```
import tensorflow as tf
#from tensorflow.contrib import rnn
from tensorflow.examples.tutorials.mnist import input_data   #手写数字相关的数据包
```

```
# 载入数据集
mnist = input_data.read_data_sets("MNIST_data",one_hot=True)    #载入数据，{数据集包路径，把标签转化为只有0和1的形式}

#输入图片是28*28
n_inputs = 28  #输入一行，一行有28个数据（有28个神经元）
max_time = 28  #一共28行（输入28次）
lstm_size = 100  #隐藏单元（block）
n_classes = 10  #10个分类（0-9）
batch_size = 50    #一次放50个样本进去
n_batch = mnist.train.num_examples // batch_size   #计算一共有多少个批次；训练集数量（整除）一个批次大小

#定义两个placeholder
x = tf.placeholder(tf.float32,[None,784])    #[行不确定，列为784]
#正确的标签
y = tf.placeholder(tf.float32,[None,10])    #数字为0-9，则为10

#初始化权值
weights = tf.Variable(tf.truncated_normal([lstm_size, n_classes], stddev=0.1))
#初始化偏置值
biases = tf.Variable(tf.constant(0.1, shape=[n_classes]))

#定义RNN网络
def RNN(X,weights,biases):
    #inputs=[batch_size, max_time, n_inputs]
    inputs = tf.reshape(X,[-1,max_time,n_inputs])
    #定义LSTM基本CELL
    lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(lstm_size)
    #final_state[0]是cell state
    #final_state[1]是hidden_state
    putputs,final_state = tf.nn.dynamic_rnn(lstm_cell,inputs,dtype=tf.float32)
    results = tf.nn.softmax(tf.matmul(final_state[1],weights) + biases)
    return results

#计算RNN的返回结果
prediction = RNN(x, weights, biases)    #预测

#定义交叉熵代价函数
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=prediction))
#使用AdamOptimizer进行优化
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

```

```
#准确数，结果存放在一个布尔型列表中
correct_prediction = tf.equal(tf.argmax(prediction,1),tf.argmax(y,1))   #比较两个参数大小是否相同，同则返回为true，不同则返回为false；argmax()：返回张量中最大的值所在的位置

#求准确率
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
#cast()：将布尔型转换为32位的浮点型；（比方说9个T和1个F，则为9个1，1个0，即准确率为90%）


#初始化变量
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for epoch in range(21):
        for batch in range(n_batch):
            batch_xs,batch_ys = mnist.train.next_batch(batch_size)
            sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys})

        acc = sess.run(accuracy,feed_dict={x:mnist.test.images,y:mnist.test.labels})
        print("Iter" + str(epoch) + ",Testing Accuracy" + str(acc))

```

```
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
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-6-01b19bef12bb> in <module>
     33
     34 #计算RNN的返回结果
---> 35 prediction = RNN(x, weights, biases)    #预测
     36
     37 #定义交叉熵代价函数
```

```
<ipython-input-6-01b19bef12bb> in RNN(X, weights, biases)
     28     #final_state[0]是cell state
     29     #final_state[1]是hidden_state
---> 30     putputs,final_state = tf.nn.dynamic_rnn(lstm_cell,inputs,dtype=tf.float32)
     31     results = tf.nn.softmax(tf.matmul(final_state[1],weights) + biases)
     32     return results
```

```
D:\anaconda\lib\site-packages\tensorflow\python\util\deprecation.py in new_func(*args, **kwargs)
    322               'in a future version' if date is None else ('after %s' % date),
    323               instructions)
--> 324       return func(*args, **kwargs)
    325     return tf_decorator.make_decorator(
    326         func, new_func, 'deprecated',
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\rnn.py in dynamic_rnn(cell, inputs, sequence_length, initial_state, dtype, parallel_iterations, swap_memory, time_major, scope)
    705         swap_memory=swap_memory,
    706         sequence_length=sequence_length,
--> 707         dtype=dtype)
    708
    709     # Outputs of _dynamic_rnn_loop are always shaped [time, batch, depth].
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\rnn.py in _dynamic_rnn_loop(cell, inputs, initial_state, parallel_iterations, swap_memory, sequence_length, dtype)
    914       parallel_iterations=parallel_iterations,
    915       maximum_iterations=time_steps,
--> 916       swap_memory=swap_memory)
    917
    918   # Unpack final output if not using output tuples.
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\control_flow_ops.py in while_loop(cond, body, loop_vars, shape_invariants, parallel_iterations, back_prop, swap_memory, name, maximum_iterations, return_same_structure)
   3499       ops.add_to_collection(ops.GraphKeys.WHILE_CONTEXT, loop_context)
   3500     result = loop_context.BuildLoop(cond, body, loop_vars, shape_invariants,
-> 3501                                     return_same_structure)
   3502     if maximum_iterations is not None:
   3503       return result[1]
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\control_flow_ops.py in BuildLoop(self, pred, body, loop_vars, shape_invariants, return_same_structure)
   3010       with ops.get_default_graph()._mutation_lock():  # pylint: disable=protected-access
   3011         original_body_result, exit_vars = self._BuildLoop(
-> 3012             pred, body, original_loop_vars, loop_vars, shape_invariants)
   3013     finally:
   3014       self.Exit()
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\control_flow_ops.py in _BuildLoop(self, pred, body, original_loop_vars, loop_vars, shape_invariants)
   2935         expand_composites=True)
   2936     pre_summaries = ops.get_collection(ops.GraphKeys._SUMMARY_COLLECTION)  # pylint: disable=protected-access
-> 2937     body_result = body(*packed_vars_for_body)
   2938     post_summaries = ops.get_collection(ops.GraphKeys._SUMMARY_COLLECTION)  # pylint: disable=protected-access
   2939     if not nest.is_sequence_or_composite(body_result):
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\control_flow_ops.py in <lambda>(i, lv)
   3454         cond = lambda i, lv: (  # pylint: disable=g-long-lambda
   3455             math_ops.logical_and(i < maximum_iterations, orig_cond(*lv)))
-> 3456         body = lambda i, lv: (i + 1, orig_body(*lv))
   3457
   3458     if executing_eagerly:
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\rnn.py in _time_step(time, output_ta_t, state)
    882           skip_conditionals=True)
    883     else:
--> 884       (output, new_state) = call_cell()
    885
    886     # Keras cells always wrap state as list, even if it's a single tensor.
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\rnn.py in <lambda>()
    868     if is_keras_rnn_cell and not nest.is_sequence(state):
    869       state = [state]
--> 870     call_cell = lambda: cell(input_t, state)
    871
    872     if sequence_length is not None:
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in __call__(self, inputs, state, scope, *args, **kwargs)
    383     # method.  See the class docstring for more details.
    384     return base_layer.Layer.__call__(
--> 385         self, inputs, state, scope=scope, *args, **kwargs)
    386
    387
```

```
D:\anaconda\lib\site-packages\tensorflow\python\layers\base.py in __call__(self, inputs, *args, **kwargs)
    535
    536       # Actually call layer
--> 537       outputs = super(Layer, self).__call__(inputs, *args, **kwargs)
    538
    539     if not context.executing_eagerly():
```

```
D:\anaconda\lib\site-packages\tensorflow\python\keras\engine\base_layer.py in __call__(self, inputs, *args, **kwargs)
    589           # Build layer if applicable (if the `build` method has been
    590           # overridden).
--> 591           self._maybe_build(inputs)
    592
    593           # Wrapping `call` function in autograph to allow for dynamic control
```

```
D:\anaconda\lib\site-packages\tensorflow\python\keras\engine\base_layer.py in _maybe_build(self, inputs)
   1879       # operations.
   1880       with tf_utils.maybe_init_scope(self):
-> 1881         self.build(input_shapes)
   1882     # We must set self.built since user defined build functions are not
   1883     # constrained to set self.built.
```

```
D:\anaconda\lib\site-packages\tensorflow\python\keras\utils\tf_utils.py in wrapper(instance, input_shape)
    293     if input_shape is not None:
    294       input_shape = convert_shapes(input_shape, to_tuples=True)
--> 295     output_shape = fn(instance, input_shape)
    296     # Return shapes from `fn` as TensorShapes.
    297     if output_shape is not None:
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py in build(self, inputs_shape)
    732     self._kernel = self.add_variable(
    733         _WEIGHTS_VARIABLE_NAME,
--> 734         shape=[input_depth + h_depth, 4 * self._num_units])
    735     self._bias = self.add_variable(
    736         _BIAS_VARIABLE_NAME,
```

```
D:\anaconda\lib\site-packages\tensorflow\python\keras\engine\base_layer.py in add_variable(self, *args, **kwargs)
   1482   def add_variable(self, *args, **kwargs):
   1483     """Alias for `add_weight`."""
-> 1484     return self.add_weight(*args, **kwargs)
   1485
   1486   @property
```

```
D:\anaconda\lib\site-packages\tensorflow\python\layers\base.py in add_weight(self, name, shape, dtype, initializer, regularizer, trainable, constraint, use_resource, synchronization, aggregation, partitioner, **kwargs)
    448             aggregation=aggregation,
    449             getter=vs.get_variable,
--> 450             **kwargs)
    451
    452         if regularizer:
```

```
D:\anaconda\lib\site-packages\tensorflow\python\keras\engine\base_layer.py in add_weight(self, name, shape, dtype, initializer, regularizer, trainable, constraint, partitioner, use_resource, synchronization, aggregation, **kwargs)
    382         collections=collections_arg,
    383         synchronization=synchronization,
--> 384         aggregation=aggregation)
    385     backend.track_variable(variable)
    386
```

```
D:\anaconda\lib\site-packages\tensorflow\python\training\tracking\base.py in _add_variable_with_custom_getter(self, name, shape, dtype, initializer, getter, overwrite, **kwargs_for_getter)
    661         dtype=dtype,
    662         initializer=initializer,
--> 663         **kwargs_for_getter)
    664
    665     # If we set an initializer and the variable processed it, tracking will not
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\variable_scope.py in get_variable(name, shape, dtype, initializer, regularizer, trainable, collections, caching_device, partitioner, validate_shape, use_resource, custom_getter, constraint, synchronization, aggregation)
   1494       constraint=constraint,
   1495       synchronization=synchronization,
-> 1496       aggregation=aggregation)
   1497
   1498
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\variable_scope.py in get_variable(self, var_store, name, shape, dtype, initializer, regularizer, reuse, trainable, collections, caching_device, partitioner, validate_shape, use_resource, custom_getter, constraint, synchronization, aggregation)
   1237           constraint=constraint,
   1238           synchronization=synchronization,
-> 1239           aggregation=aggregation)
   1240
   1241   def _get_partitioned_variable(self,
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\variable_scope.py in get_variable(self, name, shape, dtype, initializer, regularizer, reuse, trainable, collections, caching_device, partitioner, validate_shape, use_resource, custom_getter, constraint, synchronization, aggregation)
    560           constraint=constraint,
    561           synchronization=synchronization,
--> 562           aggregation=aggregation)
    563
    564   def _get_partitioned_variable(self,
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\variable_scope.py in _true_getter(name, shape, dtype, initializer, regularizer, reuse, trainable, collections, caching_device, partitioner, validate_shape, use_resource, constraint, synchronization, aggregation)
    512           constraint=constraint,
    513           synchronization=synchronization,
--> 514           aggregation=aggregation)
    515
    516     synchronization, aggregation, trainable = (
```

```
D:\anaconda\lib\site-packages\tensorflow\python\ops\variable_scope.py in _get_single_variable(self, name, shape, dtype, initializer, regularizer, partition_info, reuse, trainable, collections, caching_device, validate_shape, use_resource, constraint, synchronization, aggregation)
    862         tb = [x for x in tb if "tensorflow/python" not in x[0]][:5]
    863         raise ValueError("%s Originally defined at:\n\n%s" %
--> 864                          (err_msg, """.join(traceback.format_list(tb))))
    865       found_var = self._vars[name]
    866       if not shape.is_compatible_with(found_var.get_shape()):
```

```
ValueError: Variable rnn/basic_lstm_cell/kernel already exists, disallowed. Did you mean to set reuse=True or reuse=tf.AUTO_REUSE in VarScope? Originally defined at:

  File "D:\anaconda\lib\site-packages\tensorflow\python\framework\ops.py", line 2005, in __init__
    self._traceback = tf_stack.extract_stack()
  File "D:\anaconda\lib\site-packages\tensorflow\python\framework\ops.py", line 3616, in create_op
    op_def=op_def)
  File "D:\anaconda\lib\site-packages\tensorflow\python\util\deprecation.py", line 507, in new_func
    return func(*args, **kwargs)
  File "D:\anaconda\lib\site-packages\tensorflow\python\framework\op_def_library.py", line 788, in _apply_op_helper
    op_def=op_def)
  File "D:\anaconda\lib\site-packages\tensorflow\python\ops\gen_state_ops.py", line 1608, in variable_v2
    shared_name=shared_name, name=name)
```

```

```