TensorFlow人工智能引擎入门教程之九 RNN/LSTM循环神经网络长短期记忆网络使用

2016-05-15 蓝莓对冲... 阅 25 转 1 分享： 微信 转藏到我的图书馆

这几天空余时间玩了2天的单机游戏。 黑暗之魂，手柄玩起来挺爽， 这一章节我们讲一下 循环神经网络，RNN 是一种非常通用的神经网络，无论是图像识别 还是 声音识别 文字识别 NLP 时间系列的数据 周期的数据 等等都是通用适合的.

在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNNs之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNNs能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关.比如 图片 如果28x28像素 如果我们把每一行像素跟上一行的像素 看做时间系列处理的话，也就是 在传统的神经网络WX+B 引入了t 每一个时间系列 t范围内，下一次的w 通过上一个t-1 的wx+b得到 w 所以 图像如果使用RNN处理适合 图像理解 效果很好，NLP 效果也很好。

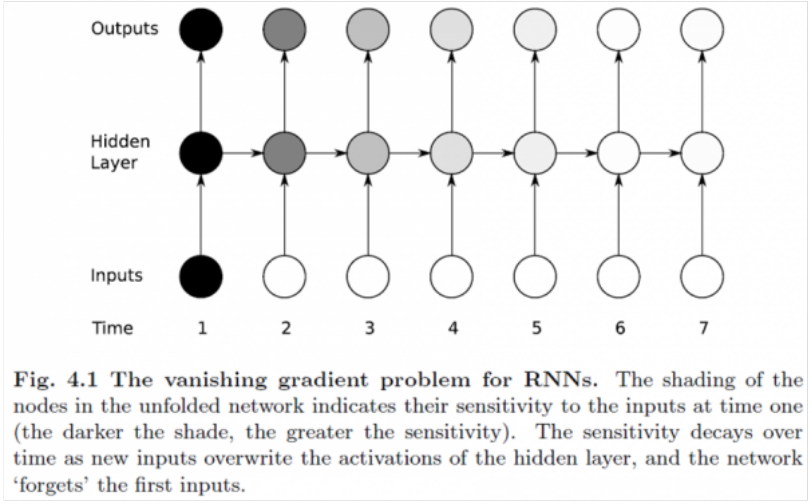
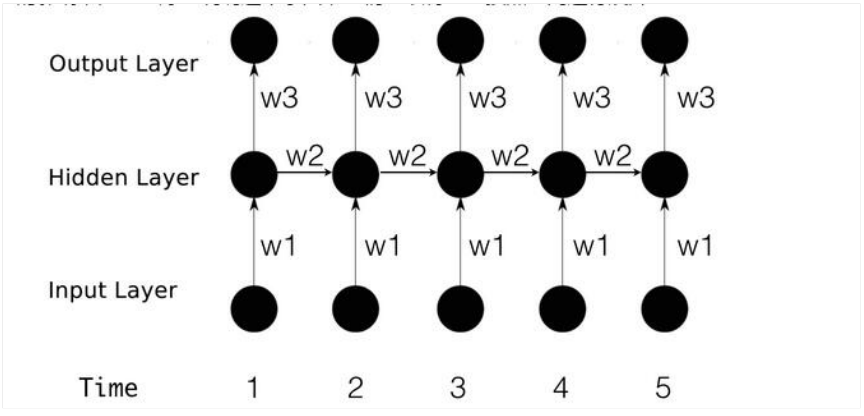


Fig. 4.1 The vanishing gradient problem for RNNs. The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.



详细的介绍 可以看这个http://www.360doc.com/content/16/0328/15/1317564_545892642.shtml

下面看看RNN输入， 首先seq/step 时间系列 ，如果28x28的图像 RNN处理的话，其实就是每一个seq/step=28长度 28的输入，所以 对于tensorflow 来说输入维度**[size_batches, seq_length, rnn_size]**

下面 我们看看 输入的X为【size_batches,input] 要转换成上面的格式 所以需要reshape重新维度

xdata.reshape((batch_size, n_steps, n_input)) 转换成 [batch,28,28] 其实28个t 上下文计算 最后的t得到输出的w b 用于全连接。也就说28步 step

所以下面我们来看看tensorflow定义 对于RNN 来说我们需要的是一个 step=28 的 的(batchsize,n_input) 一层一层上下文计算。对于每一个step得到相应的state 状态以及 output下一层连接层的输入，得到最后一个上下文来进入连接层的输入。

```
def rnn(cell, inputs, initial_state=None, dtype=None,
        sequence_length=None, scope=None):
  """Creates a recurrent neural network specified by RNNCell "cell".

  The simplest form of RNN network generated is:
    state = cell.zero_state(...)
    outputs = []
    states = []
    for input_ in inputs:
      output, state = cell(input_, state)
      outputs.append(output)
      states.append(state)
    return (outputs, states)

  However, a few other options are available:

  An initial state can be provided.
  If sequence_length is provided, dynamic calculation is performed.

  Dynamic calculation returns, at time t:
    (t >= max(sequence_length)
        ? (zeros(output_shape), zeros(state_shape))
        : cell(input, state)

  Thus saving computational time when unrolling past the max sequence length.

  Args:
    cell: An instance of RNNCell.
    inputs: A length T list of inputs, each a vector with shape [batch_size].
    initial_state: (optional) An initial state for the RNN.  This must be
      a tensor of appropriate type and shape [batch_size x cell.state_size].
    dtype: (optional) The data type for the initial state.  Required if
      initial_state is not provided.
    sequence_length: An int64 vector (tensor) size [batch_size].
    scope: VariableScope for the created subgraph; defaults to "RNN".

  Returns:
    A pair (outputs, states) where:
      outputs is a length T list of outputs (one for each input)
      states is a length T list of states (one state following each input)

  Raises:
    TypeError: If "cell" is not an instance of RNNCell.
```

下面介绍介个函数tensorflow 中 用于颠倒交换维度的

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

上面 对于3维的X 默认是 0 1 2 系数 ，我们把它置换成1 0 2 也就是 第一维 第二维进行交换

比如 (batch_size, n_steps, n_input) 交换后就是 （ nsteps,batch_size,n_input) 这正好就是我们上面需要的28步step的

（batch_size,n_input）的输入。但是这是三维的数组 所以需要降维重新展开reshape

tf.reshape(_X, [-1, n_input]) 这里我讲一下 reshape 就是重新展开 定义维度。-1 表示站位，比如 一个多维的数组，[2 ,2 ,3] 的维度的3维数组，reshape(【-1,3】) 表示要变成 后面维度为3 那么前面的-1 表示全部展开后，除掉 3 那么其实就是shape(4,3) 的维度 二维数组。

所以上面的我们只需要reshape([-1, n_input]) 那么 此时将会的到一个(n_steps*batch_size, n_input)

下面的就遵循矩阵运算 WX+B一步一步即可 最后每一个28step 的每一个step得到一个outpput 根据上下文 上文计算下文的w 所以 最后的输出 就是最后一个元素得到的w 将会最后进行全连接连接层wx+b。

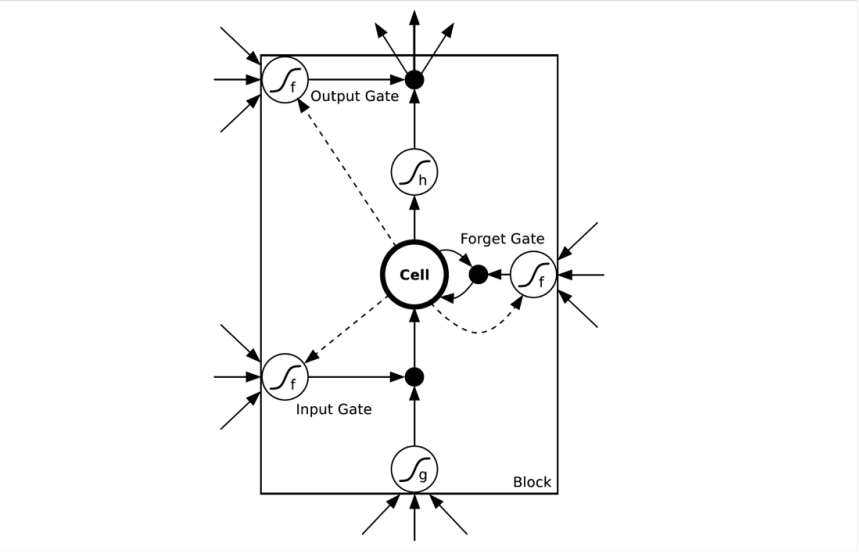
下面看看官方案例的一个例子 。 此时 把图像 从 28x28 当做一个 28 step的 28 input 的 RNN ，就好像记忆一样，我们现在所有的决定 都是过去的经验经历 记忆所影响的。后面有一个RNN上 的变体，LSTM 长短期记忆网络 解决了RNN的缺点 ，对RNN隐藏层进行改进

这里我们把RNN/LSTM 放在一起 ，是因为他本质是一样的 。

LSTM 就是把RNN的单元 换了更好的单元 就是加上了记忆。所以LSTM 长短期记忆网络

长期是通过遗忘门进行调节。 短期是通过记忆门进行调节。

LSTM引入了Cell 与其说LSTM是一种RNN结构，倒不如说LSTM是RNN的一个魔改组件，把上面看到的网络中的小圆圈换成下面的LSTM的结构




```
import input_data mnist = input_data.read_data_sets("/tmp/data/", one_hot=True) import tensorflow as tf from tensorflow.models.rnn import rnn, rnn_cell import numpy as np ''' To classify images using a reccurent neural network, we consider every image row as a sequence of pixels. Because MNIST image shape is 28*28px, we will then handle 28 sequences of 28 steps for every sample. ''' # Parameters learning_rate = 0.001 training_iters = 100000 batch_size = 128 display_step = 10 # Network Parameters n_input = 28 # MNIST data input (img shape: 28*28) n_steps = 28 # timesteps n_hidden = 128 # hidden layer num of features n_classes = 10 # MNIST total classes (0-9 digits) # tf Graph input x = tf.placeholder("float", [None, n_steps, n_input]) # Tensorflow LSTM cell requires 2x n_hidden length (state & cell) istate = tf.placeholder("float", [None, 2*n_hidden]) y = tf.placeholder("float", [None, n_classes]) # Define weights weights = {     'hidden': tf.Variable(tf.random_normal([n_input, n_hidden])), # Hidden layer weights     'out': tf.Variable(tf.random_normal([n_hidden, n_classes])) } biases = {     'hidden': tf.Variable(tf.random_normal([n_hidden])),     'out': tf.Variable(tf.random_normal([n_classes])) } def RNN(_X, _istate, _weights, _biases):          _X = tf.transpose(_X, [1, 0, 2])  # permute n_steps and batch_size          _X = tf.reshape(_X, [-1, n_input]) # (n_steps*batch_size, n_input)          _X = tf.matmul(_X, _weights['hidden']) + _biases['hidden']          lstm_cell = rnn_cell.BasicLSTMCell(n_hidden, forget_bias=1.0)          _X = tf.split(0, n_steps, _X) # n_steps * (batch_size, n_hidden)          outputs, states = rnn.rnn(lstm_cell, _X, initial_state=_istate)     # Get inner loop last output      return tf.matmul(outputs[-1], _weights['out']) + _biases['out'] pred = RNN(x, istate, weights, biases) # Define loss and optimizer cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y)) # Softmax loss optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # Adam Optimizer # Evaluate model correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1)) accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32)) # Initializing the variables init = tf.initialize_all_variables() # Launch the graph with tf.Session() as sess:     sess.run(init)     step = 1
```

```
    # Keep training until reach max iterations     while step * batch_size < training_iters:         batch_xs, ba
tch_ys = mnist.train.next_batch(batch_size)         # Reshape data to get 28 seq of 28 elements         batc
h_xs = batch_xs.reshape((batch_size, n_steps, n_input))         # Fit training using batch data         sess.ru
n(optimizer, feed_dict={x: batch_xs, y: batch_ys,                                                 istate: np.zeros((batch_siz
e, 2*n_hidden))})         if step % display_step == 0:             # Calculate batch accuracy             acc = ses
s.run(accuracy, feed_dict={x: batch_xs, y: batch_ys,                                                 istate: np.zeros((batc
h_size, 2*n_hidden))})             # Calculate batch loss             loss = sess.run(cost, feed_dict={x: batch_x
s, y: batch_ys,                                                 istate: np.zeros((batch_size, 2*n_hidden))})             print "Ite
r " + str(step*batch_size) + ", Minibatch Loss= " + "{:.6f}".format(loss) +                 ", Training Accurac
y= " + "{:.5f}".format(acc)         step += 1     print "Optimization Finished!"     # Calculate accuracy for 2
56 mnist test images     test_len = 256     test_data = mnist.test.images[:test_len].reshape((-1, n_step
s, n_input))     test_label = mnist.test.labels[:test_len]     print "Testing Accuracy:", sess.run(accuracy, fee
d_dict={x: test_data, y: test_label,                                                 istate: np.zeros((test_len, 2*n_h
idden))})
```

下面我们来运行测试。

```
root@iZu1cdurunpZ:~/tensorflowtest# python rnn.py
('Extracting', '/tmp/data/train-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/train-labels-idx1-ubyte.gz')
('Extracting', '/tmp/data/t10k-images-idx3-ubyte.gz')
('Extracting', '/tmp/data/t10k-labels-idx1-ubyte.gz')
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parall
elism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op para
llelism threads: 1
Iter 1280, Minibatch Loss= 1.750873, Training Accuracy= 0.42969
Iter 2560, Minibatch Loss= 1.449627, Training Accuracy= 0.45312
Iter 3840, Minibatch Loss= 1.365919, Training Accuracy= 0.51562
Iter 5120, Minibatch Loss= 1.035326, Training Accuracy= 0.61719
Iter 6400, Minibatch Loss= 1.060196, Training Accuracy= 0.67188
Iter 7680, Minibatch Loss= 1.205035, Training Accuracy= 0.57031
```

```
Iter 35840, Minibatch Loss= 0.271101, Training Accuracy= 0.90625
Iter 37120, Minibatch Loss= 0.243600, Training Accuracy= 0.93750
Iter 38400, Minibatch Loss= 0.256686, Training Accuracy= 0.94531
Iter 39680, Minibatch Loss= 0.255223, Training Accuracy= 0.91406
Iter 40960, Minibatch Loss= 0.473043, Training Accuracy= 0.81250
Iter 42240, Minibatch Loss= 0.225937, Training Accuracy= 0.90625
Iter 43520, Minibatch Loss= 0.211771, Training Accuracy= 0.93750
Iter 44800, Minibatch Loss= 0.262842, Training Accuracy= 0.89844
Iter 46080, Minibatch Loss= 0.193546, Training Accuracy= 0.92188
Iter 47360, Minibatch Loss= 0.275067, Training Accuracy= 0.91406
Iter 48640, Minibatch Loss= 0.303396, Training Accuracy= 0.91406
Iter 49920, Minibatch Loss= 0.273619, Training Accuracy= 0.92188
Iter 51200, Minibatch Loss= 0.290226, Training Accuracy= 0.91406
Iter 52480, Minibatch Loss= 0.268248, Training Accuracy= 0.91406
Iter 53760, Minibatch Loss= 0.112870, Training Accuracy= 0.95312
Iter 55040, Minibatch Loss= 0.465606, Training Accuracy= 0.84375
Iter 56320, Minibatch Loss= 0.284747, Training Accuracy= 0.93750
Iter 57600, Minibatch Loss= 0.295907, Training Accuracy= 0.89062
Iter 58880, Minibatch Loss= 0.371059, Training Accuracy= 0.85156
Iter 60160, Minibatch Loss= 0.281683, Training Accuracy= 0.93750
Iter 61440, Minibatch Loss= 0.258522, Training Accuracy= 0.89844
Iter 62720, Minibatch Loss= 0.188283, Training Accuracy= 0.95312
Iter 64000, Minibatch Loss= 0.150253, Training Accuracy= 0.94531
Iter 65280, Minibatch Loss= 0.211269, Training Accuracy= 0.94531
Iter 66560, Minibatch Loss= 0.106427, Training Accuracy= 0.96094
Iter 67840, Minibatch Loss= 0.175069, Training Accuracy= 0.92188
Iter 69120, Minibatch Loss= 0.141579, Training Accuracy= 0.95312
```

```
Iter 69120, Minibatch Loss= 0.141579, Training Accuracy= 0.95312
Iter 70400, Minibatch Loss= 0.164540, Training Accuracy= 0.93750
Iter 71680, Minibatch Loss= 0.179964, Training Accuracy= 0.90625
Iter 72960, Minibatch Loss= 0.251201, Training Accuracy= 0.94531
Iter 74240, Minibatch Loss= 0.141025, Training Accuracy= 0.95312
Iter 75520, Minibatch Loss= 0.195839, Training Accuracy= 0.92969
Iter 76800, Minibatch Loss= 0.150200, Training Accuracy= 0.96094
Iter 78080, Minibatch Loss= 0.106678, Training Accuracy= 0.97656
Iter 79360, Minibatch Loss= 0.325763, Training Accuracy= 0.90625
Iter 80640, Minibatch Loss= 0.184060, Training Accuracy= 0.93750
Iter 81920, Minibatch Loss= 0.152701, Training Accuracy= 0.95312
Iter 83200, Minibatch Loss= 0.075379, Training Accuracy= 0.97656
Iter 84480, Minibatch Loss= 0.093469, Training Accuracy= 0.96094
Iter 85760, Minibatch Loss= 0.140789, Training Accuracy= 0.95312
Iter 87040, Minibatch Loss= 0.113920, Training Accuracy= 0.96094
Iter 88320, Minibatch Loss= 0.120979, Training Accuracy= 0.96094
Iter 89600, Minibatch Loss= 0.120113, Training Accuracy= 0.95312
Iter 90880, Minibatch Loss= 0.039172, Training Accuracy= 1.00000
Iter 92160, Minibatch Loss= 0.152684, Training Accuracy= 0.94531
Iter 93440, Minibatch Loss= 0.072496, Training Accuracy= 0.97656
Iter 94720, Minibatch Loss= 0.149113, Training Accuracy= 0.96875
Iter 96000, Minibatch Loss= 0.060112, Training Accuracy= 0.97656
Iter 97280, Minibatch Loss= 0.120189, Training Accuracy= 0.96094
Iter 98560, Minibatch Loss= 0.062778, Training Accuracy= 0.97656
Iter 99840, Minibatch Loss= 0.045261, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.953125
root@iZu1cdurunpZ:~/tensorflowtest#
```


上一篇：原 TensorFlow人工智能引擎入门教程之八 接着补充一章MLP多层感知器网络原理以及 使用

下一篇：TensorFlow人工智能引擎入门教程之十 最强网络 RSNN深度残差网络 平均准确率96-99%