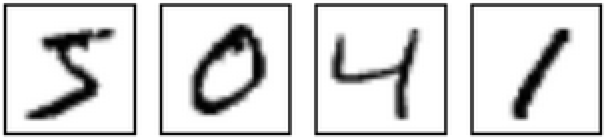
MNIST数据集

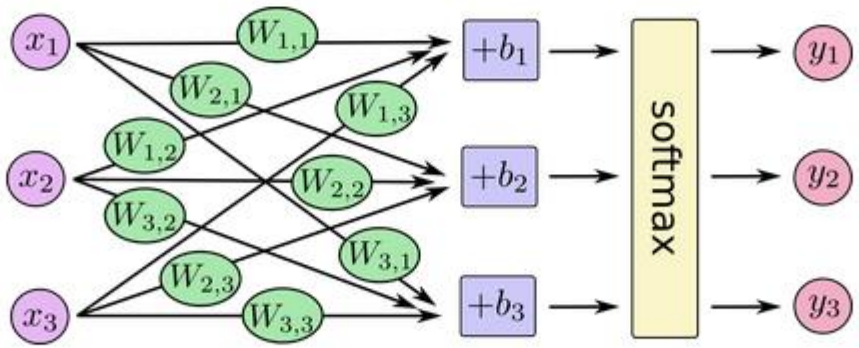
- 基础知识
  - MNIST数据集

    MNIST数据集相当于编程入门的Hello World



Label: 5 0 4 1

  - Softmax回归



MNIST——BP神经网络

- BP神经网络
  - MNIST数据集

```
# 加载MNIST数据集
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

程序报错可以下载数据集放在代码的文件夹内，不要解压

  - 定义输入输出的placeholder

```
# 输入图片为28 x 28 像素 = 784, 其中None表任意值, 特别对应tensor数目
x = tf.placeholder(tf.float32, [None, 784])
# 输出为0-9的one-hot编码
y = tf.placeholder(tf.float32, [None, 10])
```

  - 三层的BP神经网络

```
# 定义参数w和b
# 全连接层的两个参数w和b都是随机初始化的
# hidden layer => w, b
W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1')
b1 = tf.Variable(tf.random_normal([300]), name='b1')
# output layer => w, b
W2 = tf.Variable(tf.random_normal([300, 10], stddev=0.03), name='W2')
b2 = tf.Variable(tf.random_normal([10]), name='b2')

# 构造隐层网络
# hidden layer
hidden_out = tf.add(tf.matmul(x, W1), b1)
hidden_out = tf.nn.relu(hidden_out)

# 构造输出（预测值）
# 计算输出
y_ = tf.nn.softmax(tf.add(tf.matmul(hidden_out, W2), b2))
```

MNIST——BP神经网络

- BP神经网络
  - 损失函数：交叉熵损失函数

```
# BP神经网络部分—定义损失函数
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
```

  - 优化算法：梯度下降算法

```
# BP神经网络部分—定义优化算法
# 创建优化器，确定优化目标，学习率为0.5，最小化交叉熵损失函数
optimizer = tf.train.GradientDescentOptimizer(learning_rate=
                                              learning_rate).minimize(cross_entropy)
```

  - 参数初始化和模型评估

```
# BP神经网络部分—定义初始化operation和准确率node
# init operator
init_op = tf.global_variables_initializer()
# 创建准确率节点，返回一个m乘1的张量，值为T/F
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

MNIST——BP神经网络

- BP神经网络——模型训练与结果

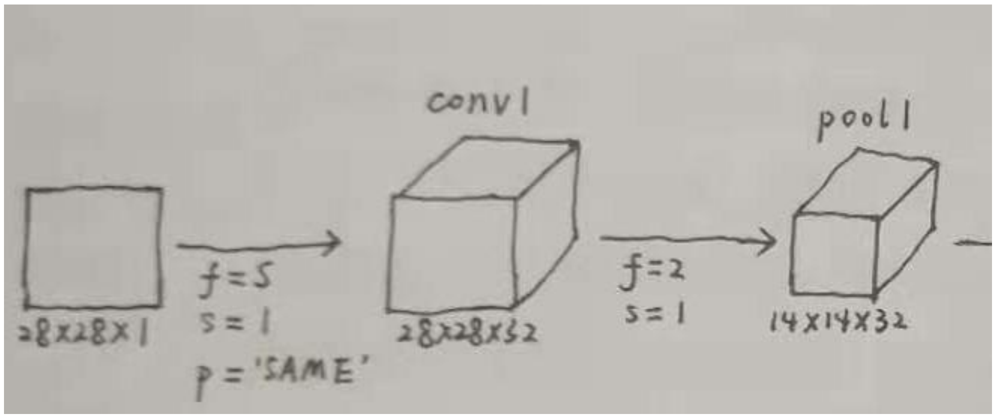
```
sess = tf.Session()
sess.run(init)
# 训练模型，循环1000次
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(optimizer, feed_dict={x: batch_xs, y_: batch_ys})
# 创建准确率节点，返回一个m乘1的张量，值为T/F，tf.argmax(y,1)返回模型预测的类别标签
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
# 任务完成，关闭会话.
sess.close()
```

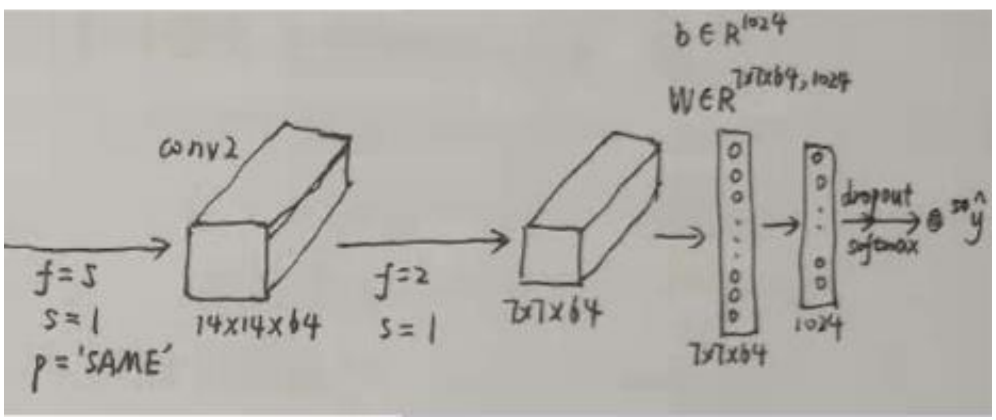
```
# 开始训练
# 创建session
with tf.Session() as sess:
    # 变量初始化
    sess.run(init_op)
    # 设定循环100=batch_size使得total_batch次可以循环结束
    total_batch = int(len(mnist.train.labels) / batch_size)
    for epoch in range(epochs):
        avg_cost = 0
        for i in range(total_batch):
            batch_x, batch_y = mnist.train.next_batch(batch_size=batch_size)
            _, c = sess.run([optimizer, cross_entropy], feed_dict={x: batch_x, y: batch_y})
            avg_cost += c / total_batch
        print("Epoch:", (epoch + 1), "cost = ", "{:.3f}".format(avg_cost))
    print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

```
Epoch: 1 cost =  0.586
Epoch: 2 cost =  0.216
Epoch: 3 cost =  0.158
Epoch: 4 cost =  0.126
Epoch: 5 cost =  0.100
Epoch: 6 cost =  0.082
Epoch: 7 cost =  0.069
Epoch: 8 cost =  0.058
Epoch: 9 cost =  0.046
Epoch: 10 cost =  0.037
0.9783
```

MNIST——CNN

- 卷积神经网络





MNIST——CNN

- 卷积神经网络
  - 定义卷积和池化函数

```
# 定义卷积层，步长为1，SAME输入输出图片一样大
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
# 定义池化层，2乘2的池化窗口，步长为2
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

  - 第一层和第二层卷积

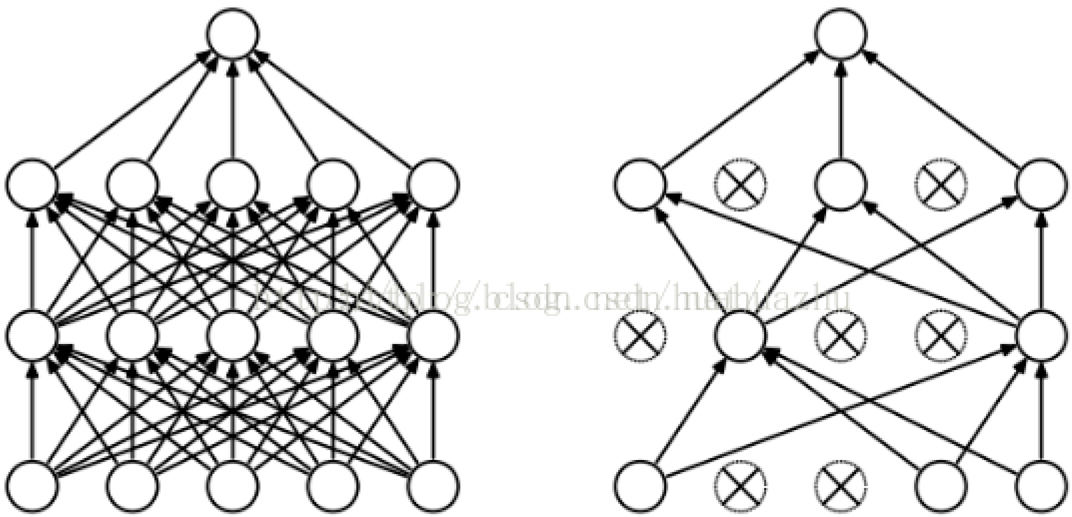
```
# 第一层卷积
# Conv1:32个filter个数，5乘5的卷积核;h_conv1.shape=[-1,28,28,32]
w_conv1 = weight_variable([5,5,1,32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(X,w_conv1) + b_conv1)
# Pool1: 最大值池化层2x2 [-1,28,28,28]->[-1,14,14,32]
h_pool1 = max_pool_2x2(h_conv1)

# 第二层卷积
# Conv2:64个filter个数，5乘5的卷积核;h_conv2.shape=[-1,14,14,64]
w_conv2 = weight_variable([5,5,32,64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1,w_conv2) + b_conv2)
# Pool2:最大值池化层2x2 [-1,14,14,64]->[-1,7,7,64]
h_pool2 = max_pool_2x2(h_conv2)
```

MNIST——CNN

- 卷积神经网络
  - Dropout

    Dropout就是在不同的训练过程中随机扔掉一部分神经元。也就是让某个神经元的激活值以一定的概率p，让其停止工作。



(a) Standard Neural Net　　(b) After applying dropout.

但在测试及验证中：每个神经元都要参加运算，但其输出要乘以概率p。

MNIST——CNN

- 卷积神经网络
  - Dropout

```python
# dropout
# 使输入tensor中某些元素变为0，其它没变0的元素变为原来的1/keep_prob大小
# 用一个placeholder来代表一个神经元的输出在dropout中保持不变的概率
keep_prob = tf.placeholder(tf.float32)    #弃权概率0.0-1.0  1.0表示不使用弃权
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# 输出层
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

sess.run(tf.initialize_all_variables())
# sess.run(tf.global_variables_initializer())

for i in range(20000):
  batch = mnist.train.next_batch(50)
  if i%100 == 0:
      # 在feed_dict中加入额外的参数keep_prob来控制dropout比例
    train_accuracy = accuracy.eval(feed_dict={
        x_:batch[0], y_: batch[1], keep_prob: 1.0})
    print ("step %d, training accuracy %g"%(i, train_accuracy))
  train.run(feed_dict={x_: batch[0], y_: batch[1], keep_prob: 0.5})

print ("test accuracy %g"%accuracy.eval(feed_dict={
    x_: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
sess.close()
```

MNIST——CNN

- 卷积神经网络
  - ADAM优化器来做梯度最速下降(85行)

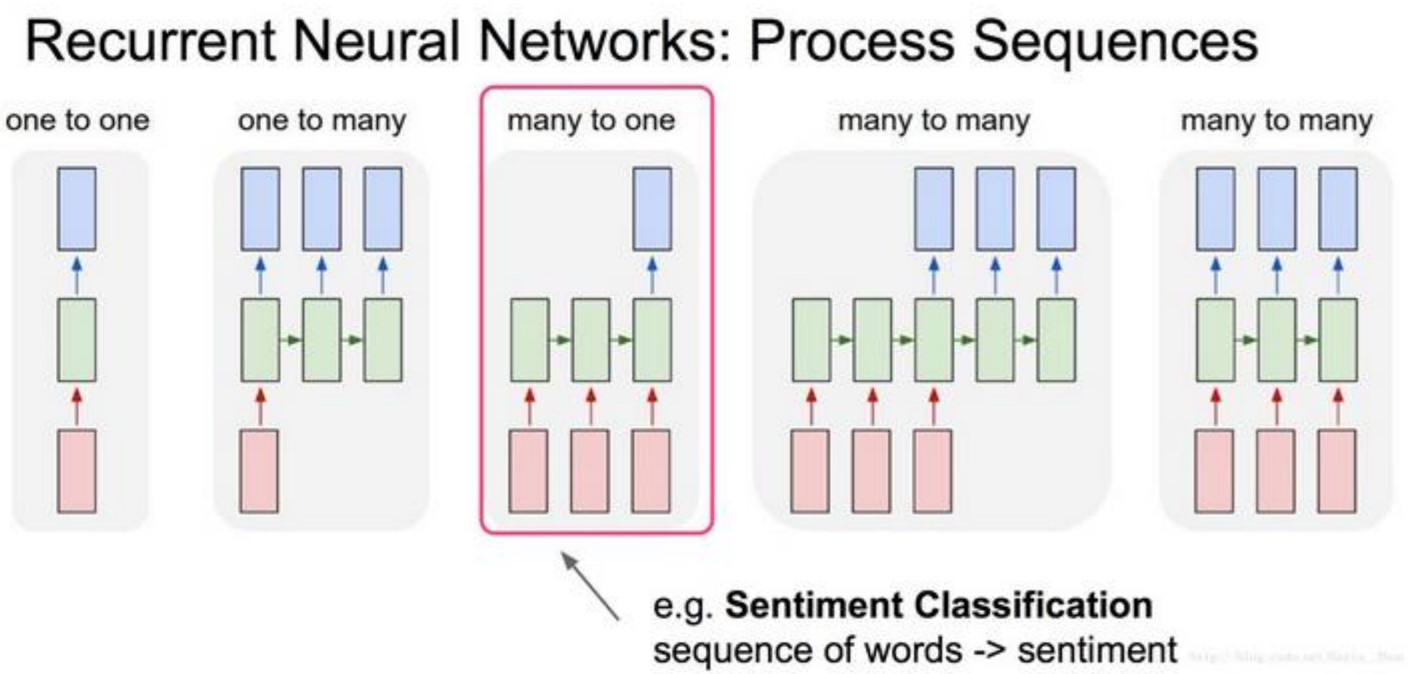
    train = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
  - 模型结果

```
In[18]: for i in range(20000):
   ...:     batch = mnist.train.next_batch(50)
   ...:     if i%100 == 0:
   ...:         train_accuracy = accuracy.eval(feed_dict={
   ...:             x_:batch[0], y_: batch[1], keep_prob: 1.0})
   ...:         print ("step %d, training accuracy %g"%(i, train_accuracy))
   ...:     train.run(feed_dict={x_: batch[0], y_: batch[1], keep_prob: 0.5})
   ...:
step 0, training accuracy 0.1
step 100, training accuracy 0.88
step 200, training accuracy 0.9
step 300, training accuracy 0.9
step 400, training accuracy 0.96
step 500, training accuracy 0.92
```

```
step 9300, training accuracy 1
step 9400, training accuracy 1
step 9500, training accuracy 1
step 9600, training accuracy 1
step 9700, training accuracy 0.98
step 9800, training accuracy 1
step 9900, training accuracy 1
step 10000, training accuracy 1
step 10100, training accuracy 0.98
```

MNIST——RNN & LSTM

- Sequence Classification



图像的分类对应上图就是个many to one的问题. 对于mnist来说其图像的size是28*28，如果将其看成28个step，每个step的size是28的话，刚好符合上图。

搭建LSTM

- 步骤1：把784个点的字符信息还原成 28 * 28 的图片

```
# **步骤1：RNN 的输入shape = (batch_size, timestep_size, input_size)
X = tf.reshape(X_input, [-1, 28, 28])
```

- 步骤2-3：定义多层LSTM模型

```
stacked_rnn = []
for iiLyr in range(layer_num):
    # **步骤2：定义一层 LSTM_cell，只需要说明 hidden_size, 它会自动匹配输入的 X 的维度
    stacked_rnn.append(tf.nn.rnn_cell.LSTMCell(num_units=hidden_size, state_is_tuple=True))
# **步骤3：调用 MultiRNNCell 来实现多层 LSTM
mlstm_cell = tf.nn.rnn_cell.MultiRNNCell(cells=stacked_rnn, state_is_tuple=True)
```

- tf.nn.rnn_cell.BscicLSTMCell

```
tf.nn.rnn_cell.BasicLSTMCell(n_hidden, forget_bias=1.0, state_is_tuple=True)
n_hidden 表示神经元个数
forget_bias LSTM忘记系数：1.0=不会忘记信息，0.0=都忘记
state_is_tuple 输出为元祖，zero_state（batch_size，dtype）
```

- tf.nn.rnn_cell.MultiRNNCell

```
tf.nn.rnn_cell.MultiRNNCell([list RNNcell], state_is_tuple=True)
[list RNNcell] RNN神经元组成的列表
```

MNIST——RNN & LSTM

搭建LSTM

- 步骤4：用全零来初始化state

```
# **步骤4：用全零来初始化state
init_state = mlstm_cell.zero_state(batch_size, dtype=tf.float32)
```

- 步骤5：方法一：按时间步展开计算

```
# **步骤5：按时间步展开计算
outputs = list()
state = init_state
with tf.variable_scope('RNN'):
    for timestep in range(timestep_size):
        (cell_output, state) = mlstm_cell(X[:, timestep, :],state)
        outputs.append(cell_output)
h_state = outputs[-1]
```

- 方法二：tf.nn.dynamic_rnn

```
outputs, state = tf.nn.dynamic_rnn(cell,inputs,...)
cell 构建的多层RNN神经网络；inputs 输入的变量X
outputs 最后一层每个step的输出值；state 每一层RNN最后一个step的输出
```

MNIST——RNN & LSTM

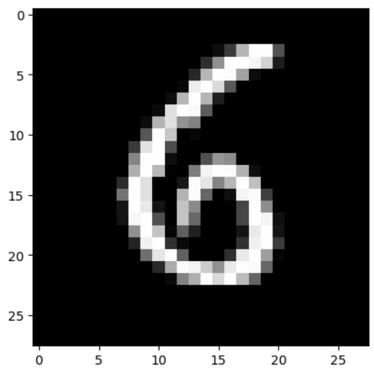
- 模型输出

```python
'''模型训练和测试'''
sess = tf.Session()
sess.run(tf.global_variables_initializer())
time0 = time.time()
for i in range(5000):
    _batch_size=100
    X_batch, y_batch = mnist.train.next_batch(batch_size=_batch_size)
    cost, acc,  _ = sess.run([cross_entropy, accuracy, train_op], feed_dict={X_input: X_batch,
                                                 y_input: y_batch, keep_prob: 0.5, batch_size: _batch_size})
    if (i+1) % 500 == 0:
        # 分 100 个batch 迭代
        test_acc = 0.0
        test_cost = 0.0
```

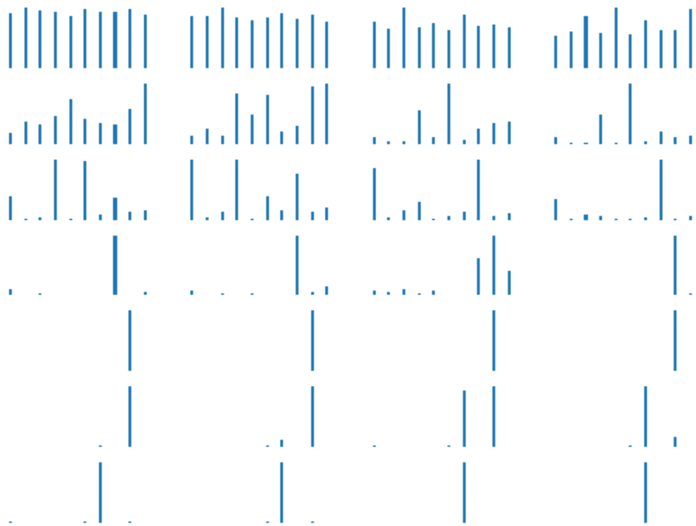
```
...:         print("step {}, train cost={:.6f}, acc={:.6f}; test cost={:.6f}, acc={:.6f}; pass {}s".format(i+1, cost,
...:         time0 = time.time()
...:
step 500, train cost=0.010285, acc=0.970000; test cost=0.012022, acc=0.962800; pass 346.39578437805176s
step 1000, train cost=0.001810, acc=1.000000; test cost=0.009855, acc=0.969400; pass 322.5399327278137s
step 1500, train cost=0.017293, acc=0.970000; test cost=0.009581, acc=0.970900; pass 322.7249677181244s
step 2000, train cost=0.001345, acc=1.000000; test cost=0.006592, acc=0.980400; pass 323.03099966049194s
step 2500, train cost=0.000591, acc=1.000000; test cost=0.005483, acc=0.985100; pass 323.7637815475464s
step 3000, train cost=0.009209, acc=0.970000; test cost=0.004949, acc=0.984800; pass 319.5827646255493s
step 3500, train cost=0.000296, acc=1.000000; test cost=0.004581, acc=0.986700; pass 323.9109010696411s
step 4000, train cost=0.001038, acc=1.000000; test cost=0.004305, acc=0.986900; pass 345.9888160228729s
step 4500, train cost=0.002839, acc=0.980000; test cost=0.004406, acc=0.988400; pass 352.09626865386963s
step 5000, train cost=0.001505, acc=0.990000; test cost=0.004181, acc=0.986900; pass 343.2950654029846s
```

MNIST——RNN & LSTM

- LSTM可视化
  - 任找一张图片(123-126行)



  - 可视化



搭建LSTM基本操作

- 基本单元
- tf.contrib.rnn.BasicRnnCell (num_units,activation)
- tf.contrib.rnn.BasicLSTMCell (num_units, forget_bias,activation,state_is_tuple=True)
- tf.contrib.rnn.GRUCell(num_units,activation)
- forget_bias:偏置； activation:激活函数。
- Dropout
- tf.contrib.rnn.DropoutWrapper (cell, keep_prob)
- 多层神经网络
- rnn_cell = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True)
- 运行
- outpusts,state = tf.nn.dynamic_rnn(cell=rnn_cell, inputs=embedding_input）
- 全连接层
- dense=tf.layers.dense(inputs, units=1024, activation=tf.nn.relu) #inputs二维张量
- tf.contrib.layers.dropout(dense, keep_prob) #全连接层dropout

搭建LSTM基本操作

- 基本单元
- tf.nn.rnn_cell.BasicRnnCell (num_units,activation)
- tf.nn.rnn_cell.GRUCell (num_units,activation)
- tf.nn.rnn_cell.BasicLSTMCell (num_units, forget_bias,activation,state_is_tuple=True)
- forget_bias:偏置； activation:激活函数。
- Dropout
- tf.nn.rnn_cell.DropoutWrapper(cell, keep_prob)
- 多层神经网络
- tf.nn.rnn_cell.MultiRNNCell(cells=cells)
- 一些函数
- tf.nn.relu()   tf.nn.softmax   tf.not_equal   tf.arg_max   tf.reduce_mean