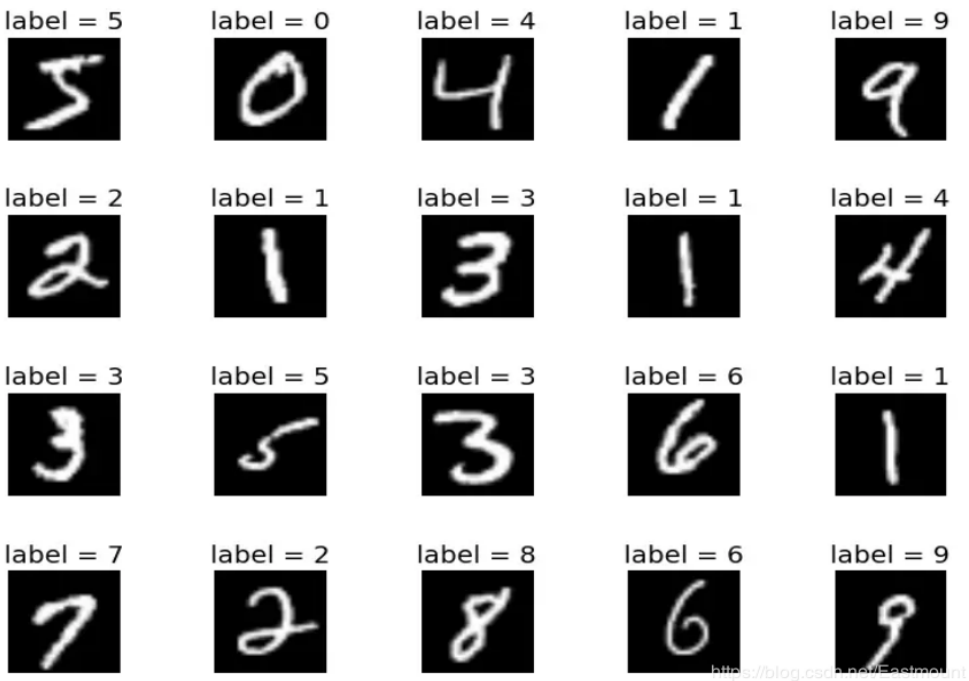
从本专栏开始，作者正式开始研究Python深度学习、神经网络及人工智能相关知识。前一篇文章讲解了Tensorboard可视化的基本用法，并绘制整个神经网络及训练、学习的参数变化情况；本篇文章将通过TensorFlow实现分类学习，以MNIST数字图片为例进行讲解。本文主要结合作者之前的博客、AI经验和"莫烦大神"的视频介绍，后面随着深入会讲解更多的Python人工智能案例及应用。

基础性文章，希望对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作者作为人工智能的菜鸟，希望大家能与我在这一笔一划的博客中成长起来，共勉。写了这么多年博客，尝试第一个付费专栏，但更多博客尤其基础性文章，还是会继续免费分享，但该专栏也会用心撰写，望对得起读者。



文章目录

同时推荐前面作者另外三个Python系列文章。从2014年开始，作者主要写了三个Python系列文章，分别是基础知识、网络爬虫和数据分析。2018年陆续增加了Python图像识别和Python人工智能专栏。

- Python基础知识系列：Pythonj基础知识学习与提升
- Python网络爬虫系列：Python爬虫之Selenium+Phantomjs+CasperJS
- Python数据分析系列：知识图谱、web数据挖掘及NLP

- Python图像识别系列：Python图像处理及图像识别
- Python人工智能系列：Python人工智能及知识图谱实战

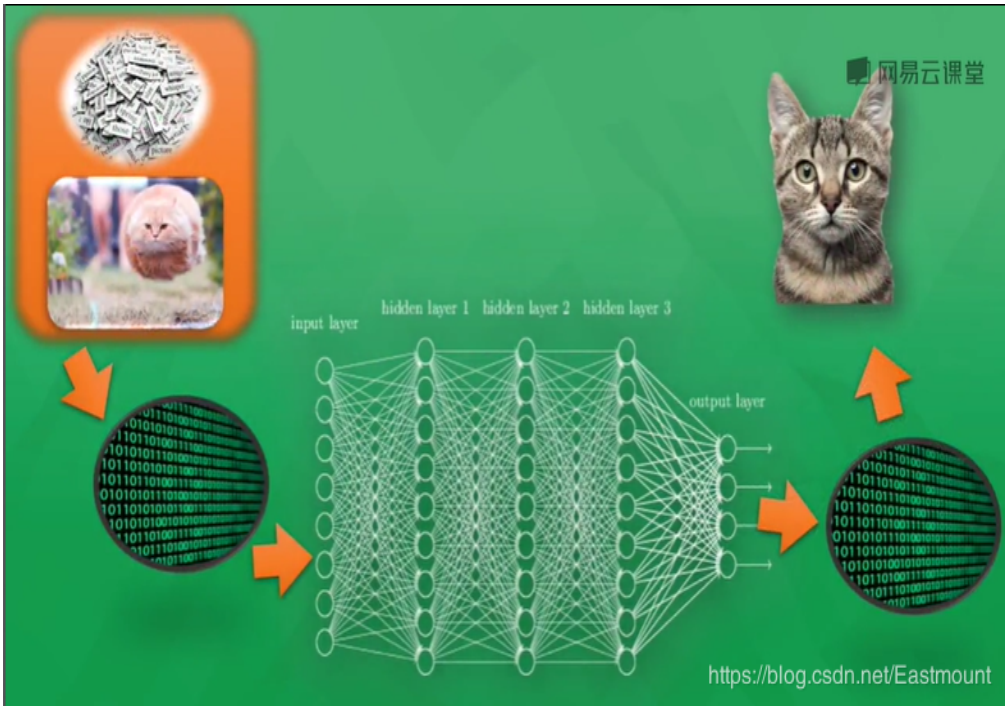
前文：
[Python人工智能] 一.TensorFlow2.0环境搭建及神经网络入门
[Python人工智能] 二.TensorFlow基础及一元直线预测案例
[Python人工智能] 三.TensorFlow基础之Session、变量、传入值和激励函数
[Python人工智能] 四.TensorFlow创建回归神经网络及Optimizer优化器
[Python人工智能] 五.Tensorboard可视化基本用法及绘制整个神经网络

代码下载地址：https://github.com/eastmountyxz/AI-for-TensorFlow

一.什么是分类学习

1.Classification

我们之前文章解决的都是回归问题，它预测的是一个连续分布的值，例如房屋的价格、汽车的速度、Pizza的价格等。而当我们遇到需要判断一张图片是猫还是狗时，就不能再使用回归解决了，此时需要通过分类学习，把它分成计算机能够识别的那一类（猫或狗）。

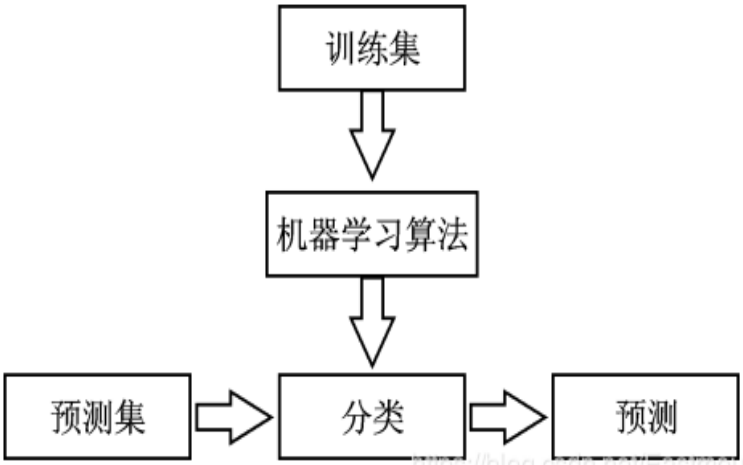


如上图所示，通常来说，计算机处理的东西和人类有所不同，无论是声音、图片还是文字，它们都只能以数字0或1出现在计算机神经网络里。神经网络看到的图片其实都是一堆数字，对数字的加工处理最终生成另一堆数字，并且具有一定认知上的意义，通过一点点的处理能够得知计算机到底判断这张图片是猫还是狗。

分类（Classification） 属于有监督学习中的一类，它是数据挖掘、机器学习和数据科学中一个重要的研究领域。分类模型类似于人类学习的方式，通过对历史数据或训练集的学习得到一个目标函数，再用该目标函数预测新数据集的未知属性。分类模型主要包括两个步骤：

- 训练。给定一个数据集，每个样本都包含一组特征和一个类别信息，然后调用分类算法训练模型。
- 预测。利用生成的模型对新的数据集（测试集）进行分类预测，并判断其分类结果。

通常为了检验学习模型的性能会使用校验集。数据集会被分成不相交的训练集和测试集，训练集用来构造分类模型，测试集用来检验多少类标签被正确分类。



那么，回归和分类有什么区别呢?
分类和回归都属于监督学习，它们的区别在于：回归是用来预测连续的实数值，比如给定了房屋面积来预测房屋价格，返回的结果是房屋价格；而分类是用来预测有限的离散值，比如判断一个人是否患糖尿病，返回值是“是”或“否”。也就是说，明确对象属于哪个预定义的目标类，预定义的目标类是离散值时为分类，连续值时为回归。

2.MNIST

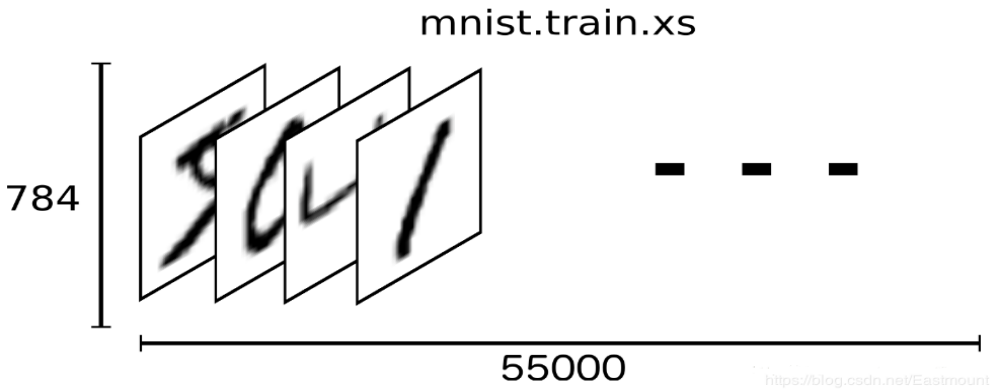
MNIST是手写体识别数据集，它是非常经典的一个神经网络示例。MNIST图片数据集包含了大量的数字手写体图片，如下图所示，我么可以尝试用它进行分类实验。

MNIST数据集是含标注信息的，上图分别表示数字5、0、4和1。该数据集共包含三部分：

- 训练数据集：55,000个样本，mnist.train
- 测试数据集：10,000个样本，mnist.test
- 验证数据集：5,000个样本，mnist.validation

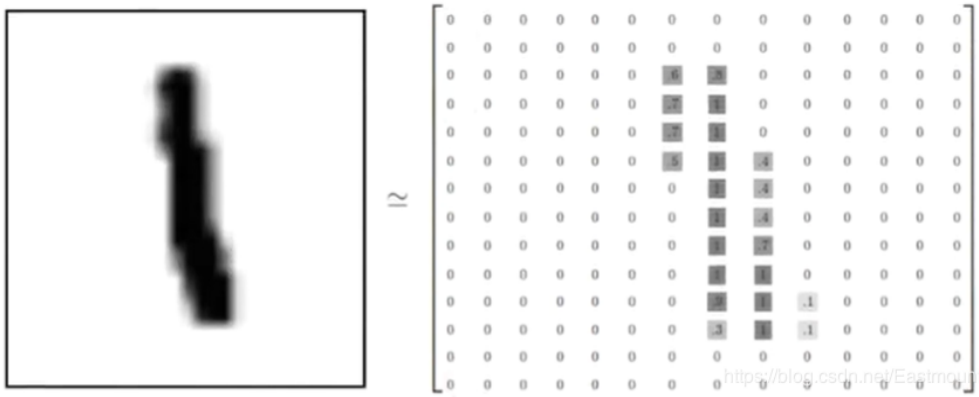
通常，训练数据集用来训练模型，验证数据集用来检验所训练出来的模型的正确性和是否过拟合，测试集是不可见的（相当于一个黑盒），但我们最终的目的是使得所训练出来的模型在测试集上的效果（这里是准确性）达到最佳。

如下图所示，数据是以该形式被计算机所读取，比如28*28=784个像素点，白色的地方都是0，黑色的地方表示有数字的，总共有55000张图片。



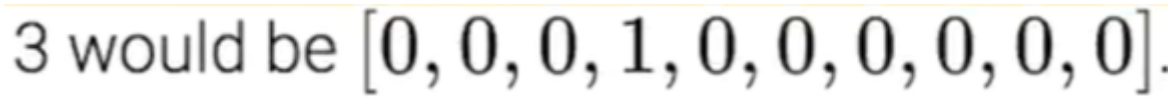
MNIST数据集中的一个样本数据包含两部分内容：手写体图片和对应的label。这里我们用xs和ys分别代表图片和对应的label，训练数据集和测试数据集都有xs和ys，使用mnist.train.images和mnist.train.labels表示训练数据集中图片数据和对应的label数据。

如下图所示，它表示由2828*的像素点矩阵组成的一张图片，这里的数字784（2828）*如果放在我们的神经网络中，它就是x输入的大小，其对应的矩阵如下图所示，类标label为1。

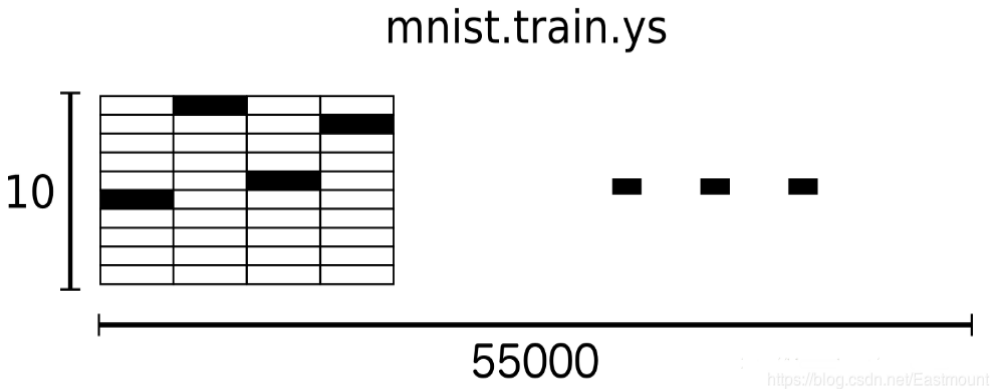


最终MNIST的训练数据集形成了一个形状为55000*784位的tensor，也就是一个多维数组，第一维表示图片的索引，第二维表示图片中像素的索引（tensor中的像素值在0到1之间）。

这里的y值其实是一个矩阵，这个矩阵有10个位置，如果它是1的话，它在1的位置（第2个数字）上写1，其他地方写0；如果它是2的话，它在2的位置（第3个数字）上写1，其他位置为0。通过这种方式对不同位置的数字进行分类，例如用[0,0,0,1,0,0,0,0,0,0]来表示数字3，如下图所示。



mnist.train.labels是一个55000*10的二维数组，如下图所示。它表示55000个数据点，第一个数据y表示5，第二个数据y表示0，第三个数据y表示4，第四个数据y表示1。



知道了MNIST数据集的组成，以及x和y具体的含义，我们就开始编写TensorFlow吧！

二.tensorflow实现MNIST分类

第一步，导入扩展包。

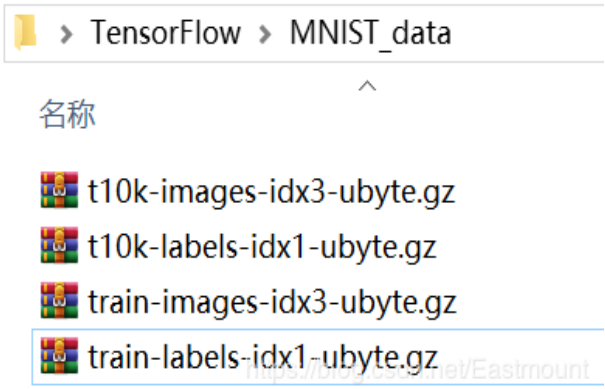
```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

第二步，下载数据集。

由于MNIST数据集是TensorFlow的示例数据，所以我们只需要下面一行代码，即可实现数据集的读取工作。如果数据集不存在它会在线下载，如果数据集已经被下载，它会被直接调用。

```
# 下载数据集 数字1到10
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

获取的数据如下图所示：



第三步，定义增加神经层的函数 add_layer()。

```
#---------------------------------定义神经层-------------------------------
# 函数：输入变量 输入大小 输出大小 激励函数默认None
def add_layer(inputs, in_size, out_size, activation_function=None):
    # 权重为随机变量矩阵
    Weights = tf.Variable(tf.random_normal([in_size, out_size]))    #行*
    # 定义偏置 初始值增加0.1 每次训练中有变化
    biases = tf.Variable(tf.zeros([1, out_size]) + 0.1)             #1行
    # 定义计算矩阵乘法 预测值
    Wx_plus_b = tf.matmul(inputs, Weights) + biases
    # 激活操作
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs
```

第四步，定义placeholder，用于传入值xs和ys至神经网络。

```
# 设置传入的值xs和ys
xs = tf.placeholder(tf.float32, [None, 784])  #每张图片28*28=784个点
ys = tf.placeholder(tf.float32,[None, 10])    #每个样本有10个输出
```

第五步，定义我们的输出层，其输入大小为784（代表像素点28*28），输出大小为10（代表0-9数字位置）。

```
# 输入是xs 784个像素点 10个输出值 激励函数softmax常用于分类
prediction = add_layer(xs, 784, 10, activation_function=tf.nn.softmax)
```

第六步，定义误差loss和训练。

计算loss，其值为真实值与预测值的误差。它的计算方法和之前的回归不太一样，这里使用cross_entropy算法。针对分类问题，如果softmax配合cross_entropy，它能实现一个比较好的分类效果。

```
#------------------------------定义loss和训练------------------------------
# 预测值与真实值误差 平均值->求和->ys*log(prediction)
cross_entropyloss = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction)
                    reduction_indices=[1]))  #loss
# 训练学习 学习效率通常小于1 这里设置为0.5可以进行对比
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entro
```

> 交叉熵（Cross Entropy）是Loss函数的一种，也称为损失函数或代价函数，用于描述模型预测值与真实值的差距大小。常见的Loss函数就是均方平方差（Mean Squared Error），交叉熵描述了两个概率分布之间的距离，当交叉熵越小说明二者之间越接近，这里的计算方式为：tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction)。

第七步，初始化操作。

```
# 定义Session
sess = tf.Session()
# 初始化
init = tf.initialize_all_variables()
sess.run(init)
```

第八步，神经网络分类学习。

```
#---------------------------------神经网络学习-----------------------------
for i in range(1000):
    # 提取一部分的xs和ys
    batch_xs, batch_ys = mnist.train.next_batch(100) #从下载好的数据集提取10
    # 训练
    sess.run(train_step, feed_dict={xs:batch_xs, ys:batch_ys})
    # 每隔50步输出一次结果
    if i % 50 == 0:
        # 计算准确度
```

```
        print(compute_accuracy(
                mnist.test.images, mnist.test.labels))
```

上述代码是从下载好的数据集中提取100个样本，优化器每次学习数据集中的这100个样本，而不像之前每次都学习所有的样本，那样会非常耗时，并且计算能力有限时造成阻碍。同时，100个样本迭代的学习效果不一定比每次学习所有样本的效果差。其核心代码如下：

- **batch_xs, batch_ys = mnist.train.next_batch(100)**

第九步，定义compute_accuracy()的功能。

mnist会分为train data（训练数据集）和test data（测试数据集），如果整个数据集拿去训练，会造成人为的误差，分好成两个独立的事件效果会更好。定义compute_accuracy()函数的代码如下：

```
#------------------------------定义计算准确度函数--------------------------
# 参数：预测xs和预测ys
def compute_accuracy(v_xs, v_ys):
    # 定义全局变量
    global prediction
    # v_xs数据填充到prediction变量中 生成预测值0到1之间的概率
    y_pre = sess.run(prediction, feed_dict={xs:v_xs})
    # 比较预测最大值(y_pre)和真实最大值(v_ys)的差别 如果等于就是预测正确,否则错误
    correct_prediction = tf.equal(tf.argmax(y_pre,1), tf.argmax(v_ys,1))
    # 计算正确的数量
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # 输出结果为百分比 百分比越高越准确
    result = sess.run(accuracy, feed_dict={xs:v_xs, ys:v_ys})
    return result
```

完整代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 17 20:28:55 2019
@author: xiuzhang Eastmount CSDN
"""
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

# 下载数据集 数字1到10
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

```
#---------------------------------定义神经层----------------------------------
# 函数：输入变量 输入大小 输出大小 激励函数默认None
def add_layer(inputs, in_size, out_size, activation_function=None):
    # 权重为随机变量矩阵
    Weights = tf.Variable(tf.random_normal([in_size, out_size]))    #行*
    # 定义偏置 初始值增加0.1 每次训练中有变化
    biases = tf.Variable(tf.zeros([1, out_size]) + 0.1)             #1行
    # 定义计算矩阵乘法 预测值
    Wx_plus_b = tf.matmul(inputs, Weights) + biases
    # 激活操作
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs

#---------------------------------定义计算准确度函数--------------------------
# 参数：预测xs和预测ys
def compute_accuracy(v_xs, v_ys):
    # 定义全局变量
    global prediction
    # v_xs数据填充到prediction变量中 生成预测值0到1之间的概率
    y_pre = sess.run(prediction, feed_dict={xs:v_xs})
    # 比较预测最大值(y_pre)和真实最大值(v_ys)的差别 如果等于就是预测正确,否则错误
    correct_prediction = tf.equal(tf.argmax(y_pre,1), tf.argmax(v_ys,1))
    # 计算正确的数量
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    # 输出结果为百分比 百分比越高越准确
    result = sess.run(accuracy, feed_dict={xs:v_xs, ys:v_ys})
    return result

#-------------------定义placeholder输入至神经网络--------------------------
# 设置传入的值xs和ys
xs = tf.placeholder(tf.float32, [None, 784])  #每张图片28*28=784个点
ys = tf.placeholder(tf.float32,[None, 10])    #每个样本有10个输出

#---------------------------------增加输出层----------------------------------
# 输入是xs 784个像素点 10个输出值 激励函数softmax常用于分类
prediction = add_layer(xs, 784, 10, activation_function=tf.nn.softmax)

#-----------------------------定义loss和训练----------------------------------
# 预测值与真实值误差 平均值->求和->平方(真实值-预测值)
cross_entropyloss = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction)
                    reduction_indices=[1]))  #loss
# 训练学习 学习效率通常小于1 这里设置为0.5可以进行对比
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entro
```

```
#----------------------------------初始化-------------------------------
# 定义Session
sess = tf.Session()
# 初始化
init = tf.initialize_all_variables()
sess.run(init)

#---------------------------------神经网络学习-----------------------------
for i in range(1000):
    # 提取一部分的xs和ys
    batch_xs, batch_ys = mnist.train.next_batch(100) #从下载好的数据集提取10
    # 训练
    sess.run(train_step, feed_dict={xs:batch_xs, ys:batch_ys})
    # 每隔50步输出一次结果
    if i % 50 == 0:
        # 计算准确度
        print(compute_accuracy(
                mnist.test.images, mnist.test.labels))
```

最终输出结果如下所示，最早预测的准确度结果非常低为7.45%，最后提升到了87.79%，由此可见TensorFlow的分类学习效果还不错。

```
wdir='C:/Users/xiuzhang/Desktop/TensorFlow')
Extracting MNIST_data\train-images-idx3-ubyte.gz
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz

0.0745
0.6235
0.7345
0.7753
0.8070
0.8239
0.8351
0.8406
0.8454
0.8554
0.8569
0.8605
0.8636
0.8651
0.8673
0.8698
0.8713
0.8744
```

```
0.8770
0.8779
```

三.总结

写到这里，这篇文章就结束了。本文主要通过TensorFlow实现了一个分类学习的案例，并详细介绍了MNIST手写体识别数据集。之前的文章是通过TensorFlow实现回归学习，其输出结果只有一个值，并且是连续的，比如房价；而本文介绍的分类学习是离散的数据，并且能输出多个值，比如猫（0）、狗（1），并且这是一个概率值，比如输出结果为：是猫的概率为0.21，是狗的概率为0.79，最终预测结果为狗。

最后，希望这篇基础性文章对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作为人工智能的菜鸟，我希望自己能不断进步并深入，后续将它应用于图像识别、网络安全、对抗样本等领域，指导大家撰写简单的学术论文，一起加油！

PS：这是作者的第一个付费专栏，会非常用心的去撰写，希望能对得起读者的9块钱。本来只想设置1快的，但CSDN固定了价格。写了八年的免费文章，这也算知识付费的一个简单尝试吧！毕竟读博也不易，写文章也花费时间和精力，但作者更多的文章会免费分享。如果您购买了该专栏，有Python数据分析、图像处理、人工智能、网络安全的问题，我们都可以深入探讨，尤其是做研究的同学，共同进步~

(By:Eastmount 2019-12-18 晚上9点夜于珞珈山 http://blog.csdn.net/eastmount/)

作者theano人工智能系列：
[Python人工智能] 一.神经网络入门及theano基础代码讲解
[Python人工智能] 二.theano实现回归神经网络分析
[Python人工智能] 三.theano实现分类神经网络及机器学习基础
[Python人工智能] 四.神经网络和深度学习入门知识
[Python人工智能] 五.theano实现神经网络正规化Regularization处理
[Python人工智能] 六.神经网络的评价指标、特征标准化和特征选择
[Python人工智能] 七.加速神经网络、激励函数和过拟合

参考文献：
[1] 神经网络和机器学习基础入门分享 - 作者的文章
[2] 斯坦福机器学习视频NG教授： https://class.coursera.org/ml/class/index
[3] 书籍《游戏开发中的人工智能》、《游戏编程中的人工智能技术》
[4] 网易云莫烦老师视频（强推 我付费支持老师一波）：
https://study.163.com/course/courseLearn.htm?courseId=1003209007
[5] 神经网络激励函数 - deeplearning
[6] tensorflow架构 - NoMorningstar

[7] 深度学习之 TensorFlow（二）： TensorFlow 基础知识 - 希希里之海
[8] Tensorflow实现CNN用于MNIST识别 - siucaan
[9] MNIST手写体识别任务 - chen645096127
[10] 机器学习实战—MNIST手写体数字识别 - RunningSucks
[11] https://github.com/siucaan/CNN_MNIST