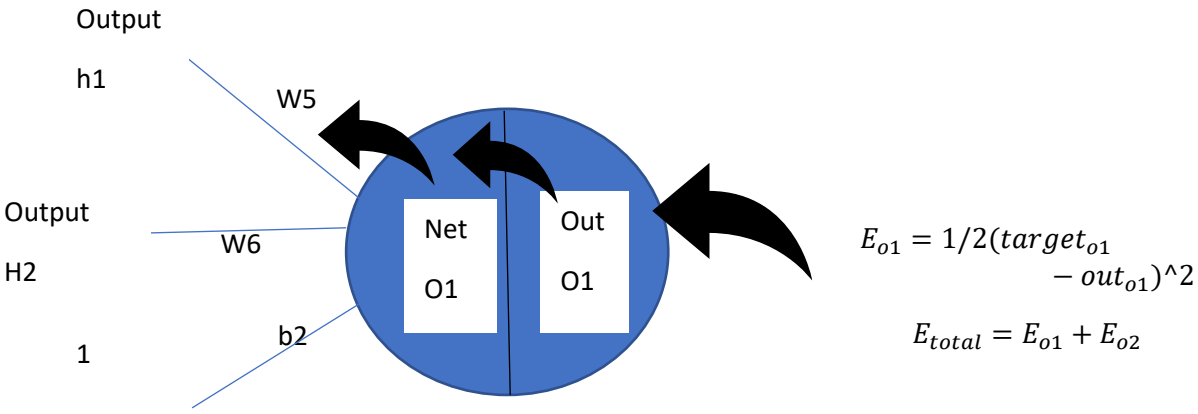
E(total)=E(o1)+E(o2)=(1/2)*(0.748598311-0.5)^2+(1/2)*(0.759762733-0.9)^2=0.025283526

2.3 反向传播

反向传播如何工作呢？反向传播 Backpropagation,简称为 backprop(简称为 BP)的工作过程如下：首先是创建一个含有一层或多层隐藏神经细胞的神经网络，并随机地为这些神经网络细胞的权重符值，例如赋给-1 和 1 之间的任意一个实数值，然后把一个输入模式馈送到网络的输入端，并考察网络的输出值。我们把这一输出值和要求达到的目标输出值之间的差称为误差值（error value）.利用这一误差值就可以用来调整来自输出底层下的那个层的输出的权重，使得当用同样得输出模式再次送人网络时，其输出能向正确答案靠近一些。一旦当前层得权重调节完毕，就可以为它前面得那个层来做同样得事情，等等，这样由输出层开始，一层一层地向输入层方向推，直到到达第一个隐藏层为止，使所有得权重都或者少量得调整，如果这一工作正确做完，则输入模式再一次被送入时，网络得实际输出将会向目标输出靠近一点，这样的全部过程要对所有不同

的输入模式重复进行许多次，知道误差值降到所处理问题可接受的一个极限值以内，这时我们可以说网络已经被训练好了。

前向传播计算完成后，需要计算反向传播，通过下图可以更加直观的看清楚误差的反向传播



2.3.1 隐含层到输出层的权值更新

隐含层到输出层的权值，在上面的例子里是 w5,w6,w7,w8

我们以 w6 参数为例子，计算 w6 对整体误差的影响有多大，可以使用整体误差对 w6 参数求偏导:

$$\frac{\partial \mathrm{E_{total}}}{\partial \mathrm{w6}}$$

很明显并没有 w6 对 Etotal 的计算公式，我们只有 w6 对 NET(o1)的计算公式，但根据偏导的链式法则，我们可以将我们存在的推导公式进行链式乘法

$$\frac{\partial E_{\mathrm{total}}}{\partial \mathrm{w6}} = \frac{\partial \mathrm{E_{total}}}{\partial \mathrm{OUT_{o1}}} * \frac{\partial \mathrm{OUT_{o1}}}{\partial \mathrm{NET_{o1}}} * \frac{\partial \mathrm{NET_{o1}}}{\partial \mathrm{w6}}$$

我们来计算每一个公式的偏导:

计算$\frac{\partial \mathrm{E_{total}}}{\partial \mathrm{OUT_{o1}}}$:

$$E_{total} = 1/2(\mathrm{target_{o1}} - \mathrm{out_{o1}})^2 + 1/2(\mathrm{target_{o2}} - \mathrm{out_{o2}})^2$$

$\frac{\partial E_{total}}{\partial OUT_{o1}}$=2*(1/2)*$(target_{o1}-out_{o1})$*(-1)+0

这是一个复合函数的导数

设 f 和 g 为两个关于 x 可导函数，则复合函数（f°g）（x）的导数
（f°g）’(x)=f’(g(x))g’(x)

这里 g(x)=target(o1)-out(o1) g’(x)-1

$\frac{\partial E_{total}}{\partial OUT_{o1}}$=-$(target_{o1}-out_{o1})$

=-(0.5-0.748598311)=0.248598311

计算$\frac{\partial OUT_{o1}}{\partial NET_{o1}}$

$$out_{o1}=\frac{1}{1+e^{-net_{o1}}}$$

$\frac{\partial OUT_{o1}}{\partial NET_{o1}}$=$out_{o1}(1-out_{o1})$

=0.748598311*(1-0.748598311)=o.18819888

计算$\frac{\partial NET_{o1}}{\partial w6}$

$$net_{o1}=w_5*out_{h1}+w_6*out_{h2}+b_2*1$$

也就是 net(o1)’=out(h2)=0.601327636

最后我们的公式

$$\frac{\partial NET_{o1}}{\partial w6}=-(target_{o1}-out_{o1})*out_{o1}(1-out_{o1})*out(h2)$$

=0.248598311*0.18819888*0.601327636

2.3.1.1 跟新 w6 的权重

w6=w6-x*$\frac{\partial NET_{o1}}{\partial w6}$

其中 x 就是我们常说的学习速率，设置 x 学习速率为 0.1，那么新的 w6 的权重就是

0.36-0.1*0.028133669=0.347186633

相同的道理，我们也可以计算新的 w5,w6,w7,w8 的权重

可是如何计算和跟新 w1,w2,w3,w4 的权重呢？

2.3.2 隐含层的权值跟新

大概的算法还是和前面类似，如下所示：

$$\frac{\partial E_{\text{total}}}{\partial \text{w1}} = \frac{\partial \text{E}_{\text{total}}}{\partial \text{OUT}_{h1}} * \frac{\partial \text{OUT}_{\text{h1}}}{\partial \text{NET}_{\text{h1}}} * \frac{\partial \text{NET}_{\text{h1}}}{\partial \text{w1}}$$

2.3.2.1 计算$\frac{\partial \text{E}_{\text{total}}}{\partial \text{OUT}_{h1}}$

对 out(h1)来说 Etotal 并不依赖 out(h1)计算，需要将 total 分拆成两个 Eo1 和 Eo2 来计算，公式如下：

$$\frac{\partial \text{E}_{\text{total}}}{\partial \text{OUT}_{h1}} = \frac{\partial \text{E}_{o1}}{\partial \text{OUT}_{h1}} + \frac{\partial \text{E}_{\text{o2}}}{\partial \text{OUT}_{h1}}$$

接着推导公式：

$$\frac{\partial \text{E}_{\text{total}}}{\partial \text{OUT}_{h1}} = \frac{\partial \text{E}_{\text{o1}}}{\partial \text{net}_{\text{o1}}} * \frac{\partial \text{net}_{\text{o1}}}{\partial \text{OUT}_{h1}}$$

计算$\frac{\partial \text{E}_{\text{total}}}{\partial \text{OUT}_{h1}}$

$\frac{\partial \text{E}_{\text{o1}}}{\partial \text{net}_{\text{o1}}} = \frac{\partial \text{E}_{\text{total}}}{\partial \text{OUT}_{h1}} * \frac{\partial \text{out}_{\text{o1}}}{\partial \text{net}_{\text{o1}}}$

$\text{net}_{\text{o1}} = \text{w}_5 * \text{out}_{\text{h1}} + \text{w}_6 * \text{out}_{\text{h2}} + \text{b}_2 * 1$

$\frac{\partial \text{out}_{\text{o1}}}{\partial \text{net}_{\text{o1}}} = \text{w}_5$

同理也可以计算$\frac{\partial E_{\text{o2}}}{\partial \text{out}_{\text{h1}}}$

2.3.2.2 计算$\frac{\partial \text{out}_{\text{h1}}}{\partial \text{net}_{h1}}$

$$\boldsymbol{out}_{\text{h1}} = \frac{1}{1 + \text{e}^{-\text{net}_{\text{h1}}}}$$

$$\frac{\partial \mathrm{out_{h1}}}{\partial \mathrm{net}_{h1}} = \mathrm{out_{h1}}(1 - \mathrm{out_{h1}})$$

2.3.2.3 计算$\frac{\partial \mathrm{net_{h1}}}{\partial w_1}$

$$\boldsymbol{net}_{\mathrm{h1}} = \mathrm{w_1} * \mathrm{i_1} + \mathrm{w_2} * \mathrm{i_2} + \mathrm{b_1} * 1$$

$$\frac{\partial \mathrm{net_{h1}}}{\partial \mathrm{w_1}} = \mathrm{i_1}$$

最后三者相乘:

$$\frac{\partial E_{\mathrm{total}}}{\partial \mathrm{w_1}} = \frac{\partial \mathrm{E_{total}}}{\partial \mathrm{out_{h1}}} * \frac{\partial \mathrm{out_{h1}}}{\partial \mathrm{net_{h1}}} * \frac{\partial \mathrm{net_{h1}}}{\partial \mathrm{w_1}}$$

2.3.2.4 整体公式

根据前面的公式，我们可以推导出最后的公式:

$$\frac{\partial E_{\mathrm{total}}}{\partial \mathrm{w_1}} = (\sum_0 \frac{\partial \mathrm{E_{total}}}{\partial \mathrm{OUT_0}} * \frac{\partial \mathrm{OUT_0}}{\partial \mathrm{NET_0}} * \frac{\partial \mathrm{NET_0}}{\partial \mathrm{out_{h1}}}) * \frac{\partial \mathrm{out_{h1}}}{\partial \mathrm{net_{h1}}} * \frac{\partial \mathrm{net_{h1}}}{\partial \mathrm{w_1}}$$

$$\frac{\partial E_{\mathrm{total}}}{\partial \mathrm{w_1}} = (\sum_{\mathrm{o}} \delta_0 * \mathrm{w_{h0}}) * \mathrm{out_{h1}}(1 - \mathrm{out_{h1}}) * \mathrm{i_1}$$

2.3.2.5 跟新 w1 的权重

和计算 w6 的权重一样:

$$\boldsymbol{w}_1^+ = \mathrm{w_1} - \boldsymbol{\mu} * \frac{\partial \mathrm{Etotal}}{\partial \mathrm{w1}}$$

设置学习速率，计算得到 w1 的权重值

2.4.计算获取最佳的权重

我们将获取的新的权重不停的迭代，迭代一定的次数后直到接近期望值 o1:0.5,o2:0.9 后，所得到权重 w1,w2…w8,就是所需要的权重

在自编码器的三层神经网络中，根据网络的输入前向计算网络中所有节点的值时，所用的激活函数为 sigmoid 函数:

$$f:sigmoid(x) = \frac{1}{1+e^{-x}}$$`取值范围是【0，1】，它的导数就是 $\dot{f}(x)=f(x)(1-f(x))$

故在前向计算过程中，神经网络中隐藏层节点的值为:

$$h_i = f\left(z_i^h\right)$$

$$z_i^h = \sum_{j=1}^{n} xj \cdot w_{j,i} + a \cdot w_{n+1,i}$$

输出层的节点值为:

$$\hat{x}_i = f\left(z_i^h\right)$$

$$z_i^{\hat{x}} = \sum_{j=1}^{m} hj \cdot w_{j,i}$$

自编码器网络计算实际输出与输入值之间的误差函数（Cost function）C 为:

$$c = \frac{1}{2}\sum_{i=1}^{n}\left(\hat{x}_i - x_i\right)^2$$

故在误差反向传播时，第l层的δ值为:

$$\delta_i^l = \dot{f}\left(z_i^l\right) \cdot \sum_{j=1}^{n_l+1} w_{ji}^l \cdot \delta_j^{l+1}$$

若按照学习过程（权值的更新方法）中，一次使用多少个样本对权值进行更新划分，可划分成:

1．离线学习/批梯度下降

2．在线学习/随机梯度下降

3．最小批度下降

批度下降是在所有的数据样本训练完毕之后，累加所有样本对权值的更新值，对权值进行一次更新，在训练过程中当有新的样本出现时，我们的权值也没有进行及时的更新，故也成为离线学习。随机梯度下降时每次只使用一个样本对权值进行更新，所以当新的样本出现时，我们也能及时更新权值，故也称为在线学习。

批度下降算法需要所有的样本进行一次权值更新，若是数据集中有很多样本时，每次更新所花费的时间就很多，而随机梯度下降虽比批梯度下降算法更新

权值速度快，但是由于更新得过于频繁，且单个样本所得到的代价函数比所有样本得到的代价函数更具有随机性，可能无法找到逼近与我们想要的恒等函数 $\boldsymbol{h_{w,b}(x) = x}$ 的最优解。故我们采用了一种折中的学习过程，mini-batch learning 最小批训练。即每次训练时，只用所有数据中的一部分数据对权值进行更新。

3.自编码器实现程序如下:

```
#导入模块
import scipy.io as scio
import numpy as np
import matplotlib.pyplot as plt
import random
```

前向传播计算

```
#w:权值矩阵
#a:神经网络内部节点
#x:神经网络外部节点
def feedforward(w,a,x):
#激活函数 sigmoid
    f = lambda s:1/(1+np.exp(-s))
#将网络内部及外部输入联合起来与权值矩阵进行加权叠加
#这里使用的 是矩阵运算使训练更加快捷
 w = np.array(w)
    temp = np.array(np.concatenate((a,x),axis=0))
    z_next = np.dot(w,temp)
    #返回计算的下一层神经元的计算结果
    #及未经过激活函数前的加权叠加结果
    return f(z_next),z_next
```

误差方向传播计算

```
#w:权值矩阵
#z:当前层的未经过激活函数前的神经元值
#delta_next:下一层的δ
def backprop(w,z,delta_next):
#sigmoid 激活函数
    f = lambda s:np.array(1/(1+np.exp(-s)))
    #激活函数 sigmoid 的导数
    df = lambda s:f(s)*(1-f(s))
    #误差反向传播计算上一层的δ并返回
    delta = df(z)*np.dot(w.T,delta_next)
    return delta
```

导入人脸数据，并进行归一化

```
DataSet = scio.loadmat('D:/yaleB_face_dataset.mat')
unlabeledData = DataSet['unlabeled_data']
dataset_size = 80#准备的无标签的人脸图片数据数量
unlabeled_data = np.zeros(unlabeledData.shape)
#利用 z-score 归一化方法归一数据
for i in range(dataset_size):
    tmp = unlabeledData[:,i]/255.
    unlabeled_data[:,i]=(tmp - np.mean(tmp))/np.std(tmp)
```

设置自编码器无监督训练参数，及神经网络结构

```
alpha = 0.5#学习步长
```

```
max_epoch =300#自编码器训练总次数
mini_batch =10#最小批训练时，每次使用 10 个样本同时进行训练
height =48#人脸数据图片的高度
width = 42#人脸数据图片的宽度
imgSize = height*width

#神经网络结构
hidden_node = 60#网络隐藏层节点数目
hidden_layer = 2
layer_struc = [[imgSize,1],
               [0,hidden_node],
               [0,imgSize]]
layer_num = 3#网络层次数目
#初始化无监督网络的权值
w = []
for l in range(layer_num-1):
    w.append(np.random.randn(layer_struc[l+1][1],sum(layer_struc[l])))
#定义神经网络的外部节点数目
X=[]
X.append(np.array(unlabeled_data[:,:]))
X.append(np.zeros((0,dataset_size)))
X.append(np.zeros((0,dataset_size)))
#初始化在网络训练中，进行误差方向传播所需的δ
delta = []
for l in range(layer_num):
    delta.append([])
```

从准备好的人脸数据中显示其中每个人的一张图片，原始图片将训练过程中自编码的输出结果展示在同一面板

```
#定义结果展示参数
```

```
nRow = max_epoch / 100 + 1
nColumn =4
eachFaceNum = 20 #对于每个人都有 20 张未标记图像数据
#在第一行中展示原始图像
for iImg in range(nColumn):
    ax = plt.subplot(nRow,nColumn,iImg + 1)
    plt.imshow(unlabeledData[:,eachFaceNum *iImg +1].reshape((width,height)).T,cm
ap = plt.cm.gray)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

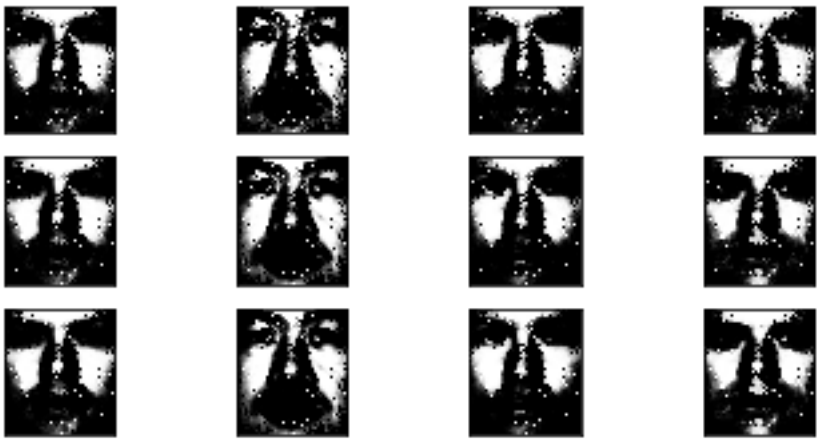
开始人脸图像自编码器的训练过程:

```
#无监督训练
count = 0#记录训练次数
print('Autoencoder training start..')
for iter in range (max_epoch):
#定义随机洗牌下标
    ind = list(range(dataset_size))
    random.shuffle(ind)
    a = []#网络内部神经元
    z = []#网络节点的加权叠加结果
    z.append([])

#对神经网络开始训练
    for i in range(int(np.ceil(dataset_size /mini_batch))):
        a.append(np.zeros((layer_struc[0][1],mini_batch)))
```

```
        x = []
        for l in range(layer_num):
            x.append(X[l][:,ind[i*mini_batch : min((i+1)*mini_batch,dataset_size)]])

#定义目标输出
        y= unlabeled_data[:,ind[i*mini_batch:min((i+1)*mini_batch,dataset_size)]]
#调用前向计算函数计算每一层节点的值
 for l in range(layer_num-1):
            a.append([])
            z.append([])
            a[l+1],z[l+1] = feedforward(w[l],a[l],x[l])

#根据最小二乘代价函数计算最后一层的δ
#即自编码器的实际输出与目标值的欧式距离
        delta[layer_num-1] = np.array(a[layer_num-1]-y) * np.array(a[layer_num -1])
        delta[layer_num-1] = delta[layer_num-1] * np.array(1-a[layer_num -1])
#误差反向传播过程
#调用 backprop 函数逐层反向计算δ的值
        for l in range(layer_num-2,0,-1):
            delta[l] = backprop(w[1],z[1],delta[l+1])

        for l in range(layer_num-1):
            dw = np.dot(delta[l+1],np.concatenate((a[l],x[l]),axis=0).T) / mini_batch
            w[l] = w[l] - alpha * dw
    count = count +1


 #每训练 100 次展示一次自编码器目前对原始图像的输出结果
    if np.mod(iter+1,100) == 0 :
        b=[]
```

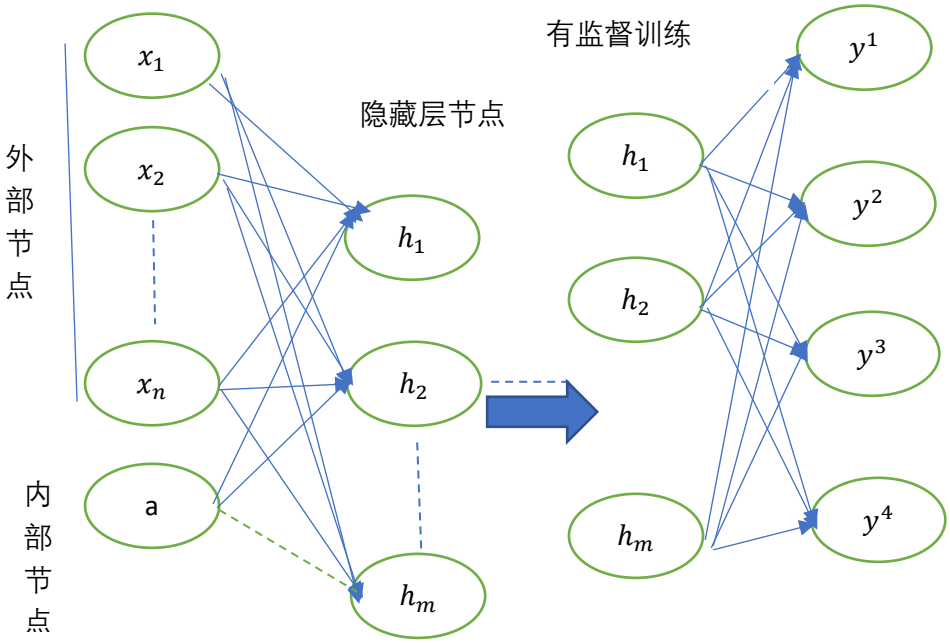
```
        b.append(np.zeros((layer_struc[0][1],dataset_size)))


        for l in range(layer_num-1):
            tempA,tempZ = feedforward(w[l],b[l],X[l])
            b.append(tempA)
        for iTmg in range(nColumn):
            ax = plt.subplot(nRow,nColumn,iTmg +nColumn *(iter+1) /100 + 1)
            tmp = b[layer_num -1][:,eachFaceNum *iTmg +1]
            dis_result = ((tmp *np.std(tmp))+np.mean(tmp)).reshape(width,height).T
            plt.imshow(dis_result,cmap=plt.cm.gray)
            ax.get_xaxis().set_visible(False)
            ax.get_yaxis().set_visible(False)
        print('Learning epoch:',count,'/',max_epoch)
```

```
Autoencoder training start..
Learning epoch: 100 / 300
Learning epoch: 200 / 300
Learning epoch: 300 / 300
```



以上为自编码器的实现，对数据进行降维

4.有监督训练总体网络结构



有监督训练程序实现：

设置有监督网络结构及参数

```
from __future__ import division
#定义有监督学习参数
supervised_alpha = 0.3#步长


max_epoch = 10000#训练次数
mini_batch =14
SJ = []#用于保存训练过程的误差变化
SAcc = []#用于记录训练过程中分类器预测的正确率变化过程
#初始化有监督升级网络结构
SL_layer_srtuc = []
SL_layer_num = 2
SL_layer_struc = [[0,hidden_node],[0,4]]
#初始化有监督训练部分神经网络的权值
supervised_weight = []
```

```
for l in range(SL_layer_num-1):
    supervised_weight.append(np.random.randn(SL_layer_struc[l+1][1],sum(SL_layer_struc[l])))


#用于有监督学习的误差调整时的 delta
delta = []
for l in range(SL_layer_num):
    delta.append([])
```

导入训练集和测试集数据，及其对应的标记数据，并对数据进行归一化

```
#训练集数据
trainData = DataSet['trainData']
trainData = trainData[:,:]
train_data = np.zeros(trainData.shape)
#归一化处理
trainData_size = 56
for i in range(trainData_size):
    tmp = trainData[:,i]
    train_data[:,i] = (tmp - np.mean(tmp)) / np.std(tmp)
train_labels = DataSet['train_labels']
train_labels = train_labels[:,:]
#测试集数据
testData = DataSet['testData']
testData = testData[:,:]
test_data = np.zeros(testData.shape)
#归一化处理
testData_size =40
for i in range(testData_size):
    tmp = testData[:,i]
```

```
    test_data[:,i] = (tmp-np.mean(tmp)) / np.std(tmp)


test_labels = DataSet['test_labels']
test_labels = test_labels[:,:]
```

利用自编码器获取训练集和测试集降维后的编码:

```
#训练集
a =[]
a.append(np.zeros((layer_struc[0][1],trainData_size)))
for l in range(hidden_layer-1):
    if l==0:
        tmpA,tmpZ = feedforward(w[l],a[l],train_data)
        a.append(tmpA)
    else:
        tmpA,tmpZ = feedforward(w[l],a[l],np.zeros((0,trainData_size)))
        a.append(tmpA)
small_trainData = a[hidden_layer-1]
#测试集
a =[]
a.append(np.zeros((layer_struc[0][1],testData_size)))
for l in range(hidden_layer-1):
    if l==0:
        tmpA,tmpZ = feedforward(w[l],a[l],test_data)
        a.append(tmpA)
    else:
        tmpA,tmpZ = feedforward(w[l],a[l],np.zeros((0,testData_size)))
        a.append(tmpA)
small_testData = a[hidden_layer-1]
```

有监督训练过程：

```
def supervised_training():
    global SJ
    global SAcc
    print('Supervised training start..')
    for iter in range(max_epoch):
#随机从数据集中挑选数据进行训练
        ind = list(range(trainData_size))
        random.shuffle(ind)


        a=[]
        z =[]
        z.append([])
        for i in range(int(np.ceil(trainData_size / mini_batch))):
#网络内部节点
            a.append(small_trainData[:,ind[i*mini_batch:min((i+1)*mini_batch,trainData
_size)]])
#准备网络训练的外部节点
            x=[]
            for l in range(SL_layer_num):
                x.append(np.zeros((0,min((i+1)*mini_batch,trainData_size)-i*mini_batc
h)))
            #网络的目标输出
            y=train_labels[:,ind[i*mini_batch:min((i+1)*mini_batch,trainData_size)]]
#进行网络的前向传播计算
            for l in range(SL_layer_num-1):
                a.append([])
                z.append([])
                a[l+1],z[l+1] = feedforward(supervised_weight[l],a[l],x[l])
#计算网络输出与目标输出的误差值
```

```
        delta[SL_layer_num-1] = np.array(a[SL_layer_num-1] -y)*np.array(a[SL_lay
er_num-1])
        delta[SL_layer_num-1] = delta[SL_layer_num-1]*np.array(1-a[SL_layer_num
-1])


#进行误差反向传播
#调用 backprop 函数逐层反向计算δ值
        for l in range(SL_layer_num-2,0,-1):
            delta[1] = backprop(supervised_weight[l],z[l],dalta[l+1])


         for l in range(SL_layer_num-1):
            dw = np.dot(delta[l+1],np.concatenate((a[l],x[l]),axis=0).T) / mini_batch
            supervised_weight[l] = supervised_weight[l] - supervised_alpha*dw


        tmpResult = a[SL_layer_num-1]
        SJ.append(np.sum(np.multiply(tmpResult[:] - y[:],tmpResult[:]-y[:]))/2/mini_b
atch)
        SAcc.append(float(sum(np.argmax(y,axis=0) == np.argmax(tmpResult,axis=
0))/mini_batch))
 #展示训练过程的预测正确率和误差值变化
    print('Supervised learning done!')
    plt.figure()
    plt.plot(SJ)
    plt.title('loss function')
    plt.figure()
    plt.plot(SAcc)
    plt.title('Accuracy')
```

测试有监督训练的结果，输出其对于训练集和测试集的预测正确率:

```
def supervised_testing():
    print('Testing..')
```

```
    #测试对训练集的预测结果正确率
    tmpA,tmpZ = feedforward(supervised_weight[0],small_trainData,np.zeros((0,train
Data_size)))
    train_pred = np.argmax(tmpA,axis =0)
    train_res = np.argmax(train_labels,axis=0)


    train_acc = float(sum(train_pred == train_res) / trainData_size)*100
    print('Training accuracy:%.2f%c'%(train_acc,'%'))
#测试对预测集的预测结果正确率
    tmpA,tmpZ = feedforward(supervised_weight[0],small_testData,np.zeros((0,testDa
ta_size)))
    test_pred = np.argmax(tmpA,axis=0)
    test_res = np.argmax(test_labels,axis=0)
    test_acc = float(sum(test_pred == test_res) / testData_size)*100
    print('Testing accuracy:%.2f%c'%(test_acc,'%'))
```

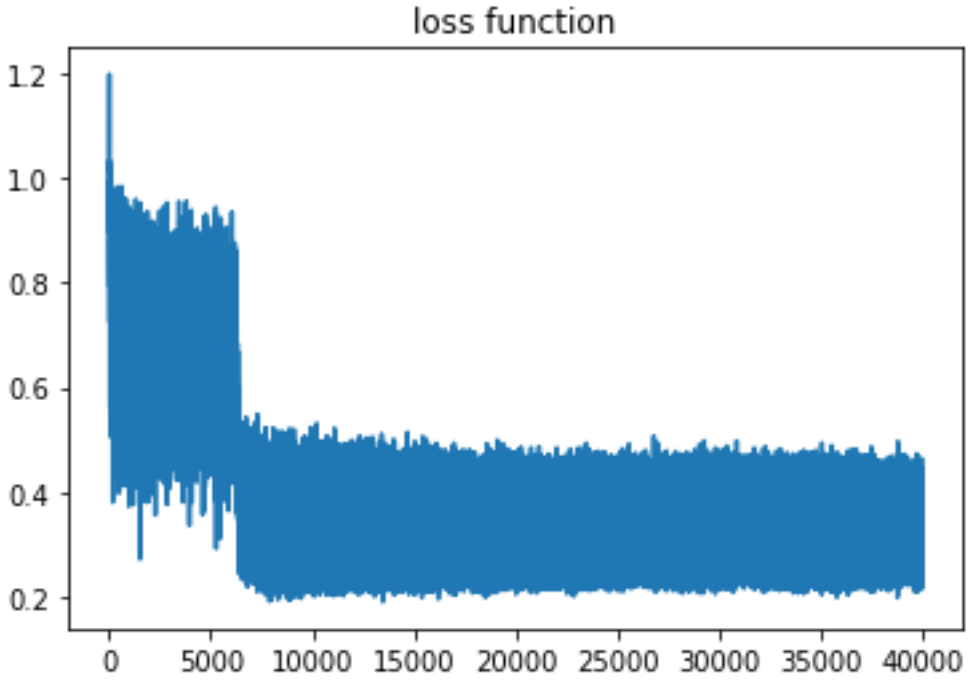
In [13]:

```
supervised_training()
supervised_testing()
plt.show()
```

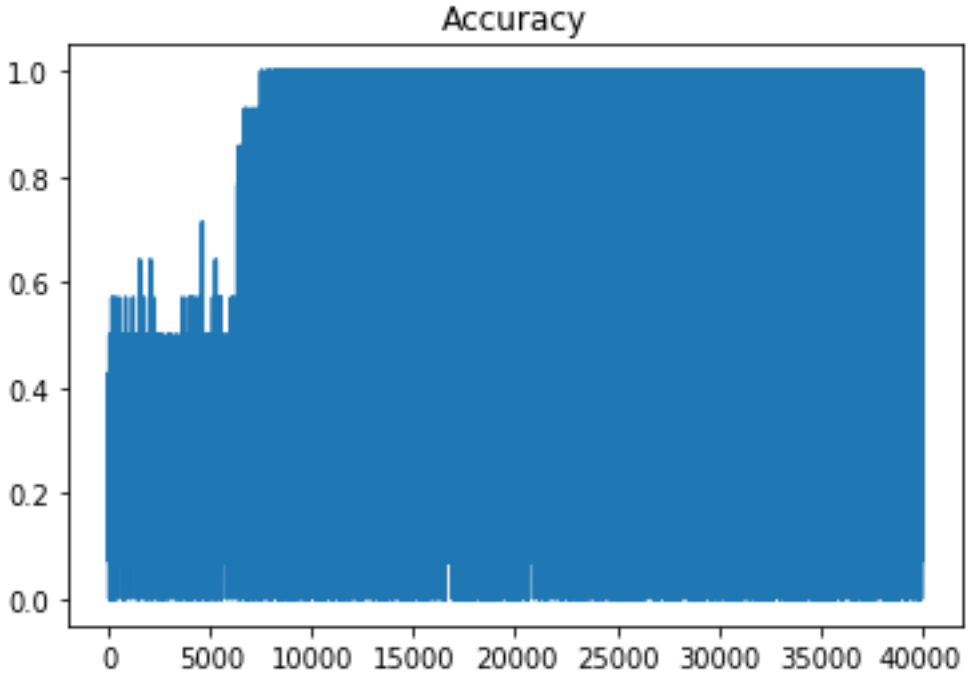
```
Supervised training start..
Supervised learning done!
Testing..
Training accuracy:100.00%
Testing accuracy:67.50%
```

最终我们得到训练集准确率百分之百，测试机准确率 67.50%，但是每次训练的到的结果中训练集准确率基本保持 98%以上，但是测试集准确率有高有低，随机性很大，可能是测试机数据太少的原因。

随着训练次数的增加，分类器的实际输出与目标数据间误差的变化：



随着训练次数的增加，分类器预测正确率的变化：



三．小组分工

程序设计及编写：孔子庆

程序调试：孔子庆

实验报告：孔子庆

四．作业总结

在这次人脸识别作业中，采用了无监督学习及有监督学习进行了神经网络的训练，在训练时对隐藏层加上稀疏限制搭建编码器，达到对原始数据进行降维后稀疏表达。可以看到训练效果不是特别的好，通过多次调试及查阅资料，总结原因有如下几点：

1. 网络参数的选择，比如学习步长，网络节点数目的设置，结构设置，批训练中一次训练多少图片等都会影响自编码器的性能，在本次实验中，当我增加迭代次数越多，自编码的结果越好，但是当训练到一定次数之后，训练结果就不会随这迭代次数的增多而出现太大的变动。于是通过查阅资料发现目前这些参数的设置大多数都靠经验，或者启发式的设置，因此在设置好网络后，很多时候都需要花大量的时间调参以获得更好的结果。

2. 本次实验采用的数据量很小，仅有 80 张无标记的数据进行特征提取，且有监督阶段也仅采用了 14 张有标记图片，而现在的神经网络应用中，其实是有大量的无标记数据的，及有专门筛选标记数据的数据库，在数据缺乏的情况下，预测的准确率也会大打折扣。

3. 在本次实验中，对于结果不太满意，以我浅薄的知识来猜想，是否存在过拟合，比如当我这个模型训练久了，用来区分猫和狗，假如长得像狗的猫和长的像猫的狗也被正确区分了，是否这时就出现了过拟合，那这样的模型在遇到新的数据时，错误分类的可能性是否会增大，由于知识有限，以后再继续研究。

附：文件说明

本次附件一共包含有：

1 大作业报告；

2 最终的 Python 实现程序源码及运行结果：神经网络面部识别.ipynb