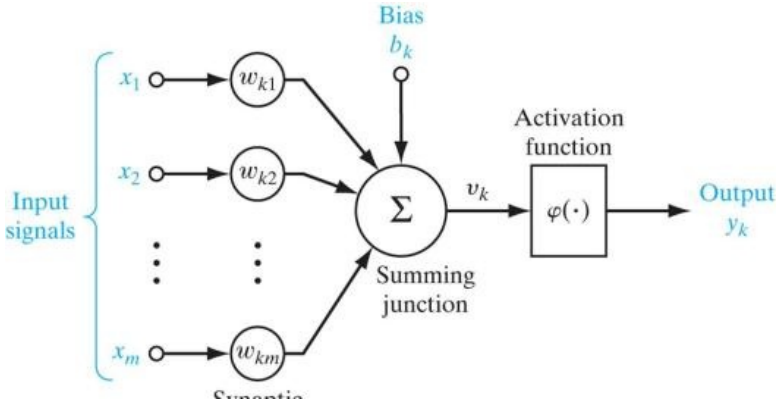
Python实现人工神经网络

1..人工神经网络简介

人工神经网络（Artificial Neural Networks，ANN），是模拟生物神经网络进行信息处理的一种数学模型。



人工神经元模型

人工神经元是人工神经网络操作的基本信息处理单位。它是人工神经网络的设计基础，一个人工神经元对输入信号X=[x1,x2,x3...xm]T的输出为y。其中Activation function（激活函数）有三种形式：阶梯函数、分段线性函数、非线性转移函数、Relu函数。

人工神经网络的学习也成为训练，指的是神经网络在受到外部环境的刺激下调整神经网络的参数，使神经网络以一种新的方式对外部环境做出反应的一个过程。在分类与预测中，人工神经网络主要用于有指导的学习方式，即根据给定的训练样本，调整人工神经网络的参数以使网络输出接近于一直的样本类标记或其他形式的因变量。

在人工神经网络的发展中，没有一种学习规则适用于所有网络结构和具体问题。在分类与预测中，学习规则（误差校正学习算法）是最广泛的一种，它根据神经网络的输出误差对神经元的连接强度进行修正，属于有指导学习。设神经网络中神经元i作为输入，神经元j为输出神经元，它们的连接权值为Wij，则对权值的修正为

$\Delta wij == \lambda \delta_j Y_i$，其中$\lambda$为学习率，即输出神经元j的实际输出和教师信号之差。神经网络训练是否完成常用误差函数（目标函数）R来衡量。当误差函数小于某一个设定的值时即停止神经网络的训练。

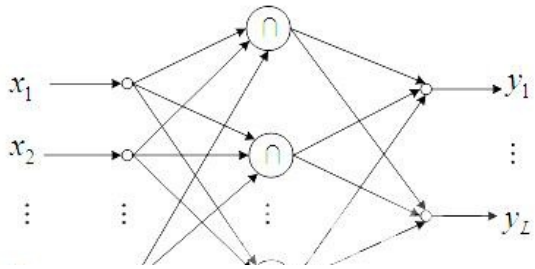
2.BP神经网络

是一种按误差逆传播算法训练的多层前馈网络。学习网络是δ学习规则，是目前应用最广泛的神经网络模型之一。

BP神经网络的学习算法是δ学习规则。

反向传播算法的特征是利用输出后的误差来估计输出层的直接前导层的误差，再用这个误差估计更前一层的误差，如此一层一层的反向传播下去，就获得了所有其他各层的误差估计，就形成了将输出层表现出的误差沿着与输入传送相反的方向逐级向网络的输入层传递的过程。

BP算法的学习过程由信号的正向传播与误差的逆向传播两个过程组成。正向传播时，输入信号经过隐层处理后，传向输出层。若输出层节点未能得到期望的输出，则转入误差的逆向传播阶段，将输出误差按某种子形式，通过隐层向输入层返回，并分摊给隐层4个节点与输入层x1，x2，x3三个输入节点，从而获得各层单元的参数误差或称为误差信号，作为修改各单元权值的依据。这种信号正向传播与误差逆向传播的各层权矩阵的修改过程，是周而复始进行的。权值不断修改的过程，也就是网络的学习过程。



三层BP神经网络结构

算法开始后，给定学习次数上限，初始化学习次数为0，对权值和阈值赋予较小的随机数，一般在[-1,1]之间，属于样本数据，网络正向传播，得到中间层与输出层的值，比较输出层的值与教师信号值的误差，用误差函数E来判断误差是否小于误差上限，如不小于误差上限，则对中间层和输出层权值和阈值进行更新，更新的算法为δ学习规则。更新权值和阈值后，再次将样本数据作为输入，得到中间层与输出层的值，计算误差E是否小于上限。学习次数是否达到指定值，如果达到，则学习结束。

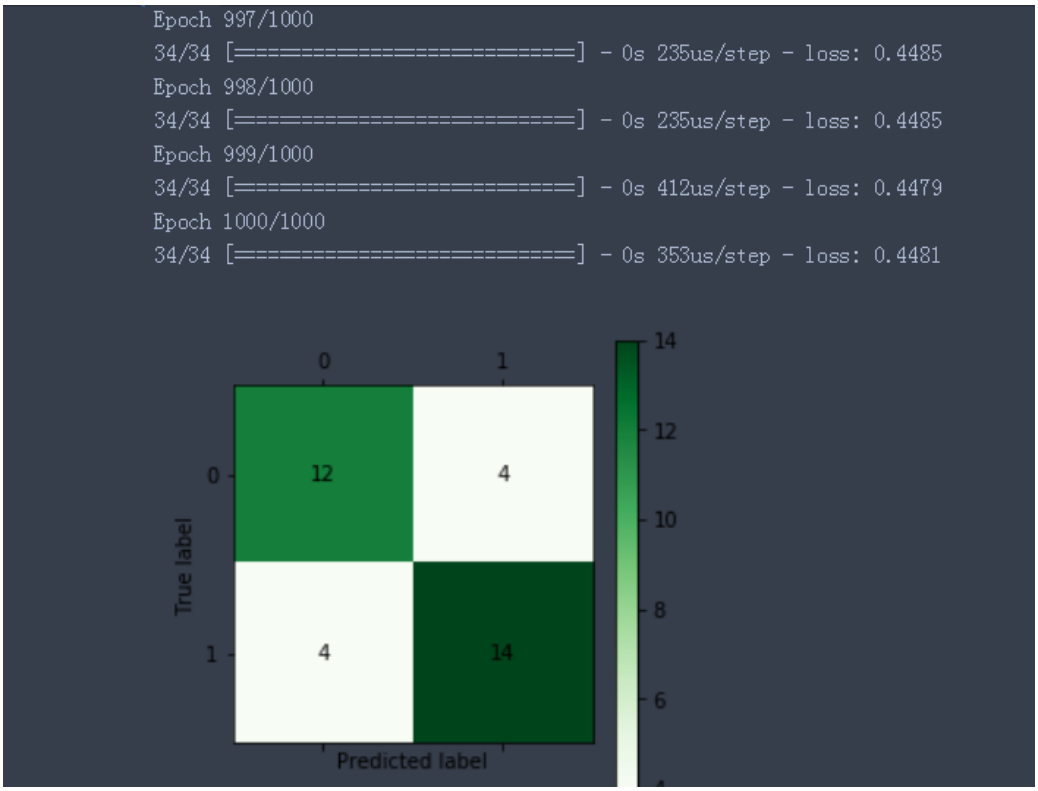
BP算法只用到均方误差函数对权值和阈值的一阶导数（梯度）的信息，使得算法存在收敛速度缓慢、易陷入局部极小等缺陷。

3.实例

我们建立神经网络算法进行建模，建立的神经网络有3个输入节点，10个隐藏节点和1个输出节点。

```
#-*- coding: utf-8 -*-
#使用神经网络算法预测销量高低

import pandas as pd

#参数初始化
inputfile = 'F:/fenxiyuwajue/dataandcode/chapter5/chapter5/demo/data/sales_data.xls'
data = pd.read_excel(inputfile, index_col = u'序号') #导入数据

#数据是类别标签，要将它转换为数据
#用1来表示“好”、“是”、“高”这三个属性，用0来表示“坏”、“否”、“低”
data[data == u'好'] = 1
data[data == u'是'] = 1
data[data == u'高'] = 1
data[data != 1] = 0
x = data.iloc[:,:3].as_matrix().astype(int)
y = data.iloc[:,3].as_matrix().astype(int)

from keras.models import Sequential
from keras.layers.core import Dense, Activation

model = Sequential() #建立模型
model.add(Dense(input_dim = 3, output_dim = 10))
model.add(Activation('relu')) #用relu函数作为激活函数，能够大幅提供准确度
model.add(Dense(input_dim = 10, output_dim = 1))
model.add(Activation('sigmoid')) #由于是0-1输出，用sigmoid函数作为激活函数

model.compile(loss = 'binary_crossentropy', optimizer = 'adam', class_mode = 'binary')
#编译模型。由于我们做的是二元分类，所以我们指定损失函数为binary_crossentropy，以及模式为binary
#另外常见的损失函数还有mean_squared_error、categorical_crossentropy等，请阅读帮助文件。
#求解方法我们指定用adam，还有sgd、rmsprop等可选

model.fit(x, y, nb_epoch = 1000, batch_size = 10) #训练模型，学习一千次
yp = model.predict_classes(x).reshape(len(y)) #分类预测
#混淆矩阵可视化函数
def cm_plot(y, yp):
    from sklearn.metrics import confusion_matrix #导入混淆矩阵函数
    cm = confusion_matrix(y, yp) #混淆矩阵

    plt.matshow(cm, cmap=plt.cm.Greens) #画混淆矩阵图，配色风格使用cm.Greens，更多风格请参考官网。
    plt.colorbar() #颜色标签
    for x in range(len(cm)): #数据标签
        for y in range(len(cm)):
            plt.annotate(cm[x,y], xy=(x, y), horizontalalignment='center', verticalalignment='center')
    plt.ylabel('True label') #坐标轴标签
    plt.xlabel('Predicted label') #坐标轴标签
    return plt
cm_plot(y,yp).show()
```

得出混淆矩阵结果如下：



由上图可以看出，检测样本为34个，预测正确的个数为26个，预测准确率为76.4%，预测准确率较低，原因是训练数据较少。

在本文案例中，并没有考虑过度拟合的问题，而神经网络的拟合能力较强，容易出现过拟合现象，与传统的添加“惩罚项”不同，神经网络防止过拟合的方法是随机低让部分神经网络节点休眠。