简介

任务时间：30min ~ 60min

本实验介绍深层神经网络在 TensorFlow 上的实现，并使用模型处理 MNIST 数据集。

MNIST 数据集介绍

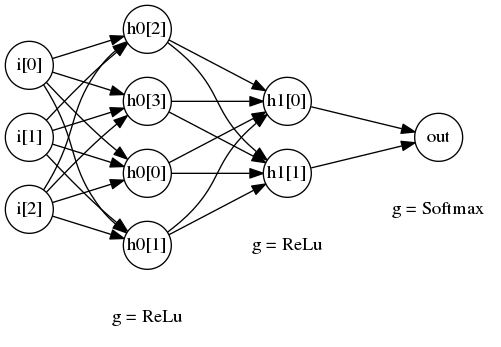
[MNIST](http://yann.lecun.com/exdb/mnist/) 是一个手写阿拉伯数字的数据集。

其中包含有 60000 个已经标注了的训练集，还有 10000 个用于测试的测试集。

本次实验的任务就是通过手写数字的图片，识别出具体写的是 0-9 之中的哪个数字。

理论知识回顾

一个两层的深层神经网络结构如下：



1. 上图所示的是一个具有两层隐藏层的深层神经网络
2. 第一个隐藏层有 4 个节点，对应的激活函数为 ReLu 函数
3. 第一个隐藏层有 2 个节点，对应的激活函数也是 Relu 函数
4. 最后，输出层使用 [softmax](http://ufldl.stanford.edu/wiki/index.php/Softmax%E5%9B%9E%E5%BD%92) 函数作为激活函数

**激活函数定义（activation function）：**

<formula>

softmax(x^i) = \left[

\begin{matrix}

p(y^i=1|x^i,\theta) \\

p(y^i=2|x^i,\theta) \\

\cdots \\

p(y^i=n|x^i,\theta)

\end{matrix}

\right] = \frac{1}{\sum\_{j=1}^{k}e^{\theta\_j^Tx^i}} \left[

\begin{matrix}

e^{\theta\_1^Tx^i} \\

e^{\theta\_2^Tx^i} \\

\cdots \\

e^{\theta\_k^Tx^i}

\end{matrix}

\right]

</formula>

<formula>

ReLu(x) = max(0, x)

</formula>

模型设计

与[上一个教程](https://cloud.tencent.com/developer/labs/lab/10298)类似:

1. MNIST 数据一共有 784 个输入，所以我们需要一个有 784 个节点的输入层。
2. MNIST 数据使用 One-Hot 格式输出，有 0-9 10 个 label，分别对应是否为数字 0-9，所以我们在输出层有 10 个节点，由于 0-9 的概率是**互斥**的，我们使用 [Softmax](http://ufldl.stanford.edu/wiki/index.php/Softmax%E5%9B%9E%E5%BD%92) 函数作为该层的激活函数。

**不一样**的是我们可以通过调整深度神经网络的层次来看看能不能达到不一样的效果。

训练模型

任务时间：30min ~ 60min

数据准备

首先我们需要先下载 MNIST 的数据集。使用以下的命令进行下载：

wget https://devlab-1251520893.cos.ap-guangzhou.myqcloud.com/t10k-images-idx3-ubyte.gz

wget https://devlab-1251520893.cos.ap-guangzhou.myqcloud.com/t10k-labels-idx1-ubyte.gz

wget https://devlab-1251520893.cos.ap-guangzhou.myqcloud.com/train-images-idx3-ubyte.gz

wget https://devlab-1251520893.cos.ap-guangzhou.myqcloud.com/train-labels-idx1-ubyte.gz

创建代码

现在您可以在 /home/ubuntu 目录下创建源文件 deep\_neural\_networks.py，内容可参考：

**示例代码：/home/ubuntu/deep\_neural\_networks.py**

#-\*- encoding:utf-8 -\*-

#!/usr/local/env python

import numpy as np

import tensorflow as tf

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

def add\_layer(inputs, in\_size, out\_size, activation\_function=None):

W = tf.Variable(tf.random\_normal([in\_size, out\_size]))

b = tf.Variable(tf.zeros([1, out\_size]) + 0.01)

Z = tf.matmul(inputs, W) + b

if activation\_function is None:

outputs = Z

else:

outputs = activation\_function(Z)

return outputs

if \_\_name\_\_ == "\_\_main\_\_":

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

learning\_rate = 0.01

batch\_size = 128

n\_epochs = 70

X = tf.placeholder(tf.float32, [batch\_size, 784])

Y = tf.placeholder(tf.float32, [batch\_size, 10])

layer\_dims = [784, 500, 500, 10]

layer\_count = len(layer\_dims)-1 # 不算输入层

layer\_iter = X

for l in range(1, layer\_count): # layer [1,layer\_count-1] is hidden layer

layer\_iter = add\_layer(layer\_iter, layer\_dims[l-1], layer\_dims[l], activation\_function=tf.nn.relu)

prediction = add\_layer(layer\_iter, layer\_dims[layer\_count-1], layer\_dims[layer\_count], activation\_function=None)

entropy = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=Y, logits=prediction)

loss = tf.reduce\_mean(entropy)

optimizer = tf.train.GradientDescentOptimizer(learning\_rate).minimize(loss)

init = tf.initialize\_all\_variables()

with tf.Session() as sess:

sess.run(init)

n\_batches = int(MNIST.test.num\_examples/batch\_size)

for i in range(n\_epochs):

for j in range(n\_batches):

X\_batch, Y\_batch = MNIST.train.next\_batch(batch\_size)

\_, loss\_ = sess.run([optimizer, loss], feed\_dict={X: X\_batch, Y: Y\_batch})

if i % 10 == 5 and j == 0:

print "Loss of epochs[{0}]: {1}".format(i, loss\_)

# test the model

n\_batches = int(MNIST.test.num\_examples/batch\_size)

total\_correct\_preds = 0

for i in range(n\_batches):

X\_batch, Y\_batch = MNIST.test.next\_batch(batch\_size)

preds = sess.run(prediction, feed\_dict={X: X\_batch, Y: Y\_batch})

correct\_preds = tf.equal(tf.argmax(preds, 1), tf.argmax(Y\_batch, 1))

accuracy = tf.reduce\_sum(tf.cast(correct\_preds, tf.float32))

total\_correct\_preds += sess.run(accuracy)

print "Accuracy {0}".format(total\_correct\_preds/MNIST.test.num\_examples)

代码讲解

**add\_layer 函数**

允许用户指定上一层的输出节点的个数作为 input\_size，本层的节点个数作为 output\_size，并指定激活函数 activation\_function 可以看到我们调用的时候位神经网络添加了两个隐藏层和输出层

for l in range(1, layer\_count): # layer [1,layer\_count-1] is hidden layer

layer\_iter = add\_layer(layer\_iter, layer\_dims[l-1], layer\_dims[l], activation\_function=tf.nn.relu)

prediction = add\_layer(layer\_iter, layer\_dims[layer\_count-1], layer\_dims[layer\_count], activation\_function=None)

entropy = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=Y, logits=prediction)

loss = tf.reduce\_mean(entropy)

**神经网络配置**

注意这一行，我们配置了一个深度的神经网络，它包含两个隐藏层，一个输入层和一个输出层 隐藏层的节点数为 500

layer\_dims = [784, 500, 500, 10]

执行代码

python deep\_neural\_networks.py

运行过程中，如果出现网络错误，请重试。

运行输出：

Loss of epochs[5]: 19.5915279388

Loss of epochs[15]: 7.20698690414

...

Loss of epochs[65]: 0.24952724576

Accuracy 0.939

可以看到经过 70 轮的训练，准确度大约在 94% 左右

探索（play around）

你可以通过修改下面的这个代码片段来修改整个神经网络。

**示例代码：/home/ubuntu/deep\_neural\_networks.py**

layer\_dims = [784, 500, 500, 10]

比如你可以去掉一层隐藏层，并将隐藏层的节点数改为 600

**示例代码：/home/ubuntu/deep\_neural\_networks.py**

layer\_dims = [784, 600, 10]

训练完的结果大概是这样：

Loss of epochs[5]: 13.1280488968

Loss of epochs[15]: 9.14239501953

Loss of epochs[25]: 3.60039186478

...

Loss of epochs[65]: 3.32054066658

Accuracy 0.9161

准确度大概 92% 的样子。have fun!

完成实验

任务时间：1min

实验内容已完成