# LSTM的使用——以手写数字识别为例

## LSTM网络简介

关于LSTM内部机制在这里不详细说明，请参考有关资料，这里主要说明LSTM网络。

如图所示LSTM网络。LSTM网络由LSTM单元构成，LSTM网络接受的是序列，序列长度为TIME\_STEP，即有TIME\_STEP个LSTM单元构成LSTM网络，每个输入向量大小为vec\_size，TIME\_STEP个向量组成一个样本。例如在使用LSTM网络预测sin(x)时，vec\_size为1，在下面的例子中，将图像每行（28个像素）作为一个向量输入到一个LSTM单元中，vec\_size为28，每个图像共28行，每行对应一个LSTM单元，TIME\_STEP = 28。Batch\_size个样本可以作为一次全部输入到网络中。每个LSTM单元都有一个输出，一般LSTM单元的输出后再经过一个全连接层，最后一个单元的输出为网络的最终输出。

1

2

28

TIME\_STEP = 28

OUTPUT[-1]

batch\_size

vec\_size

图 1 LSTM网络

每个LSTM单元内部包含若干个神经元，在图中没有画出，这在使用BasicLSTMCell来定义LSTM单元的时候需要指定。

## 基于RNN的手写数字识别实现

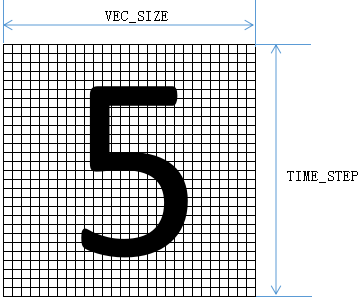


图 2 图像的分解

如图 2所示，将一张28\*28的灰度图像每行作为一个向量，共28行，将其视为一个28个向量的序列，这样一张图像可以看作一个序列样本。

### 2.1网络定义

LSTM单元的定义，使用BasicLSTMCell定义单元：

lstm\_cell = rnn.BasicLSTMCell(n\_hidden, forget\_bias=1.0)

使用static\_rnn定义LSTM网络：

outputs, states = rnn.static\_rnn(lstm\_cell, x, dtype = tf.float32)

返回outputs为网络输出，output[-1]为网络的最终输出，states为网络的最终状态，是一个n\_hidden长度的向量。

将outputs[-1]后接一个全连接层，得到最后的输出。

这里要注意的是，static\_rnn接受的输入的形状为长度为n\_step的列表，列表的元素的形状为[batch\_size, n\_input]的array，而我们开始的输入数据是[batch\_size, n\_steps, n\_input]，因此需要将其进行转化。

|  |
| --- |
| **def** model(x, weights, biases):  *#x的shape为[batch\_size, n\_steps, n\_input]* x = tf.transpose(x, [1, 0, 2])  *#x的shape为[batch\_size \* n\_step, n\_input]* x = tf.reshape(x, [-1, n\_input])x = tf.split(axis=0, num\_or\_size\_splits=n\_steps, value=x)lstm\_cell = rnn.BasicLSTMCell(n\_hidden, forget\_bias=1.0)outputs, states = rnn.static\_rnn(lstm\_cell, x, dtype = tf.float32)**return** tf.matmul(outputs[-1], weights) + biases |

### 2.2 session的定义

在定义session时，使用了如下的定义方法，不知为何如不这样定义会报错。

|  |
| --- |
| session\_conf = tf.ConfigProto() session\_conf.gpu\_options.allow\_growth = **True with** tf.Session(config=session\_conf) **as** sess: |

## 代码

|  |
| --- |
| *# -\*- coding: utf-8 -\*-* **import** tensorflow **as** tf **from** tensorflow.contrib **import** rnn **from** tensorflow.examples.tutorials.mnist **import** input\_data  batch\_size = 128 test\_size = 256 n\_input = 28 n\_steps = 28 n\_hidden = 128 n\_classes = 10  **def** init\_weights(shape):  **return** tf.Variable(tf.random\_normal(shape, stddev=0.01))  **def** model(x, weights, biases):  *#x的shape为[batch\_size, n\_steps, n\_input]* x = tf.transpose(x, [1, 0, 2])  *#x的shape为[batch\_size \* n\_step, n\_input]* x = tf.reshape(x, [-1, n\_input])  *#x为一个列表，列表有n\_steps个元素，每个元素为一个数组，每个数组的长度为batch\_size* x = tf.split(axis=0, num\_or\_size\_splits=n\_steps, value=x)  *#创建一个lstm单元* lstm\_cell = rnn.BasicLSTMCell(n\_hidden, forget\_bias=1.0)  *#基于lstm单元创建循环神经网络，outputs为长度为n\_steps的输出列表， states为最终状态* outputs, states = rnn.static\_rnn(lstm\_cell, x, dtype = tf.float32)  *#output[-1]为最拉的输出* **return** tf.matmul(outputs[-1], weights) + biases  mnist = input\_data.read\_data\_sets(**"E:/mnist/"**, one\_hot=**True**)X = tf.placeholder(**"float"**, [**None**, 28, 28]) Y = tf.placeholder(**"float"**, [**None**, 10]) W = init\_weights([n\_hidden, 10]) *# 输出层权重矩阵28×10* B = init\_weights([10]) *# 输出层bais* py\_x = model(X, W, B)  cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=py\_x, labels=Y)) train\_op = tf.train.AdamOptimizer(0.001).minimize(cost) correct\_pred = tf.equal(tf.argmax(py\_x, 1), tf.argmax(Y, 1)) accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))  session\_conf = tf.ConfigProto() session\_conf.gpu\_options.allow\_growth = **True  with** tf.Session(config=session\_conf) **as** sess:  tf.global\_variables\_initializer().run()  step = 1  **while** step \* batch\_size < 100000:  batch\_x, batch\_y = mnist.train.next\_batch(batch\_size)  batch\_x = batch\_x.reshape((batch\_size, n\_steps, n\_input))  sess.run(train\_op, feed\_dict={X: batch\_x, Y: batch\_y})  **if** step % 100 == 0:  acc = sess.run(accuracy, feed\_dict={X: batch\_x, Y: batch\_y})  loss = sess.run(cost, feed\_dict={X: batch\_x, Y: batch\_y})  print(**"Iter "** + str( step \* batch\_size) + **", Minbatch loss = "** + **"{:.6}"**.format(loss) +  **",Trainning Accuracy = "** + **"{:.5f}"**.format(acc))  step +=1  print(**"Optimization Finished!"**)  test\_len = 128  test\_data = mnist.test.images[:test\_len].reshape((-1, n\_steps, n\_input))  test\_label = mnist.test.labels[:test\_len]  print(**"Test Accuracy:"**, sess.run(accuracy, feed\_dict={X: test\_data, Y: test\_label})) |