TensorFlow人工智能入门教程之十一 最强网络DLSTM 双向长短期记忆网络（阿里小A...

2016-05-15 蓝莓对冲... 阅 7

失眠。。。。上一章 讲了 最强网络之一 RSNN 深度残差网络 这一章节 我们来讲讲 还有一个很强的网络模型，就是双向LSTM 也就是前一阵阿里吹牛逼的小AI的 实现的一个重要网络部分，当然实际上 比这还要复杂 层数 以及 多个网络配合，其实就好像 alphaGo 一样，其实多个网络配合 多层 复用 效果是最好的，这就像 我们有大脑第一中枢系统 ，但是我们脊髓 是第二中枢系统一样，脊髓可以控制我们身体的某些肌肉 关节运动，与大脑相互配合调节，通过神经传输 相互传递信息，互相配合调节，大脑为主 脊髓为辅。

最近在学钢琴，那真难。有些东西境界到的人 懂的人自然会懂。所以我博客分享一下我的理解，这都是自己自学摸索研究的东西，主要一是 希望可以给自己 做个整理，无聊写写东西，其实这些东西 对我来说都是不重要的东西，但是可以让大家 学习了解下人工智能，人工智能 就这么点么，这是基础，前面所有章节全部是基础 ，基础知识，你全部掌握了这些，你还只是一个门外汉，最主要的是要能够熟练的使用 ，无论是用来做什么，随心所欲，因地制宜，能够知道怎么运用，这才是最重要的。所以我把这些对我来说还算很简单的知识吧，这里以及后面,至于方向，我将的东西也许有些是自己的理解，但是绝对不会影响大家的使用，本人去年一年创业 就是使用tensorflow ，然后把它在spark上整合实现了，重新改写了bp反馈层 ff前向层 同时改写了部分代码、实现了0.6时候的tensorflow的与spark 并行 训练，所以对人工智能方面 也许没有很好的数学基础，但是对代码 对理解方面 还是算可以的吧。创业项目基本就是人工智能的运用 以及 使用。

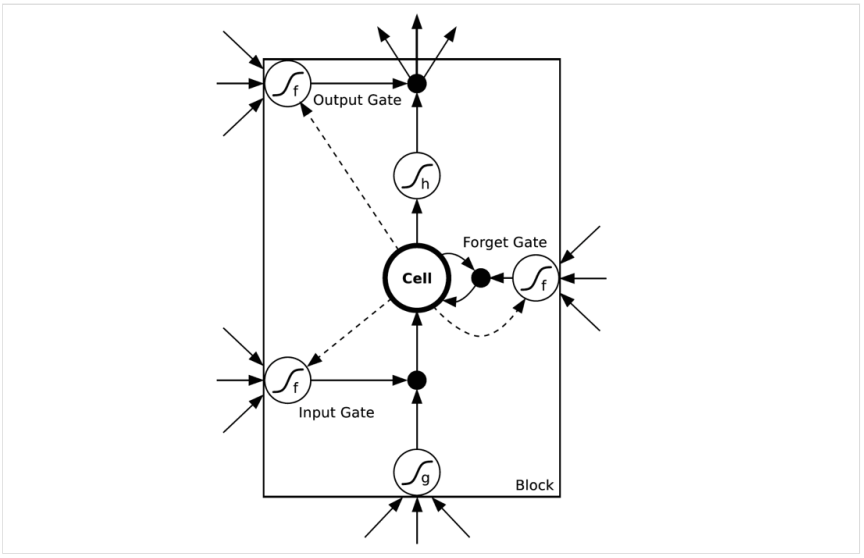
双向LSTM 阿里的小AI 就是使用它，我估计是使用了双向LSTM 之后接着一个RNN层并 增强学习。 但是小AI 里面最重要的还是这个双向LSTM，结合RNN 结合 其他的几种网络还有增强学习 .

LSTM 是为了解决 RNN的一些问题，对隐藏层进行改进，让前面上下文信息 能够有更远的印象，也就是记忆，

LSTM网络本质还是RNN网络，基于LSTM的RNN架构上的变化有最先的BRNN（双向）

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LSTM的block，就是所谓的LSTM了。那它的block长什么样子呢？

1. Cell，就是我们的小本子，有个叫做state的参数东西来记事儿的
2. Input Gate，Output Gate，在参数输入输出的时候起点作用，算一算东西
3. Forget Gate：遗忘门 就像人体的遗忘曲线一样，正是因为遗忘的调节才能知道 那些更重要，因为原始的LSTM在这个位置就是一个值1，是连接到下一时间的那个参数，以前的事情记太牢了，最近的就不住就不好了，所以要选择性遗忘一些东西。通过遗忘进行调节，这样知道那些更重要。那些值得记忆。



上上一章我们讲了RNN/LSTM 的使用，所以 那些操作 不理解的可以到上上一章去看。

这里讲一下双向LSTM

LSTM网络本质还是RNN网络，基于LSTM的RNN架构上的变化有最先的BRNN（双向）

在大多数 应用里面 NLP 自动问答 基于时间有关的 上下文有关的，一般都是双向LSTM+LSTM/RNN横向扩展 来实现的， 效果非常好。好像国内很多吹逼的 都是这样的机构实现的，虽然叫的名字不同但是 其实是一个东西。

双向LSTM 顾名思义采用了 能够双向的LSTM cell单元。是的每次能够访问 下文 也能访问下文

下面看看BIRNN的结构

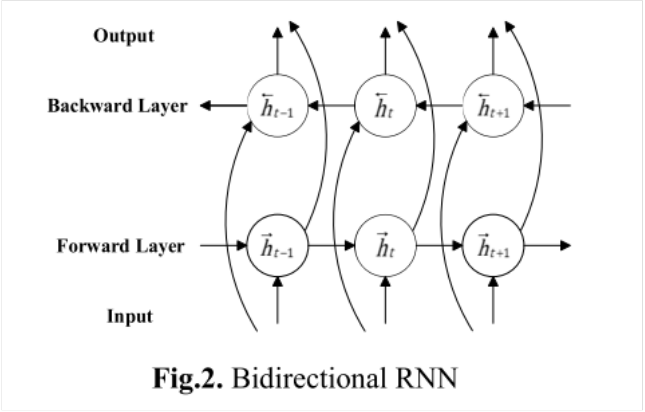


Fig.2. Bidirectional RNN

而 LSTM 我们上面讲了 其实就是RNN 把其中的组件部位换了 加上了cell 也就是记忆单元。所以双向LSTM

就是把上面双向RNN 里面h 那些园的单元全部换成LSTM单元 就是双向LSTM. 阿里的小AI 就是使用它，我估计是使用了双向LSTM 之后接着一个RNN层 吧。 但是小AI 里面最重要的还是这个双向LSTM，结合RNN 结合 其他的几种网络 还有增强学习 .

双向LSTM 在tensorflow中 与 上上篇文章 不同的地方就是

我们直接使用rnn.rnn 来构建RNN 然后传入的LSTMcell（单元） ，这里双向是

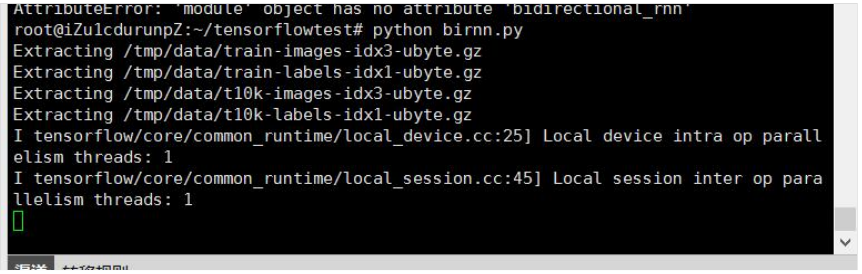
rnn.bidirectional_rnn

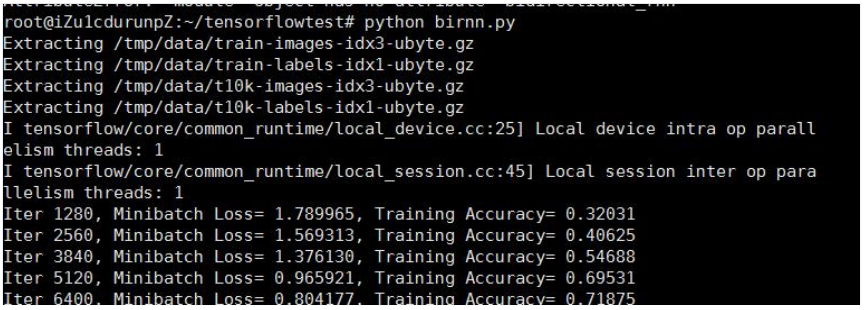
其他基本与上上章基本相同 ，替换一下 稍微修改下即可，不理解的可以跳回去 看看 上上章 LSTM/RNN的内容

下面贴出 示例代码

```
import input_data mnist = input_data.read_data_sets("/tmp/data/", one_hot=True) import tensorflow as tf f
rom tensorflow.python.ops.constant_op import constant from tensorflow.models.rnn import rnn, rnn_cell im
port numpy as np # Parameters learning_rate = 0.001 training_iters = 100000 batch_size = 128 display_ste
p = 10 # Network Parameters n_input = 28 # MNIST data input (img shape: 28*28) n_steps = 28 # timeste
ps n_hidden = 128 # hidden layer num of features n_classes = 10 # MNIST total classes (0-9 digits) # tf Gra
ph input x = tf.placeholder("float", [None, n_steps, n_input]) # Tensorflow LSTM cell requires 2x n_hidden le
ngth (state & cell) istate_fw = tf.placeholder("float", [None, 2*n_hidden]) istate_bw = tf.placeholder("floa
t", [None, 2*n_hidden]) y = tf.placeholder("float", [None, n_classes]) # Define weights weights = {     # Hidde
n layer weights => 2*n_hidden because of foward + backward cells     'hidden': tf.Variable(tf.random_norma
l([n_input, 2*n_hidden])),     'out': tf.Variable(tf.random_normal([2*n_hidden, n_classes])) } biases = {     'hidde
n': tf.Variable(tf.random_normal([2*n_hidden])),     'out': tf.Variable(tf.random_normal([n_classes])) } def BiRN
N(_X, _istate_fw, _istate_bw, _weights, _biases, _batch_size, _seq_len):     # BiRNN requires to supply sequenc
e_length as [batch_size, int64]     # Note: Tensorflow 0.6.0 requires BiRNN sequence_length parameter to b
e set     # For a better implementation with latest version of tensorflow, check below     _seq_len = tf.fill([_ba
tch_size], constant(_seq_len, dtype=tf.int64))     # input shape: (batch_size, n_steps, n_input)     _X = tf.transp
ose(_X, [1, 0, 2])  # permute n_steps and batch_size     # Reshape to prepare input to hidden activation
    _X = tf.reshape(_X, [-1, n_input]) # (n_steps*batch_size, n_input)     # Linear activation     _X = tf.matmu
l(_X, _weights['hidden']) + _biases['hidden']     # Define lstm cells with tensorflow     # Forward direction cell
    lstm_fw_cell = rnn_cell.BasicLSTMCell(n_hidden, forget_bias=1.0)     # Backward direction cell     lstm_b
w_cell = rnn_cell.BasicLSTMCell(n_hidden, forget_bias=1.0)     # Split data because rnn cell needs a list of inp
uts for the RNN inner loop     _X = tf.split(0, n_steps, _X) # n_steps * (batch_size, n_hidden)     # Get lstm cel
l output     outputs = rnn.bidirectional_rnn(lstm_fw_cell, lstm_bw_cell, _X,                                    initia
l_state_fw=_istate_fw,                                    initial_state_bw=_istate_bw,
                                    sequence_length=_seq_len)     # Linear activation     # Get inner loop last outpu
t     return tf.matmul(outputs[-1], _weights['out']) + _biases['out'] pred = BiRNN(x, istate_fw, istate_bw, weigh
ts, biases, batch_size, n_steps) # Define loss and optimizer cost = tf.reduce_mean(tf.nn.softmax_cross_entro
py_with_logits(pred, y)) # Softmax loss optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).mini
mize(cost) # Adam Optimizer # Evaluate model correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1)) acc
uracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32)) # Initializing the variables init = tf.initialize_all_variab
les() # Launch the graph with tf.Session() as sess:     sess.run(init)     step = 1     # Keep training until reach m
ax iterations     while step * batch_size < training_iters:         batch_xs, batch_ys = mnist.train.next_batch(bat
ch_size)         # Reshape data to get 28 seq of 28 elements         batch_xs = batch_xs.reshape((batch_siz
e, n_steps, n_input))         # Fit training using batch data         sess.run(optimizer, feed_dict={x: batch_xs, y: b
atch_ys,                                       istate_fw: np.zeros((batch_size, 2*n_hidden)),                                       istat
e_bw: np.zeros((batch_size, 2*n_hidden))})         if step % display_step == 0:             # Calculate batch accura
cy             acc = sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys,                                                istat
e_fw: np.zeros((batch_size, 2*n_hidden)),                                                    istate_bw: np.zeros((batch_siz
e, 2*n_hidden))})             # Calculate batch loss             loss = sess.run(cost, feed_dict={x: batch_xs, y: batc
h_ys,                                                istate_fw: np.zeros((batch_size, 2*n_hidden)),
                                               istate_bw: np.zeros((batch_size, 2*n_hidden))})             print "Iter " + str(step*ba
tch_size) + ", Minibatch Loss= " + "{:.6f}".format(loss) +                 ", Training Accuracy= " + "{:.5f}".forma
t(acc)         step += 1     print "Optimization Finished!"     # Calculate accuracy for 128 mnist test images
    test_len = 128     test_data = mnist.test.images[:test_len].reshape((-1, n_steps, n_input))     test_label = mn
ist.test.labels[:test_len]     print "Testing Accuracy:", sess.run(accuracy, feed_dict={x: test_data, y: test_label,
                                        istate_fw: np.zeros((test_len, 2*n_hidden)),
                                        istate_bw: np.zeros((test_len, 2*n_hidden))})
```

下面贴出运行测试截图。





上一篇：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%

下一篇：TensorFlow人工智能引擎入门教程之十二 Caffe转换tensorflow并 跨平台调用