TensorFlow初步学习总结

曲力 深圳基带部 2016.03

**一、准备工作**

1、Tensorflow的安装可以在网上自行搜索

首先在linux系统上安装tensorflow的docker镜像（docker镜像在WXCOP社区可以下载）。

常用的命令：

创建镜像：sudo docker run -i -t --name=quli\_tensorflow tensor\_flow

启动镜像：docker start quli\_tensorflow

停止镜像：docker stop quli\_tensorflow

查看当前运行镜像：docker ps

连接镜像：docker attach quli\_tensorflow

退出镜像：Ctrl+P+Q

从镜像中复制文件到本地：docker cp 18f3e1c0d2cb:/usr/lib/python2.7 /home/quli/mnist

从本地复制文件到镜像中：docker inspect -f '{{.Id}}' 18f3e1c0d2cb （查询出完整ID）

cp docker/docker-start.sh （Host本地目录）/var/lib/docker/aufs/mnt/（固定格式）a77a72ac178c1e35708d2af446197c10239b0b1bd8932104578e334b83eb93a2/root/（容器目录）

从windows向linux系统拷贝代码文件时需要用dos2unix.exe工具进行转换， 之后上传到FTP，再从FTP拷贝到linux本地。

包含三个场所：windows本地、linux本地、镜像内部

2、下载数据文件，放到本地目录中/tmp/mnist

'train-images-idx3-ubyte.gz'   训练集图片

'train-labels-idx1-ubyte.gz' 训练集标签

't10k-images-idx3-ubyte.gz' 测试集图片

't10k-labels-idx1-ubyte.gz'  测试集标签

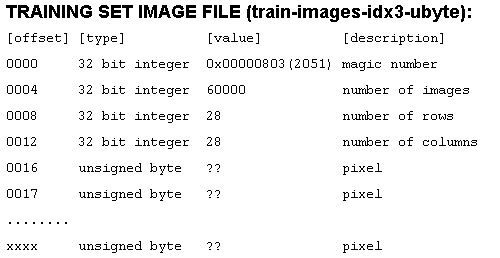
3、下载input\_data文件，放到python安装目录

注：input\_data版本有所不同，import失败的话可以换另一个版本的试一下

**二、手写数字识别实验**

基本思想，可以参考tensorflow的入门文档。

1. **理解mnist的数据结构及代码**



在train-images.idx3-ubyte中，第一个数为32位的整数（魔数，图片类型的数），第二个数为32位的整数（图片的个数），第三和第四个也是32为的整数（分别代表图片的行数和列数），接下来的都是一个字节的无符号数（即像素，值域为0~255），因此，为了输出其中的图片，我们只需要依次获取魔数和图片的个数，然后获取图片的长和宽，最后逐个像素读取就可以了。

代码如下：

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from PIL import Image

import struct

def read\_image(filename):

f = open(filename, 'rb')

index = 0

buf = f.read()

f.close()

magic, images, rows, columns = struct.unpack\_from('>IIII' , buf , index)

index += struct.calcsize('>IIII')

for i in range(images):

image = Image.new('L', (columns, rows))

for x in range(rows):

for y in range(columns):

image.putpixel((y, x), int(struct.unpack\_from('>B', buf, index)[0]))

index += struct.calcsize('>B')

print ('save ' + str(i) + 'image')

image.save(str(i) + '.png')

def read\_label(filename, saveFilename):

f = open(filename, 'rb')

index = 0

buf = f.read()

f.close()

magic, labels = struct.unpack\_from('>II' , buf , index)

index += struct.calcsize('>II')

labelArr = [0] \* labels

for x in range(labels):

labelArr[x] = int(struct.unpack\_from('>B', buf, index)[0])

index += struct.calcsize('>B')

save = open(saveFilename, 'w')

save.write(','.join(map(lambda x: str(x), labelArr)))

save.write('\n')

save.close()

print ('save labels success')

if \_\_name\_\_ == '\_\_main\_\_':

read\_image(r'E:\prac\t10k-images-idx3-ubyte\t10k-images.idx3-ubyte')

read\_label(r'E:\prac\t10k-labels-idx1-ubyte\t10k-labels.idx1-ubyte', r'E:\prac\label.txt')

注：首先打开文件，然后分别读取魔数，图片个数，以及行数和列数，在struct中，可以看到，使用了’>IIII’，这是什么意思呢？意思就是使用大端规则，读取四个整形数（Integer），如果要读取一个字节，则可以用’>B’（当然，这里用没用大端规则都是一样的，因此只有两个或两个以上的字节才有用）。

然后对于每张图片，先创建一张空白的图片，其中的’L’代表这张图片是灰度图，最后逐个像素读取，然后写进空白图片里，最后保存图片，就可以了

1. **手工制作图片验证训练模型**

Tensorflow使用的数据集都是二进制形式保存在文件中，将训练集中的数据（训练图片和对应标签）作为输入，经过迭代得到对于100个交叉熵的和来说最小的结果，即W和b，再将测试集中的数据（测试图片和对应标签）输入，每个图片都是一个像素矩阵x（28\*28），根据y=Wx+b计算出y值，判断y和y\_（标签矩阵）最大值的index是否一致（是则记为1，不是则记为0），再对这些值求平均，就可以算出正确的概率。

如果用手工制作图片验证模型，需要先得出W和b的值，再根据y=Wx+b算出y，再对y进行softmax，求出分别代表数字0-9的概率。

1）打印w和b值代码如下

#!/usr/bin/env python

import input\_data

import tensorflow as tf

import numpy as np

#导入Minst数据集

mnist = input\_data.read\_data\_sets("/tmp/mnist", one\_hot=True)

#输入变量，把28\*28的图片变成一维数组（丢失结构信息）

x = tf.placeholder("float", [None, 784])

#权重矩阵，把28\*28=784的一维输入，变成0-9这10个数字的输出

W = tf.Variable(tf.zeros([784,10]))

#偏置量矩阵

b = tf.Variable(tf.zeros([10]))

#核心运算，其实就是softmax（x\*w+b）

y = tf.nn.softmax(tf.matmul(x,W) + b)

#这个是训练集的正确结果，用于存放图像对应label的矩阵

y\_ = tf.placeholder("float", [None,10])

#交叉熵，作为损失函数

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y))

#梯度下降算法，最小化交叉熵

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)

#初始化，在run之前必须进行的

init = tf.initialize\_all\_variables()

#创建session以便运算

sess = tf.Session()

sess.run(init)

#迭代1000次

for i in range(1000):

#获取训练数据集的图片输入和正确表示数字

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

#运行刚才建立的梯度下降算法，x赋值为图片输入，y\_赋值为正确的表示数字

sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys})

#tf.argmax获取最大值的索引。比较运算后的结果和本身结果是否相同。

#这步的结果应该是[1,1,1,1,1,1,1,1,0,1...........1,1,0,1]这种形式。

#1代表正确，0代表错误

correct\_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y\_,1))

#tf.cast先将数据转换成float，防止求平均不准确。

#tf.reduce\_mean由于只有一个参数，就是上面那个数组的平均值。

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

#m就是W的值（784个list，每个list有10个值）

m = W.eval(sess)

#此处c并不是空值，需要将图像的灰度值矩阵输入

c = []

#将m组合成一个矩阵

w = np.matrix(m)

#p就是b的值（1个list，包含10个值）

p = b.eval(sess)

#将p组合成一个矩阵

B = np.matrix(p)

#直接计算y的softmax值，其中dot(c,W)代表矩阵相乘

result = tf.nn.softmax(np.dot(c,w) + B)

#计算c\*w+b

r = np.dot(c,w) + B

print result

print r

#选取测试集中的数据来验证概率

print sess.run(accuracy, feed\_dict={x: mnist.test.images, y\_: mnist.test.labels})

**2）手工制作图片的像素矩阵获取**

对于灰度图片：

from PIL import Image

import sys

im = Image.open(r'E:\prac\img\002.jpg')

width = im.size[0]

height = im.size[1]

print (width)

print (height)

count = 0

arr = []

for h in range(0, width):

for w in range(0, height):

pixel = im.getpixel((w, h))

arr.append(pixel)

print (arr)

print (len(arr))

fl = open(r'E:\prac\img\0002.txt','w+')

fl.write(str(arr))

对于彩色图片：

from PIL import Image

im = Image.open(r'E:\prac\img\09.jpg')

im\_L = im.convert('L') #进行灰度转换，将RGB值转换为灰度值

width = im\_L.size[0]

height = im\_L.size[1]

print (width)

print (height)

arr = []

for h in range(0, height):

for w in range(0, width):

pixel = im\_L.getpixel((w, h))

arr.append(pixel)

print (arr)

print (len(arr))

fl = open(r'E:\prac\img\09.txt','w+')

fl.write(str(arr))

3）通过对像素矩阵与训练的权重参数（w/b）计算y，然后进行softmax回归，得到某一数字的概率值

import math

def softmax():

x = [1,2,3,4,5,6,7,8,9]

max = 0.0;

sum = 0.0;

for i in range(0,len(x)):

if(max < x[i]):

max = x[i]

for i in range(0,len(x)):

x[i] = math.exp(x[i] - max)

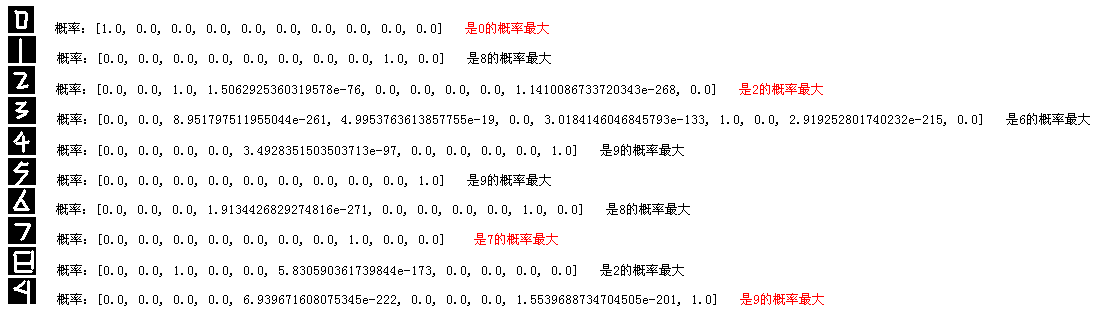
sum += x[i]

for i in range(0,len(x)):

x[i] /= sum

return x,sum

实验结果：



注：该模型对于mnist测试库图片的识别概率大概是91%左右，而对于自制图片，识别概率并不高，可能是制作图片本身的问题造成的。

# 三、卷积神经网络方法识别手写数字

卷积神经网络的原理还不是很理解，但是这个方法的正确率可以达到99%以上，缺点是运行耗时很长（6万个数据图片训练时间大概在8个小时），硬件条件允许的话，卷积神经网络是一个不错的选择。

代码如下：

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

import tensorflow as tf

sess = tf.InteractiveSession()

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME')

W\_conv1 = weight\_variable([5, 5, 1, 32])

b\_conv1 = bias\_variable([32])

x\_image = tf.reshape(x, [-1,28,28,1])

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

W\_conv2 = weight\_variable([5, 5, 32, 64])

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])

b\_fc1 = bias\_variable([1024])

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

W\_fc2 = weight\_variable([1024, 10])

b\_fc2 = bias\_variable([10])

y\_conv=tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y\_conv))

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

sess.run(tf.initialize\_all\_variables())

for i in range(20000):

batch = mnist.train.next\_batch(50)

if i%100 == 0:

train\_accuracy = accuracy.eval(feed\_dict={x:batch[0], y\_: batch[1], keep\_prob: 1.0})

print("step %d, training accuracy %g"%(i, train\_accuracy))

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

print("test accuracy %g"%accuracy.eval(feed\_dict={x: mnist.test.images, y\_: mnist.test.labels, keep\_prob: 1.0}))

===END===