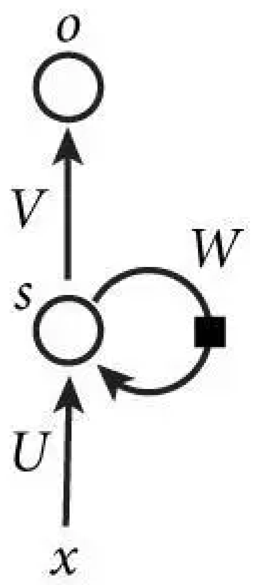
RNN入门（一）识别MNIST数据集

原创 jclian NLP奇幻之旅 2018-09-25

RNN介绍

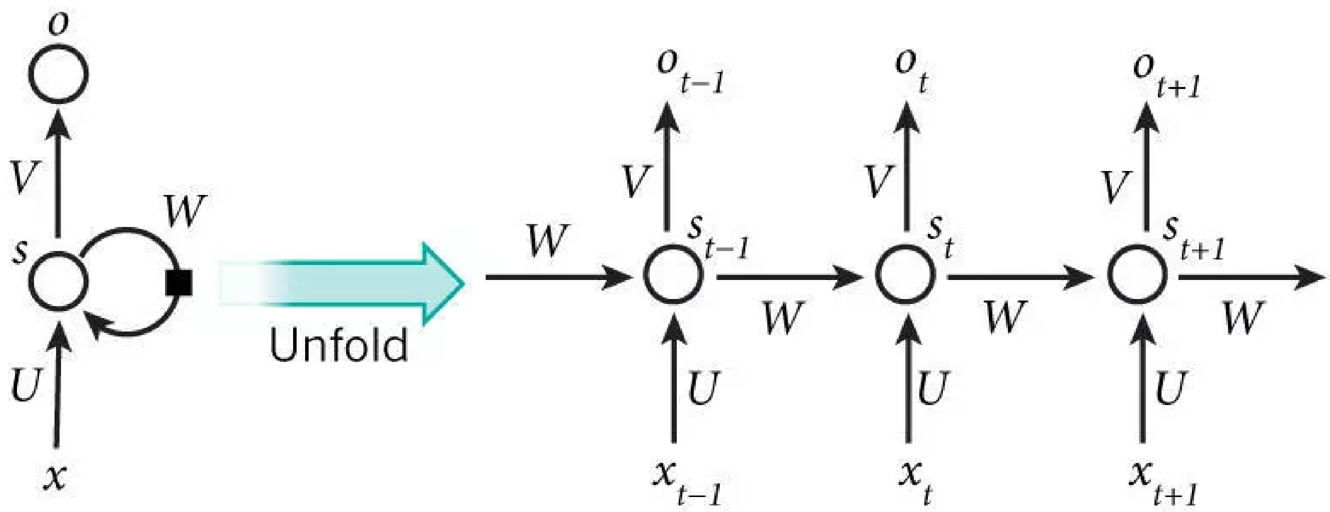
在读本文之前，读者应该对**全连接神经网络**（Fully Connected Neural Network，FCNN）和**卷积神经网络**（Convolutional Neural Network，CNN）有一定的了解。对于FCNN和CNN来说，他们能解决很多实际问题，但是它们都只能单独的取处理一个个的输入，前一个输入和后一个输入是完全没有关系的 。而在现实生活中，我们输入的向量往往存在着前后联系，即前一个输入和后一个输入是有关联的，比如文本，语音，视频等，因此，我们需要了解深度学习中的另一类重要的神经网络，那就是*循环神经网络(Recurrent Neural Network，RNN).*

*循环神经网络(Recurrent Neural Network，RNN)*依赖于一个重要的概念：**序列（Sequence）**，即输入的向量是一个序列，存在着前后联系。简单RNN的结构示意图如下：



简单RNN的结构示意图

相比于之前的FCNN，RNN的结构中多出了一个自循环部分，即W所在的圆圈，这是RNN的精华所在，它展开后的结构如下：



RNN展开后的结构

对于t时刻的输出向量 o_t ，它的输出不仅仅依赖于t时刻的输入向量 x_t ，还依赖于t-1时刻的隐藏层向 s_{t-1} ，以下是输出向量 o_t 的计算公式：

$$s_t = f(Ux_t + Ws_{t-1})$$

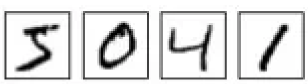
$$o_t = g(Vs_t)$$

其中，第二个式子为输出层的计算公式，输出层为全连接层，V为权重矩阵，g为激活函数。第一个式子中，U是输入x的权重矩阵，W是上一次隐藏层值s的输入权重矩阵，f为激活函数。注意到，RNN的所有权重矩阵U,V,W是共享的，这样可以减少计算量。

本文将会用TensorFlow中已经帮我们实现好的RNN基本函数tf.contrib.rnn.BasicRNNCell()，tf.nn.dynamic_rnn()来实现简单RNN，并且用该RNN来识别MNIST数据集。

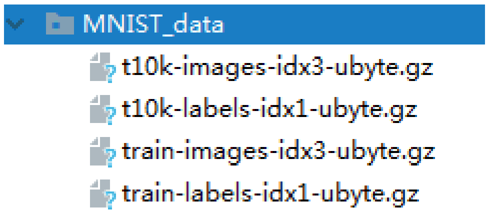
MNIST数据集

MNIST数据集是深度学习的经典入门demo，它是由6万张训练图片和1万张测试图片构成的，每张图片都是**28*28**大小（如下图），而且都是黑白色构成（这里的黑色是一个0-1的浮点数，黑色越深表示数值越靠近1），这些图片是采集的不同的人手写从0到9的数字。



MNIST中的手写数字

在TensorFlow中，已经内嵌了MNIST数据集，笔者已经下载下来了，如下：



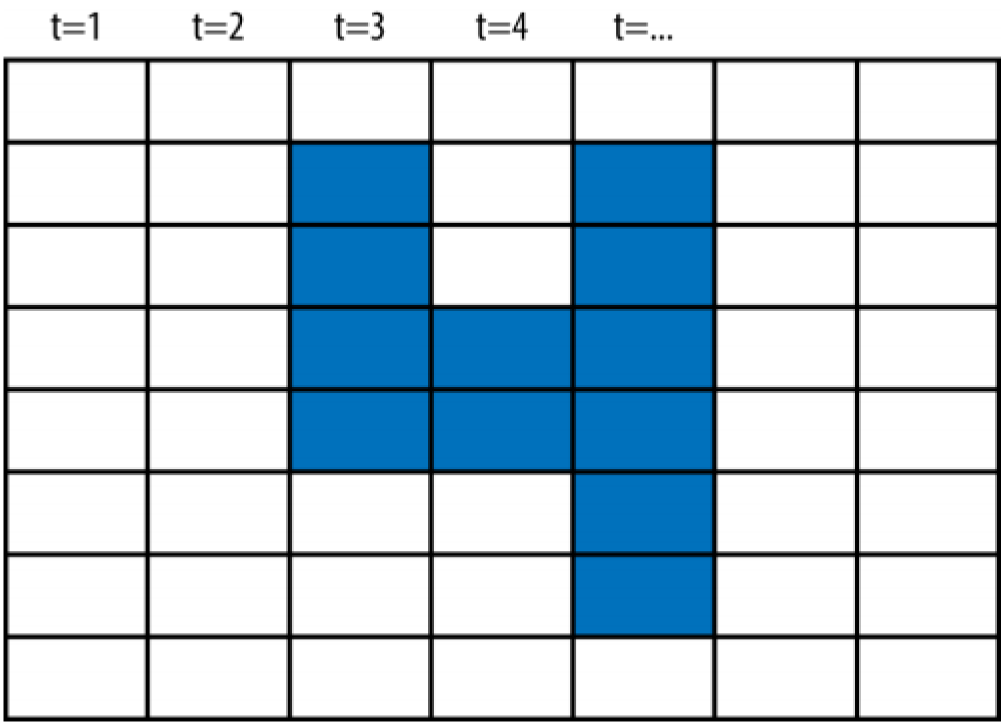
TensorFlow中的MNIST数据集

接下来本文将要用MNIST数据集作为RNN应用的一个demo.

RNN大战MNIST数据集

用CNN来识别MNIST数据集，我们好理解，这是利用了图片的空间信息。可是，RNN要求输入的向量是序列，那么，如何把图片看成是序列呢？

图片的大小为28*28,我们把每一列向量看成是某一时刻的向量，那么每张图片就是一个序列，里面含有28个向量，每个向量含有28个元素，如下：



将图片看成序列

下面给出如何利用TensorFlow来搭建简单RNN，用来识别MNIST数据集，完整的Python代码如下：

```
# -*- coding: utf-8 -*-
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

# 获取MNIST数据
mnist = input_data.read_data_sets(r"./MNIST_data", one_hot=True)

# 设置RNN结构
element_size = 28
time_steps = 28
num_classes = 10
batch_size = 128
hidden_layer_size = 150

# 输入向量和输出向量
_inputs = tf.placeholder(tf.float32, shape=[None, time_steps, element_size], name='input
y = tf.placeholder(tf.float32, shape=[None, num_classes], name='inputs')

# 利用TensorFlow的内置函数BasicRNNCell, dynamic_rnn来构建RNN的基本模块
rnn_cell = tf.contrib.rnn.BasicRNNCell(hidden_layer_size)
outputs, _ = tf.nn.dynamic_rnn(rnn_cell, _inputs, dtype=tf.float32)
Wl = tf.Variable(tf.truncated_normal([hidden_layer_size, num_classes], mean=0,stddev=.01
bl = tf.Variable(tf.truncated_normal([num_classes],mean=0,stddev=.01))

def get_linear_layer(vector):
    return tf.matmul(vector, Wl) + bl

# 取输出的向量outputs中的最后一个向量最为最终输出
last_rnn_output = outputs[:,-1,:]
final_output = get_linear_layer(last_rnn_output)

# 定义损失函数并用RMSPropOptimizer优化
softmax = tf.nn.softmax_cross_entropy_with_logits(logits=final_output, labels=y)
cross_entropy = tf.reduce_mean(softmax)
train_step = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cross_entropy)

# 统计准确率
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(final_output,1))
accuracy = (tf.reduce_mean(tf.cast(correct_prediction, tf.float32)))*100

sess=tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
# 测试集
test_data = mnist.test.images[:batch_size].reshape((-1, time_steps, element_size))
test_label = mnist.test.labels[:batch_size]

# 每次训练batch_size张图片，一共训练3000次
for i in range(3001):
    batch_x, batch_y = mnist.train.next_batch(batch_size)
    batch_x = batch_x.reshape((batch_size, time_steps, element_size))
    sess.run(train_step, feed_dict={_inputs:batch_x, y:batch_y})
    if i % 100 == 0:
        loss = sess.run(cross_entropy, feed_dict={_inputs: batch_x, y: batch_y})
        acc = sess.run(accuracy, feed_dict={_inputs:batch_x, y: batch_y})
        print ("Iter " + str(i) + ", Minibatch Loss= " + \
               "{:.6f}".format(loss) + ", Training Accuracy= " + \
               "{:.5f}".format(acc))
```

```
# 在测试集上的准确率
print("Testing Accuracy:", sess.run(accuracy, feed_dict={_inputs:test_data, y:test_label
```

运行上述代码，输出的结果如下：

```
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
Iter 0, Minibatch Loss= 2.301171, Training Accuracy= 11.71875
Iter 100, Minibatch Loss= 1.718483, Training Accuracy= 47.65625
Iter 200, Minibatch Loss= 0.862968, Training Accuracy= 71.09375
Iter 300, Minibatch Loss= 0.513068, Training Accuracy= 86.71875
Iter 400, Minibatch Loss= 0.570475, Training Accuracy= 83.59375
Iter 500, Minibatch Loss= 0.254566, Training Accuracy= 92.96875
Iter 600, Minibatch Loss= 0.457989, Training Accuracy= 85.93750
Iter 700, Minibatch Loss= 0.151181, Training Accuracy= 96.87500
Iter 800, Minibatch Loss= 0.171168, Training Accuracy= 94.53125
Iter 900, Minibatch Loss= 0.142494, Training Accuracy= 94.53125
Iter 1000, Minibatch Loss= 0.155114, Training Accuracy= 97.65625
Iter 1100, Minibatch Loss= 0.096007, Training Accuracy= 96.87500
Iter 1200, Minibatch Loss= 0.341476, Training Accuracy= 88.28125
Iter 1300, Minibatch Loss= 0.133509, Training Accuracy= 96.87500
Iter 1400, Minibatch Loss= 0.076408, Training Accuracy= 98.43750
Iter 1500, Minibatch Loss= 0.122228, Training Accuracy= 98.43750
Iter 1600, Minibatch Loss= 0.099382, Training Accuracy= 96.87500
Iter 1700, Minibatch Loss= 0.084686, Training Accuracy= 97.65625
Iter 1800, Minibatch Loss= 0.067009, Training Accuracy= 98.43750
Iter 1900, Minibatch Loss= 0.189703, Training Accuracy= 94.53125
Iter 2000, Minibatch Loss= 0.116077, Training Accuracy= 96.09375
Iter 2100, Minibatch Loss= 0.028867, Training Accuracy= 100.00000
Iter 2200, Minibatch Loss= 0.064198, Training Accuracy= 99.21875
Iter 2300, Minibatch Loss= 0.078259, Training Accuracy= 97.65625
Iter 2400, Minibatch Loss= 0.106613, Training Accuracy= 97.65625
Iter 2500, Minibatch Loss= 0.078722, Training Accuracy= 98.43750
Iter 2600, Minibatch Loss= 0.045871, Training Accuracy= 98.43750
Iter 2700, Minibatch Loss= 0.030953, Training Accuracy= 99.21875
Iter 2800, Minibatch Loss= 0.062823, Training Accuracy= 96.87500
Iter 2900, Minibatch Loss= 0.040367, Training Accuracy= 99.21875
Iter 3000, Minibatch Loss= 0.017787, Training Accuracy= 100.00000
Testing Accuracy: 97.6563
```

可以看到，用简单RNN来识别MNIST数据集，也能取得很好的效果！

本次分享到此结束，欢迎大家交流~

注意：本人现已开通微信公众号：轻松学会Python爬虫（微信号为：easy_web_scrape），欢迎大家关注哦~~