神经网络在二维数据集分类问题中的应用研究

绍兴市第一中学　韩程凯

【摘要】机器学习领域中，自动分类问题有着重要作用。人工神经网络经过训练能够有效拟合数据集中的数据分类规律，并对新的同源数据样本进行准确地自动分类。本文首先介绍人工神经网络的基本结构及算法流程，而后基于开源深度学习框架Tensorflow建立神经网络并对二维数据集进行自动分类，展示神经网络模型在机器自动分类问题中的实用性和高效性。

【关键词】机器学习；人工神经网络；自动分类；Tensorflow

1.引言

数据处理是计算机极其重要的应用领域，其中，基于计算机的数据自动分类操作在数据处理领域中始终起着基础性作用。数据样本在计算机中可以形式化地以特征向量表示，即：$S=(x_0,x_1,x_2,\ldots,x_{n-1})$，其中向量$S$为$n$维实值向量，其中$x_i,i\in\{0,\ldots n-1\}$为样本$S$的第$i$维特征。数据集为多个样本向量组成的集合，即：集合$D=\{S_0,S_1,\ldots,S_{l-1}\}$，表示含有$l$个样本的数据集合。在一些数据集合中，每一个样本都有一个类型标签c，c的取值范围决定数据集中样本的类型总数。针对此类数据集合，计算机能够利用该数据集合”训练”某种算法，使得该算法掌握数据集合中样本的分类规律，从而利用该算法对相同来源的新的未知类型的数据进行自动分类。如图1所示，二维平面上蓝色点和黑色点分属于两种不同类型，机器自动分类即是拟合出能够尽可能地准确区分两种类型数据的曲线，例如图1中红线Γ即为分类曲线。

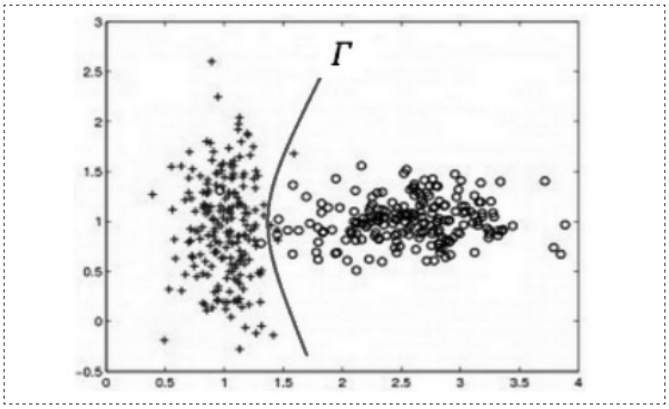

图1 二维数据点集及其分类曲线

机器自动分类算法属于机器学习范畴，诸多机器学习算法中，人工神经网络具有很强的非线性拟合能力，且具有学习规则简单和便于实现等优点，因此被广泛应用于多种数据分类，标注及自动控制等领域[1]。随着大数据以及高性能计算等领域的长足发展，神经网络及深度神经网络业已成为当前机器学习领域及人工智能领域的研究热点[2]。本文组织方式如下：第二节介绍人工神经网络的基本结构及其运算过程，同时介绍开源深度学习框架Tensorflow；第三节基于Tensorflow 搭建人工神经网络，并对二维数据集进行分类；第四节为文中实现的神经网络所具有的性能；第五节对本文结论进行阐述。

2.神经网络结构及Tensorflow框架

2.1 人工神经网络

人工神经网络由神经元以及神经元之间的连接组成[3]。 如图2所示，一个神经元有多条输入链路，并产生一个输出信号。每一条输入链路都传递由其他神经元或输入向量传递而来的信号，且每条输入链路都有相应的权重，以增强或减弱当前输入链路传递的神经信号。 神经元的接收端汇总各条输入链路的信号，而后通过激活函数产生 0到 1之间的输出信号，随后输出信号传递给后继的神经元。由此，神经元i所进行的运算可以形式化地归结为以下两步：

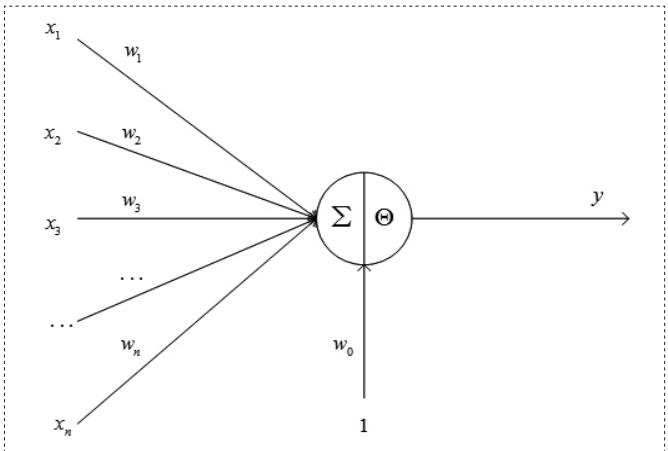

图2 单神经元示意图

加权和：$sum_i=\sum_0^{j-1}y_j*w_{ji}$

神经元输出：$y_i=\Theta(sum_i)$， 其中$\Theta(x)$为激活函数。

各个神经元中的激活函数是一个定义在全体实数集上的连续可导函数，值域为$(-1,1)$，其作用是增强神经网络的非线性拟合能力，使神经网络更加准确地拟合非线性特征。常用的激活函数有ReLU，sigmoid等函数。

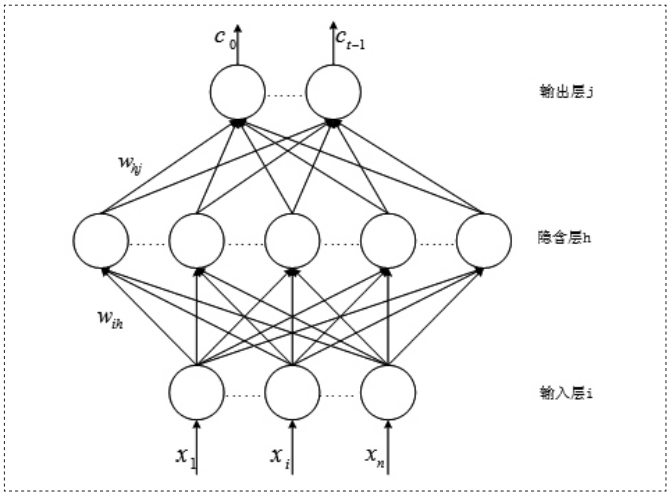

图3 具有3层神经元的神经网络结构图

图3展示一个具有3层神经元的人工神经网络结构。从下往上，最底部为输入层，输入层神经元与样本特征向量的各个维度一一对应。输入层神经元采用全连接方式连接隐含层神经元，即每一个输入层神经元都连接到所有的隐含层神经元。以同样方式，隐含层神经元与输出层神经元全连接。输出层神经元的数量取决于数据集的类型总数，即假设数据集D总共有t数据，则输出层神经元的数量为t。将所有输出信号排列在一起，形成输出向量 $\boldsymbol{T}'=(c_0, c_1, \ldots, c_{t-1})$，其中$c_j \in (0,1),\ j \in \{0,\ldots,t-1\}$。与此同时，对每一数据样本$S$的真实类型构造理论向量$T$：若样本$S$属于第$j$类，则向量$T$的第$j$维为1，其余所有维均为0。定义损失函数$\boldsymbol{L}(\boldsymbol{T}',\boldsymbol{T})$确定输出向量$\boldsymbol{T}'$与理论向量$T$的差异，以此判断人工神经网络的分类效果，并通过更新神经网络各链路的权重来减小损失函数。损失函数通常为平方差函数，即：

$$\boldsymbol{L}(\boldsymbol{T}',\boldsymbol{T})=\frac{1}{2}(\boldsymbol{T}'-\boldsymbol{T})^2$$

人工神经网络需要预先进行训练，即以事先分类的样本集(称为训练集)对适用于当前数据源的神经网络进行训练。训练过程中，通过调整神经网络所有链路的权重，尽可能地减小神经网络在训练集上损失函数的值。神经网络的初始权重设置为随机数，之后，每一批训练数据的训练过程分为前向传播和反向传播两个过程。

前向传播过程中， 每一个样本按照上文所述运算规则进行逐层计算，最终得到预测向量$\boldsymbol{T}'_t$以及关于该样本的损失函数值。因损失函数是向量$\boldsymbol{T}'$的函数，而向量$\boldsymbol{T}'_t$是神经网络的权重函数，因此通过求解偏导数的链式法则，可以反向的逐层计算所有权重的变化率(即梯度)，该过程称为反向传播过程。反向传播过程中，则某一边上权值w_{ji}可根据 $w_{ji} \leftarrow w_{ji}-\eta * \Delta w_{ji}$ 进行更新，其中Δw_{ji}为权值w_{ji}的变化率，η称为学习率（通常为小于1的正数）。神经网络反向传播的推导过程可参考文献[3]，此处不再赘述。经过多个训练样本及多轮前向-反向传播，神经网络在训练数据集上的总体损失函数会到达一个可以接受的范围。此时，神经网络的训练过程结束。

对于未标记分类的数据样本，使用神经网络的前向传播得到预测向量$\boldsymbol{T}'$，取预测向量中数值最大的分量所对应的类型作为该样本的分类结果。此外，可事先划分出一部分已分类数据作为验证数据集，用以检验神经网络的准确率。

2.2 Tensorflow 框架

Tensorflow 是谷歌公司开发的第二代神经网络开发框架[4]。该框架采用数据流图作为底层模型，数据流图的节点代表计算，节点之间的连接代表计算的前驱后继关系。顾名思义， Tensorflow 框架的操作对象为张量(“Tensor”)。从计算机编程的角度而言，张量可以简单理解为多维数组：0阶张量为标量；1阶张量为向量；2阶张量则为矩阵，以此类推。Tensorflow 通过其编程接口(API) 定义各种张量的基本运算，如矩阵-向量乘，矩阵-矩阵乘等基础运算。2.1节中神经网络的前向传播过程可以概括为矩阵-向量乘运算，如图4所示神经网络，h层的所有输入可以通过图4所示的矩阵-向量乘操作替代。针对神经网络的激活函数，误差函数及反向传播过程等要素，Tensorflow提供了简便的API，程序员只需几行代码即可通过Tensorflow实现相应的操作。

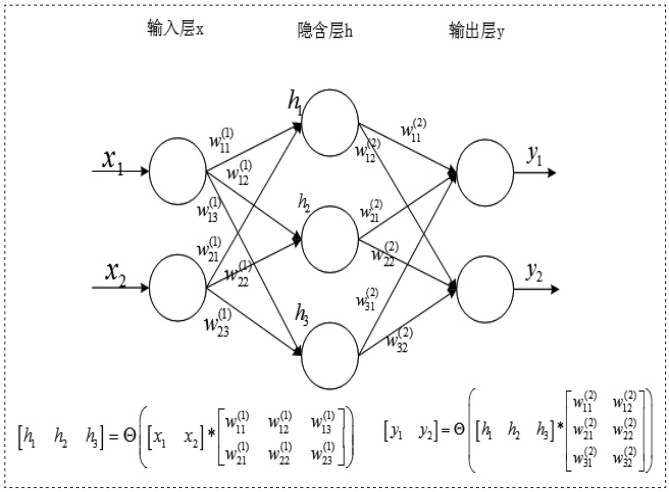

图4 包含矩阵运算的三层神经网络示意图

3.神经网络实现

程序列表1可生成图5所示的神经网络。每个节点使用sigmoid激活函数，设定损失函数为均方误差函数。如程序列表1使用Tensorflow模型能够极其便捷地产生当前神经网络。若需要增加神经网络的层数或某层神经元的数量，只需申请新的权值矩阵以及调整相应的矩阵维度即可。使用训练好的神经网络对测试数据集进行分类，sigmoid函数限定了神经网络的输出于区间(0, 1)内变化，因此，当神经网络输出大于0.5时，定义该样本类型为1，反之为0。

程序列表1 神经网络的训练与分类测试的python代码：

```
import tensorflow as tf
from numpy.random import RandomState
import numpy as np

"""The following codes are for model training"""
training_size = 128
testsample_size = 100
batch_size = 1
STEPS = input("Enter the iteration count: ")
pltSteps = []
pltMSE = []
with tf.Session() as sess:
    w1 = tf.Variable(tf.random_normal([2, 3], stddev=1, seed=1))
    w2 = tf.Variable(tf.random_normal([3, 1], stddev=1, seed=1))
    x  = tf.placeholder(tf.float32, shape=(None, 2), name="x-input")
    y_ = tf.placeholder(tf.float32, shape=(None, 1), name="y-input")

    a = tf.nn.sigmoid(tf.matmul(x, w1))
    y = tf.nn.sigmoid(tf.matmul(a, w2))
    squaredError = tf.reduce_sum((y_ - y) ** 2)
    train_step = tf.train.GradientDescentOptimizer(0.01).minimize(squaredError)

    rdm = RandomState(1)
    X = rdm.rand(training_size, 2)
    Y = [[int(x1 ** 2 < x2)] for (x1, x2) in X]
    init_op = tf.global_variables_initializer()
    sess.run(init_op)

    for i in range(STEPS):
        start = (i * batch_size) % training_size
        end = min(start + batch_size, training_size)
        sess.run(train_step, feed_dict={x:
X[start:end], y_: Y[start:end]})

        if i % 1000 == 0:
            total_cross_squaredError = sess.run(squaredError, feed_dict={x: X,
y_: Y})
            pltMSE += [total_cross_squaredError]
            pltSteps += [i]
            print("After %d training step(s) average squared error" \
                  "on all training data is %g" \
                  % (i, total_cross_squaredError))
"""The following part are for validation"""
    X_test = rdm.rand(testsample_size, 2)
    Y_test = [[int(x1 ** 2 < x2)] for (x1, x2) in X_test]
    x_tfTest = tf.placeholder(tf.float32, shape=(None, 2), name="xTest")
    y_tfTest = tf.placeholder(tf.float32, shape=(None, 1), name="yStd")
    a1 = tf.nn.sigmoid(tf.matmul(x_tfTest, w1))
    y_1 = tf.nn.sigmoid(tf.matmul(a1, w2))
    squaredError2 = tf.reduce_sum((y_tfTest - y_1) ** 2)
    y_result = sess.run(y_1, feed_dict={x_tfTest: X_test})
    total_cross_squaredError2 = sess.run(squaredError2, feed_dict={x_tfTest:
X_test, y_tfTest: Y_test})

    correctCnt = 0
    for i in range(testsample_size):
        if y_result[i][0] > 0.5 and Y_test[i][0] == 1:
            correctCnt += 1
        elif y_result[i][0] < 0.5 and Y_test[i][0] == 0:
            correctCnt += 1

    print("The squared error on all training data is %g" \
          % total_cross_squaredError2)
    print ("The accuracy = %g" % (float(correctCnt) / testsample_size))
```

4.实验结果分析

本文采用不可线性划分的数据集作为测试对象。设二维平面的随机点(x,y),$x \in (-1,1)$，$y > x^2$时，设置数据类型为1；$y < x^2$时，设置数据类型为-1。神经网络的目标是拟合出分离曲线$y = x^2$。文中训练样本集容量为1000，测试样本集容量为100。表1表明在测试数据集上，神经网络的分类准确率随着训练次数的增多而提高，在训练80000 次之后，该神经网络能够在98% 的准确率上对测试数据进行正确分类，充分说明该神经网络在当前实验条件下的实用性。图5为训练过程中损失函数（平方误差和）的变换规律，损失函数值随训练次数的增加，呈现单调下降趋势，充分表明随着训练次数增多，神经网络分类效果更好。

表1 测试集和准确率

实验次数	50000	10000	20000	40000	80000
准确率	67%	67%	83%	96%	98%

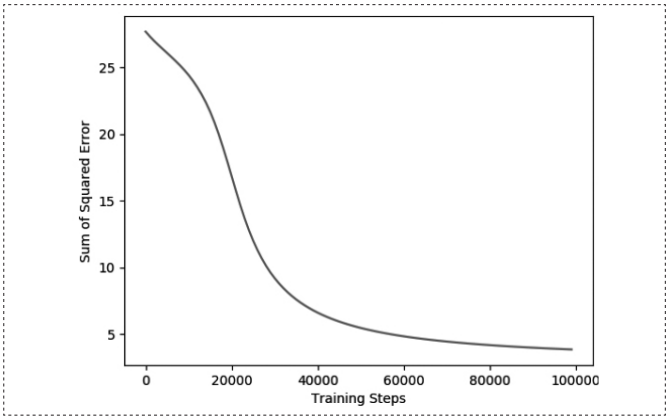

图5 损失函数在神经网络训练过程中的变化趋势

5.结论

本文首先介绍了机器自动分类操作的基本内容，并简要概括人工神经网络的组成要素。 实例中，基于Tensorflow框架，实现一个具有三层结构的人工神经网络，值得注意的是，基于相似方法，可以进一步构造层次更深，神经元数量更多的神经网络模型，充分说明了Tensorflow框架的实用性。本文合成非线性可分的数据集，用以验证示例神经网络模型的性能，实验结果表明当前神经网络对测试数据集的分类准确率高达90%以上，充分说明了神经网络模型具有较强的非线性拟合能力。

参考文献

[1]柴绍斌.基于神经网络的数据分类研究[D].大连理工大学,2007.

[2]吴岸城.神经网络与深度学习[M].电子工业出版社,2016.

[3]海金.神经网络与机器学习[M].机械工业出版社,2009.

[4]黄文坚,唐源.TensorFlow实战[M].电子工业出版社,2017.