**实验五 基于RNN模型和MNIST数据集的手写数字识别(Tensorflow)**

# 5.1实验目的

1. 熟悉Linux基本命令；
2. 熟悉深度学习一体机的操作环境；
3. 了解tensorflow框架的基本结构；
4. 了解MNIST数据集相关信息；
5. 了解最简单的循环神经网络——LSTM的网络结构；

# 5.2实验要求

本次试验后，要求学生能：

1. 熟悉服务器的环境，以及Linux下的基本操作；
2. 熟悉Tensorflow的基本使用；
3. 了解MNIST数据集的组成结构；
4. 简单了解LSTM网络的结构，及各层的功能；
5. 熟悉各种用到的工具脚本；
6. 训练出可以识别手写数字图片的模型。

# 5.3实验原理

5.3.1 MNIST数据集

MNIST是一个经典的机器学习数据集，数据集包含0到9,十个数字。共70000张图片。其中，训练样本有60000个、测试样本有10000个。TensorFlow中有用于下载MNIST数据集的python代码，可调用其中模块，下载MNIST数据集。源码为TensorFlow根目录下的examples/tutorials/mnist/input\_data.py，调用方式如下。

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

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

底层源码会执行下载、解压、重构图片和标签数据来组成data\_sets.train、data\_sets.validation、data\_sets.test。其中data\_sets.train有55000组图片和标签用于训练，data\_sets.validation有5000组图片和标签用于迭代验证训练的准确性、data\_sets.test有10000组图片和标签，用于测试训练的准确性。每一个数据单元由两部分构成——图片和对应的标签。如训练数据的图片和标签分别为mnist.train.images和 mnist.train.labels。

执行read\_data\_sets()函数将会返回一个Dataset实例，实例包含了以上3个数据集。函数DataSet.next\_batch()用于获取batch\_size大小的一个元祖，其中包含了一组图片和标签，该元祖会用于当前TensorFlow运算会话中。

images\_feed, labels\_feed = data\_set.next\_batch(FLAGS.batch\_size)

每张图片的大小均为 IMG_256像素，均可用一个数字数组表示，如图1-1所示。

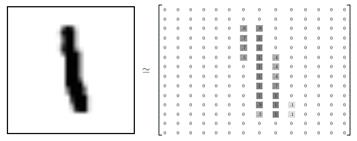


图5-1  MNIST手写数字的数组表示

将此数组展开成为一个向量，长度为IMG_258，由此，则MNIST数字训练集中的每个图片都可以看做一个784维的向量。

在MNIST训练数据集中，mnist.train.images是一个形状为[60000,784]大小的张量，第一维数字表示60000个训练数据集的第几个数据，第二维数字表示某一个训练图片（拉伸为784维的向量）中的第几个像素。张量的每一个元素值介于0-1之间，表示了像素值的强弱。而mnist.train.labels为一个形状为[60000,10]的张量，第二维对应每张图片的标签，标签为“one-hot vector”即只有一位为1，其余均为0。例如，2的标签为[0,0,1,0,0,0,0,0,0,0,]。

5.3.2 LSTM网络

RNN背后的想法是利用顺序信息。在传统的神经网络中，我们假设所有输入（和输出）彼此独立；但对于许多任务而言这是一个非常糟糕的主意，如果你想预测句子中的下一个单词，你最好知道它前面有哪些单词。RNN被称为循环，因为它们对序列的每个元素执行相同的任务，输出取决于先前的计算。考虑RNN的另一种方式是它们有一个“记忆”，它可以捕获到目前为止计算的信息。理论上RNN可以利用任意长序列中的信息，但实际上它们仅限于回顾几个步骤。

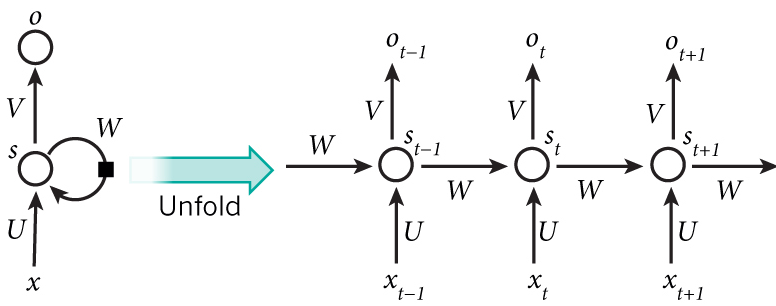


图5-2 典型RNN结构展开图

上图显示了RNN展开到完整网络中。例如我们关心的序列是5个单词的句子，则网络将展开为5层神经网络，每个单词一个层。管理RNN中发生的计算的公式如下：

a.是在t时刻的输入，例如可能是对应于一个句子的第二个单词的one-hot向量

b.是t时刻中的隐藏状态，他是网络的“记忆”。是基于先前的隐藏状态和当前时刻的输出计算得到：。函数f通常是非线性的如tanh或ReLU函数，计算第一隐藏状态所需的通常被初始化为全0。

c.是t时刻的输出，例如我们想要预测句子中的下一个单词，那么他就是词汇表中概率的向量，。

最常用的RNN模型是LSTM，它在捕获长期依赖性特点上表现较好。LSTM与RNN的架构没有太大区别，但它们使用不同的函数来计算隐藏状态。LSTM中的内存称为单元格，将以前的状态和当前输入作为输入。在内部这些单元格决定要保留什么（以及从中擦除什么）内存。然后，它们组合先前的状态、当前内存和输入。事实证明，这些类型的单元在捕获长期依赖性方面非常有效。

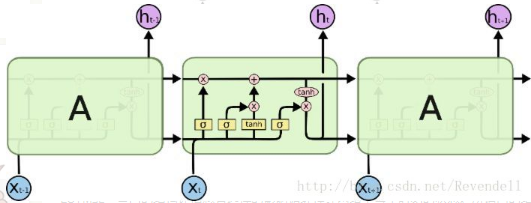


图5-3 LSTM网络结构图

LSTM靠一些门的结构让信息有选择的影响循环神经网络中每个时刻的状态。所谓门的结构就是一个使用sigmoid神经网络和按位做乘法的操作，这两个操作合在一起就是一个门的结构。当门打开时（sigmoid神经网络层输出为1时），全部信息都可以通过;当门关上时（sigmoid神经网络层输出为0时），任何信息都无法通过。

# 5.4实验步骤

本实验选用tensorflow1.4+python2.7镜像。

5.4.1 安装依赖

1、  修改apt-get源

vim /etc/apt/sources.list

添加

deb http://archive.ubuntu.com/ubuntu/ raring main restricted universe multiverse

deb http://archive.ubuntu.com/ubuntu/ raring-security main restricted universe multiverse

deb http://archive.ubuntu.com/ubuntu/ raring-updates main restricted universe multiverse

deb http://archive.ubuntu.com/ubuntu/ raring-proposed main restricted universe multiverse

deb http://archive.ubuntu.com/ubuntu/ raring-backports main restricted universe multiverse

deb-src http://archive.ubuntu.com/ubuntu/ raring main restricted universe multiverse

deb-src http://archive.ubuntu.com/ubuntu/ raring-security main restricted universe multiverse

deb-src http://archive.ubuntu.com/ubuntu/ raring-updates main restricted universe multiverse

deb-src http://archive.ubuntu.com/ubuntu/ raring-proposed main restricted universe multiverse

deb-src http://archive.ubuntu.com/ubuntu/ raring-backports main restricted universe multiverse

执行

apt-get update

安装OpenSSL

# apt-get install libssl-dev

安装zlib

apt-get install zlib1g-dev

重新编译python，支持其中的ssl和zlib模块。

# cd ~/cDeep/Python-2.7.13

# vim Modules/Setup.dist

仅将文件中涉及SSL的四行代码以及涉及zlib的一行代码前的注释符号去掉，使这两部分内容生效。生效的5行代码如下：

SSL部分：

SSL=/usr/local/ssl

\_ssl \_ssl.c \

        -DUSE\_SSL -I$(SSL)/include -I$(SSL)/include/openssl \

        -L$(SSL)/lib -lssl –lcrypto

Zlib部分：

zlib zlibmodule.c -I$(prefix)/include -L$(exec\_prefix)/lib -lz

重新编译python

# ./configure --enable-unicode=ucs4

# make

# make install

1. vim文件操作设置

编辑/etc/vim/vimrc文件，设置tab键为4个空格，设置显示行号以及中文编码。

#vim /etc/vim/vimrc

在文件最后添加如下内容：

set ts=4#设置tab=4

set nu #永久显示行号

set fileencodings=utf-8,gbk,utf-16le,cp1252,iso-8859-15,ucs-bom

set termencoding=utf-8

set encoding=utf-8

5.4.2 模型训练

mkdir /root/mnistRnn

vim settings.py

代码如下，其中数据集下载调用了tensorflow.examples.tutorials.mnist import input\_data部分。

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

mnist = input\_data.read\_data\_sets('mnist/', one\_hot=True)# minst测试集

batch\_size = 100# 每次使用100条数据进行训练

# 图像向量

width = 28

height = 28

rnn\_size = 256# LSTM隐藏神经元数量

out\_size = 10# 输出层one-hot向量长度的

第一次运行代码，速度较慢，因为程序会先下载数据集。

# python settings.py

若下载失败或下载较慢，也可直接从实验平台中获取数据集。置于mnist文件夹下。编辑rnn\_train.py文件。

vim rnn\_train.py

代码如下：

import os

import tensorflow as tf

from datetime import datetime

from settings import mnist, batch\_size, width, height, rnn\_size, out\_size

def weight\_variable(shape, w\_alpha=0.01):

'''

增加噪音，随机生成权重

:param shape:

:param w\_alpha:

:return:

'''

initial = w\_alpha \* tf.random\_normal(shape)

return tf.Variable(initial)

def bias\_variable(shape, b\_alpha=0.1):

'''

增加噪音，随机生成偏置项

:param shape:

:param b\_alpha:

:return:

'''

initial = b\_alpha \* tf.random\_normal(shape)

return tf.Variable(initial)

def rnn\_graph(x, rnn\_size, out\_size, width, height):

'''

循环神经网络计算图

:param x:

:param rnn\_size:

:param out\_size:

:param width:

:param height:

:return:

'''

# 权重及偏置

w = weight\_variable([rnn\_size, out\_size])

b = bias\_variable([out\_size])

# LSTM

lstm\_cell = tf.nn.rnn\_cell.BasicLSTMCell(rnn\_size)

# 原排列[0,1,2]transpose为[1,0,2]代表前两维装置，如shape=(1,2,3)转为shape=(2,1,3)

# 这里的实际意义是把所有图像向量的相同行号向量转到一起，如x1的第一行与x2的第一行

x = tf.transpose(x, [1, 0, 2])

# reshape -1 代表自适应，这里按照图像每一列的长度为reshape后的列长度

x = tf.reshape(x, [-1, width])

# split默任在第一维即0 dimension进行分割，分割成height份，这里实际指把所有图片向量按对应行号进行重组

x = tf.split(x, height)

# 这里RNN会有与输入层相同数量的输出层，我们只需要最后一个输出

outputs, status = tf.nn.static\_rnn(lstm\_cell, x, dtype=tf.float32)

y\_conv = tf.add(tf.matmul(outputs[-1], w), b)

return y\_conv

def optimize\_graph(y, y\_conv):

'''

优化计算图

:param y:

:param y\_conv:

:return:

'''

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=y\_conv, labels=y))

optimizer = tf.train.AdamOptimizer().minimize(loss)

return optimizer

def accuracy\_graph(y, y\_conv):

'''

偏差计算图

:param y:

:param y\_conv:

:return:

'''

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

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

return accuracy

def train():

'''

rnn训练

:return:

'''

# 按照图片大小申请占位符

x = tf.placeholder(tf.float32, [None, height, width])

y = tf.placeholder(tf.float32)

# rnn模型

y\_conv = rnn\_graph(x, rnn\_size, out\_size, width, height)

# 最优化

optimizer = optimize\_graph(y, y\_conv)

# 偏差

accuracy = accuracy\_graph(y, y\_conv)

# 启动会话.开始训练

saver = tf.train.Saver()

session = tf.Session()

session.run(tf.global\_variables\_initializer())

step = 0

acc\_rate = 0.98

while 1:

batch\_x, batch\_y = mnist.train.next\_batch(batch\_size)

batch\_x = batch\_x.reshape([batch\_size, height, width])

session.run(optimizer, feed\_dict={x: batch\_x, y: batch\_y})

# 每训练10次测试一次

if step % 10 == 0:

batch\_x\_test = mnist.test.images

batch\_y\_test = mnist.test.labels

batch\_x\_test = batch\_x\_test.reshape([-1, height, width])

acc = session.run(accuracy, feed\_dict={x: batch\_x\_test, y: batch\_y\_test})

print(datetime.now().strftime('%c'), ' step:', step, ' accuracy:', acc)

# 偏差满足要求，保存模型

if acc >= acc\_rate:

model\_path = os.getcwd() + os.sep + str(acc\_rate) + "mnist.model"

saver.save(session, model\_path, global\_step=step)

break

step += 1

session.close()

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

train()

训练模型

python rnn\_train.py

5.4.3 模型测试

编辑mnist\_rnn.py文件。

vim mnist\_rnn.py

代码如下：

import tensorflow as tf

import numpy as np

from rnn\_train import rnn\_graph

from settings import mnist, width, height, rnn\_size, out\_size

def mnist2text(image\_list, height, width, rnn\_size, out\_size):

x = tf.placeholder(tf.float32, [None, height, width])

y\_conv = rnn\_graph(x, rnn\_size, out\_size, width, height)

saver = tf.train.Saver()

with tf.Session() as sess:

saver.restore(sess, tf.train.latest\_checkpoint('.'))

predict = tf.argmax(y\_conv, 1)

vector\_list = sess.run(predict, feed\_dict={x: image\_list})

vector\_list = vector\_list.tolist()

return vector\_list

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

batch\_x\_test = mnist.test.images

batch\_x\_test = batch\_x\_test.reshape([-1, height, width])

batch\_y\_test = mnist.test.labels

batch\_y\_test = list(np.argmax(batch\_y\_test, 1))

pre\_y = list(mnist2text(batch\_x\_test, height, width, rnn\_size, out\_size))

for text in batch\_y\_test:

print('Label:', text, ' Predict:', pre\_y[batch\_y\_test.index(text)])

运行代码：

# python mnist\_rnn.py

# 5.5实验结果

5.5.1 模型训练

第一次运行程序较慢，运行后，当前目录下会生成存放数据集的文件夹mnist。运行结果如下，准确率为98%，部分输出如下。

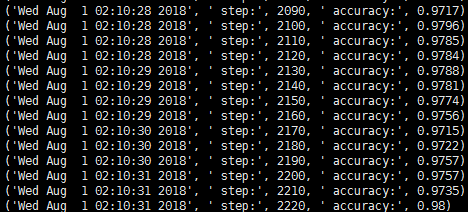


图5-4 训练运行结果图

5.5.2 模型测试

部分输出如下。

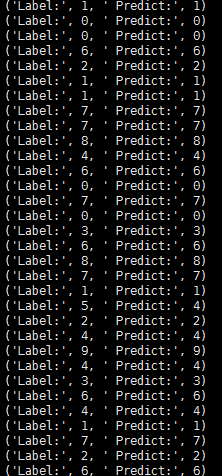


图5-5 模型测试结果图