Deep_networks_convolutional_&_recurrent_architectures

December 6, 2017

0.1 Exercise 06: Deep networks: convolutional & recurrent architectures

Group Name: Alwaysonline

Omar Sherif
Omar Roushdy
Hsiwei Kao
Changbin Lu
Zhanwang Chen

1 (a)

```
In [1]: from tensorflow.examples.tutorials.mnist import input_data
        mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
        import tensorflow as tf
```

```
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
```

```
In [18]: W = tf.Variable(tf.zeros([784, 10]))
         b = tf.Variable(tf.zeros([10]))
         x = tf.placeholder(tf.float32, [None, 784])
         y = tf.nn.softmax(tf.matmul(x, W) + b)
         y_ = tf.placeholder(tf.float32, [None, 10])
         cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
         train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
         sess = tf.InteractiveSession()
         tf.global_variables_initializer().run()
         for _ in range(10000):
           batch_xs, batch_ys = mnist.train.next_batch(100)
           sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

         correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
         print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

0.9196

2 (b)

In [12]:
```
# Network Parameters
n_hidden_1 = 1500
n_hidden_2 = 1500
n_hidden_3 = 1500
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10 # MNIST total classes (0-9 digits)
X = tf.placeholder("float", [None, n_input])
Y = tf.placeholder("float", [None, n_classes])
weights = {
    'h1': tf.Variable(tf.truncated_normal(shape=[n_input, n_hidden_1],mean=0,stddev=0
    'h2': tf.Variable(tf.truncated_normal(shape=[n_hidden_1, n_hidden_2],mean=0,stddev
    'h3': tf.Variable(tf.truncated_normal(shape=[n_hidden_2, n_hidden_3],mean=0,stddev
    'out': tf.Variable(tf.truncated_normal(shape=[n_hidden_3, n_classes],mean=0,stddev
}
biases = {
    'b1': tf.Variable(tf.constant(value=0.1,shape=[n_hidden_1])),
    'b2': tf.Variable(tf.constant(value=0.1,shape=[n_hidden_2])),
    'b3': tf.Variable(tf.constant(value=0.1,shape=[n_hidden_3])),
    'out': tf.Variable(tf.constant(value=0.1,shape=[n_classes]))
}
def multilayer_perceptron(x):
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.relu(layer_2)
    layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
    layer_3 = tf.nn.relu(layer_3)
    # Output fully connected layer with a neuron for each class
    out_layer = tf.nn.softmax(tf.matmul(layer_3, weights['out']) + biases['out'])
    return out_layer

logits = multilayer_perceptron(X)
# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate,
                                   beta1=0.9,beta2=0.999,epsilon=10**(-8))
train_op = optimizer.minimize(loss_op)
# Initializing the variables
init = tf.global_variables_initializer()

learning_rate = 0.001
```

```
training_epochs = 20000
batch_size = 100
display_step = 1000

with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.


        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop) and cost op (to get loss value)
        _, c = sess.run([train_op, loss_op], feed_dict={X: batch_x,
                                                        Y: batch_y})

        if epoch % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost={:.9f}".format(c))
            pred = tf.nn.softmax(logits)  # Apply softmax to logits
            correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(Y, 1))
    # Calculate accuracy
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
            print("Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labe

    print("Optimization Finished!")
        #correct_pred = tf.equal(tf.argmax(y_estimate, 1), tf.argmax(y, 1))
        #accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
        #accuracy.eval({x: mnist.test.images, y: mnist.test.labels, keep_prob: 1.0})
    # Display logs per epoch step


    # Test model
    pred = tf.nn.softmax(logits)  # Apply softmax to logits
    correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(Y, 1))
    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print("Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels}))
```

```
Epoch: 0001 cost=2.302832127
Accuracy: 0.098
Epoch: 1001 cost=1.625870466
Accuracy: 0.7968
Epoch: 2001 cost=1.650844216
Accuracy: 0.8351
Epoch: 3001 cost=1.601058960
Accuracy: 0.8426
Epoch: 4001 cost=1.582948565
```

```
Accuracy: 0.8837
Epoch: 5001 cost=1.601150870
Accuracy: 0.8911
Epoch: 6001 cost=1.641150713
Accuracy: 0.8127
Epoch: 7001 cost=1.611150861
Accuracy: 0.8848
Epoch: 8001 cost=1.601150870
Accuracy: 0.8558
Epoch: 9001 cost=1.641150832
Accuracy: 0.8293
Epoch: 10001 cost=1.551150680
Accuracy: 0.8698
Epoch: 11001 cost=1.611150861
Accuracy: 0.847
Epoch: 12001 cost=1.671150684
Accuracy: 0.8477
Epoch: 13001 cost=1.701112866
Accuracy: 0.7268
Epoch: 14001 cost=1.731150866
Accuracy: 0.7437
Epoch: 15001 cost=1.631150842
Accuracy: 0.8159
Epoch: 16001 cost=1.571150661
Accuracy: 0.8241
Epoch: 17001 cost=1.641150713
Accuracy: 0.7736
Epoch: 18001 cost=1.681150675
Accuracy: 0.8232
Epoch: 19001 cost=1.571150780
Accuracy: 0.8495
Optimization Finished!
Accuracy: 0.842
```

3 (c)

```
In [2]: dropout = 0.75
        learning_rate = 0.001
        training_epochs = 20000
        batch_size = 100
        display_step = 1

        # Network Parameters
        n_hidden_1 = 1500
        n_hidden_2 = 1500
        n_hidden_3 = 1500
```

```
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10 # MNIST total classes (0-9 digits)
X = tf.placeholder("float", [None, n_input])
Y = tf.placeholder("float", [None, n_classes])
weights = {
    'h1': tf.Variable(tf.truncated_normal(shape=[n_input, n_hidden_1],mean=0,stddev=0.
    'h2': tf.Variable(tf.truncated_normal(shape=[n_hidden_1, n_hidden_2],mean=0,stddev=
    'h3': tf.Variable(tf.truncated_normal(shape=[n_hidden_2, n_hidden_3],mean=0,stddev=
    'out': tf.Variable(tf.truncated_normal(shape=[n_hidden_3, n_classes],mean=0,stddev=
}
biases = {
    'b1': tf.Variable(tf.constant(value=0.1,shape=[n_hidden_1])),
    'b2': tf.Variable(tf.constant(value=0.1,shape=[n_hidden_2])),
    'b3': tf.Variable(tf.constant(value=0.1,shape=[n_hidden_3])),
    'out': tf.Variable(tf.constant(value=0.1,shape=[n_classes]))
}
def multilayer_perceptron(x):
    layer_1 = tf.nn.relu(tf.add(tf.matmul(x, weights['h1']), biases['b1']))
    layer_1 = tf.nn.dropout(layer_1,0.5)
    layer_2 = tf.nn.relu(tf.add(tf.matmul(layer_1, weights['h2']), biases['b2']))
    layer_2 = tf.nn.dropout(layer_2,0.5)
    layer_3 = tf.nn.relu(tf.add(tf.matmul(layer_2, weights['h3']), biases['b3']))
    layer_3= tf.nn.dropout(layer_3,0.5)
    # Output fully connected layer with a neuron for each class
    out_layer = tf.nn.softmax(tf.matmul(layer_3, weights['out']) + biases['out'])
    return out_layer

logits = multilayer_perceptron(X)
# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate,beta1=0.9,beta2=0.999,e
train_op = optimizer.minimize(loss_op)
# Initializing the variables
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([train_op, loss_op], feed_dict={X: batch_x,
                                                            Y: batch_y})
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
            if epoch % display_step == 0:
```

```
                print("Epoch:", '%04d' % (epoch+1), "cost={:.9f}".format(avg_cost))
        print("Optimization Finished!")

        # Test model
        pred = tf.nn.softmax(logits)  # Apply softmax to logits
        correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(Y, 1))
        # Calculate accuracy
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
        print("Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels}))
```

```
Epoch: 0001 cost=1.782518808
Epoch: 0002 cost=1.691063712
Epoch: 0003 cost=1.694938970
Epoch: 0004 cost=1.711379024
Epoch: 0005 cost=1.710448947
```

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

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-2-82dfb5148e6e> in <module>()
     56             # Run optimization op (backprop) and cost op (to get loss value)
     57             _, c = sess.run([train_op, loss_op], feed_dict={X: batch_x,
---> 58                                                             Y: batch_y})
     59             # Compute average loss
     60             avg_cost += c / total_batch


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
    893     try:
    894       result = self._run(None, fetches, feed_dict, options_ptr,
--> 895                          run_metadata_ptr)
    896       if run_metadata:
    897         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
   1122     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1123       results = self._do_run(handle, final_targets, final_fetches,
-> 1124                              feed_dict_tensor, options, run_metadata)
   1125     else:
   1126       results = []


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
   1319     if handle is None:
```

```
   1320       return self._do_call(_run_fn, self._session, feeds, fetches, targets,
-> 1321                            options, run_metadata)
   1322     else:
   1323       return self._do_call(_prun_fn, self._session, handle, feeds, fetches)


 /media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
   1325   def _do_call(self, fn, *args):
   1326     try:
-> 1327       return fn(*args)
   1328     except errors.OpError as e:
   1329       message = compat.as_text(e.message)


 /media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
   1304           return tf_session.TF_Run(session, options,
   1305                                    feed_dict, fetch_list, target_list,
-> 1306                                    status, run_metadata)
   1307
   1308     def _prun_fn(session, handle, feed_dict, fetch_list):


 KeyboardInterrupt:
```

4 (d)

```
In [28]: from tensorflow.examples.tutorials.mnist import input_data
         mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
         import tensorflow as tf
         # Training Parameters
         learning_rate = 0.001
         num_steps = 20000
         batch_size = 128
         display_step = 10

         # Network Parameters
         num_input = 784 # MNIST data input (img shape: 28*28)
         num_classes = 10 # MNIST total classes (0-9 digits)
         dropout = 0.75 # Dropout, probability to keep units

         # tf Graph input
         X = tf.placeholder(tf.float32, [None, num_input])
         Y = tf.placeholder(tf.float32, [None, num_classes])
         keep_prob = tf.placeholder(tf.float32) # dropout (keep probability)
```

```
# Create some wrappers for simplicity
def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)


def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1],
                          padding='SAME')


# Create model
def conv_net(x, weights, biases, dropout):
    # MNIST data input is a 1-D vector of 784 features (28*28 pixels)
    # Reshape to match picture format [Height x Width x Channel]
    # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
    x = tf.reshape(x, shape=[-1, 28, 28, 1])

    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)

    # Convolution Layer
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling)
    conv2 = maxpool2d(conv2, k=2)

    # Fully connected layer
    # Reshape conv2 output to fit fully connected layer input
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    # Apply Dropout
    fc1 = tf.nn.dropout(fc1, dropout)

    # Output, class prediction
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out

# Store layers weight & bias
weights = {
    # 5x5 conv, 1 input, 32 outputs
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
```

```
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    # fully connected, 7*7*64 inputs, 1024 outputs
    'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])),
    # 1024 inputs, 10 outputs (class prediction)
    'out': tf.Variable(tf.random_normal([1024, num_classes]))
}

biases = {
    'bc1': tf.Variable(tf.random_normal([32])),
    'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([num_classes]))
}

# Construct model
logits = conv_net(X, weights, biases, keep_prob)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)


# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    for step in range(1, num_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: 0.8})
        if step % display_step == 0 or step == 1:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                                 Y: batch_y,
                                                                 keep_prob: 1.0})
            print("Step " + str(step) + ", Minibatch Loss= " + \
```

```
                          "{:.4f}".format(loss) + ", Training Accuracy= " + \
                          "{:.3f}".format(acc))

            print("Optimization Finished!")

            # Calculate accuracy for 256 MNIST test images
            print("Testing Accuracy:", \
                sess.run(accuracy, feed_dict={X: mnist.test.images[:256],
                                              Y: mnist.test.labels[:256],
                                              keep_prob: 1.0}))
```

```
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
Step 1, Minibatch Loss= 68293.0000, Training Accuracy= 0.234
Step 10, Minibatch Loss= 19787.0898, Training Accuracy= 0.266
Step 20, Minibatch Loss= 7345.4521, Training Accuracy= 0.680
Step 30, Minibatch Loss= 8065.6045, Training Accuracy= 0.625
Step 40, Minibatch Loss= 5238.3516, Training Accuracy= 0.711
Step 50, Minibatch Loss= 4786.7939, Training Accuracy= 0.766
Step 60, Minibatch Loss= 2997.5984, Training Accuracy= 0.836
Step 70, Minibatch Loss= 4402.3105, Training Accuracy= 0.805
Step 80, Minibatch Loss= 2944.4343, Training Accuracy= 0.844
Step 90, Minibatch Loss= 1620.0334, Training Accuracy= 0.906
Step 100, Minibatch Loss= 3405.2231, Training Accuracy= 0.836
Step 110, Minibatch Loss= 1812.5856, Training Accuracy= 0.883
Step 120, Minibatch Loss= 2312.6060, Training Accuracy= 0.883
Step 130, Minibatch Loss= 1869.3074, Training Accuracy= 0.875
Step 140, Minibatch Loss= 360.8992, Training Accuracy= 0.977
Step 150, Minibatch Loss= 2032.6212, Training Accuracy= 0.906
Step 160, Minibatch Loss= 1598.8591, Training Accuracy= 0.914
Step 170, Minibatch Loss= 1715.2766, Training Accuracy= 0.922
Step 180, Minibatch Loss= 1923.0825, Training Accuracy= 0.891
Step 190, Minibatch Loss= 459.8496, Training Accuracy= 0.961
Step 200, Minibatch Loss= 991.2896, Training Accuracy= 0.969
Step 210, Minibatch Loss= 1138.0076, Training Accuracy= 0.945
Step 220, Minibatch Loss= 1456.3777, Training Accuracy= 0.883
Step 230, Minibatch Loss= 1011.3575, Training Accuracy= 0.945
Step 240, Minibatch Loss= 696.5654, Training Accuracy= 0.945
Step 250, Minibatch Loss= 677.1764, Training Accuracy= 0.953
Step 260, Minibatch Loss= 182.3857, Training Accuracy= 0.969
Step 270, Minibatch Loss= 1124.8761, Training Accuracy= 0.930
Step 280, Minibatch Loss= 581.8705, Training Accuracy= 0.938
Step 290, Minibatch Loss= 262.9534, Training Accuracy= 0.953
Step 300, Minibatch Loss= 1049.5366, Training Accuracy= 0.914
Step 310, Minibatch Loss= 612.7902, Training Accuracy= 0.945
Step 320, Minibatch Loss= 1071.6720, Training Accuracy= 0.922
```

```
Step 330, Minibatch Loss= 1398.4375, Training Accuracy= 0.906
Step 340, Minibatch Loss= 817.4532, Training Accuracy= 0.938
Step 350, Minibatch Loss= 712.0858, Training Accuracy= 0.945
Step 360, Minibatch Loss= 970.9204, Training Accuracy= 0.930
Step 370, Minibatch Loss= 693.0248, Training Accuracy= 0.938
Step 380, Minibatch Loss= 1514.5437, Training Accuracy= 0.883
Step 390, Minibatch Loss= 762.6429, Training Accuracy= 0.945
Step 400, Minibatch Loss= 597.1836, Training Accuracy= 0.930
Step 410, Minibatch Loss= 1498.8977, Training Accuracy= 0.938
Step 420, Minibatch Loss= 876.9764, Training Accuracy= 0.938
Step 430, Minibatch Loss= 786.6605, Training Accuracy= 0.961
Step 440, Minibatch Loss= 753.3494, Training Accuracy= 0.953
Step 450, Minibatch Loss= 935.2626, Training Accuracy= 0.930
Step 460, Minibatch Loss= 951.3262, Training Accuracy= 0.922
Step 470, Minibatch Loss= 990.8436, Training Accuracy= 0.945
Step 480, Minibatch Loss= 389.8413, Training Accuracy= 0.953
Step 490, Minibatch Loss= 807.3003, Training Accuracy= 0.945
```

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

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-28-080f3128e0e6> in <module>()
    108         batch_x, batch_y = mnist.train.next_batch(batch_size)
    109         # Run optimization op (backprop)
--> 110         sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: 0.8})
    111         if step % display_step == 0 or step == 1:
    112             # Calculate batch loss and accuracy


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
    893     try:
    894       result = self._run(None, fetches, feed_dict, options_ptr,
--> 895                          run_metadata_ptr)
    896       if run_metadata:
    897         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
   1122     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1123       results = self._do_run(handle, final_targets, final_fetches,
-> 1124                              feed_dict_tensor, options, run_metadata)
   1125     else:
   1126       results = []
```

```
/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
   1319     if handle is None:
   1320       return self._do_call(_run_fn, self._session, feeds, fetches, targets,
-> 1321                            options, run_metadata)
   1322     else:
   1323       return self._do_call(_prun_fn, self._session, handle, feeds, fetches)


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
   1325   def _do_call(self, fn, *args):
   1326     try:
-> 1327       return fn(*args)
   1328     except errors.OpError as e:
   1329       message = compat.as_text(e.message)


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
   1304           return tf_session.TF_Run(session, options,
   1305                                    feed_dict, fetch_list, target_list,
-> 1306                                    status, run_metadata)
   1307
   1308     def _prun_fn(session, handle, feed_dict, fetch_list):


KeyboardInterrupt:
```

4.0.1 Exercise H6.2: Long short-term memory (LSTM)

In [ ]:
```
import random
import numpy as np
import tensorflow as tf
from tensorflow.contrib import rnn
serieses = []
labels = []
for _ in range(10000):
    series = np.random.randint(0,9,30)
    serieses.append(series)
    if(sum(series)>=100):
        labels.append([0,1])
    else:
        labels.append([1,0])
train_input = serieses[0:8000]
test_input = serieses[8000:10000]
train_output = labels[0:8000]
test_output = labels[8000:10000]
```

In [17]:
```
import random
```

```
import numpy as np
import tensorflow as tf
from tensorflow.contrib import rnn
serieses = []
labels = []
for _ in range(10000):
    series = np.random.randint(0,9,30)
    serieses.append(series)
    if(sum(series)>=100):
        labels.append([0,1])
    else:
        labels.append([1,0])
train_input = serieses[0:8000]
test_input = serieses[8000:10000]
train_output = labels[0:8000]
test_output = labels[8000:10000]
# Training Parameters
learning_rate = 0.001
training_steps = 10000
batch_size = 100
display_step = 200

# Network Parameters
num_input = 1 # MNIST data input (img shape: 28*28)
timesteps = 30 # timesteps
num_hidden = 200 # hidden layer num of features
num_classes = 2 # MNIST total classes (0-9 digits)
X = tf.placeholder("float", [None, 30, 1])
Y = tf.placeholder("float", [None, 2])
weights = {
    'out': tf.Variable(tf.random_normal([num_hidden, num_classes]))
}
biases = {
    'out': tf.Variable(tf.random_normal([num_classes]))
}
def RNN(x, weights, biases):

    # Prepare data shape to match `rnn` function requirements
    # Current data input shape: (batch_size, timesteps, n_input)
    # Required shape: 'timesteps' tensors list of shape (batch_size, n_input)

    # Unstack to get a list of 'timesteps' tensors of shape (batch_size, n_input)
    x = tf.unstack(x, timesteps, 1)

    # Define a lstm cell with tensorflow
    lstm_cell = rnn.BasicLSTMCell(num_hidden, forget_bias=1.0)

    # Get lstm cell output
```

```
    outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)

    # Linear activation, using rnn inner loop last output
    return tf.matmul(outputs[-1], weights['out']) + biases['out']

logits = RNN(X, weights, biases)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)

# Evaluate model (with test logits, for dropout to be disabled)
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    for step in range(0, 80):
        batch_x, batch_y = np.array(train_input[step*100:(step+1)*100]),np.array(trai
        # Reshape data to get 28 seq of 28 elements
        batch_x = batch_x.reshape((batch_size, timesteps, num_input))
        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
        if step % display_step == 0 or step == 1:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                                 Y: batch_y})
            print("Step " + str(step) + ", Minibatch Loss= " + \
                  "{:.4f}".format(loss) + ", Training Accuracy= " + \
                  "{:.3f}".format(acc))

    print("Optimization Finished!")

    # Calculate accuracy for 128 mnist test images

    test_data = np.array(test_input)
    test_data = test_data.reshape((batch_size, timesteps, num_input))
    test_label = np.array(test_output)
```

```
        print("Testing Accuracy:", \
            sess.run(accuracy, feed_dict={X: test_data, Y: test_label}))
```

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

ValueError                                Traceback (most recent call last)

<ipython-input-17-ee072a0e572d> in <module>()
     53     return tf.matmul(outputs[-1], weights['out']) + biases['out']
     54
---> 55 logits = RNN(X, weights, biases)
     56 prediction = tf.nn.softmax(logits)
     57


<ipython-input-17-ee072a0e572d> in RNN(x, weights, biases)
     48
     49     # Get lstm cell output
---> 50     outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)
     51
     52     # Linear activation, using rnn inner loop last output


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
   1235             state_size=cell.state_size)
   1236       else:
-> 1237         (output, state) = call_cell()
   1238
   1239       outputs.append(output)


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
   1222         varscope.reuse_variables()
   1223       # pylint: disable=cell-var-from-loop
-> 1224       call_cell = lambda: cell(input_, state)
   1225       # pylint: enable=cell-var-from-loop
   1226       if sequence_length is not None:


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
    178       with vs.variable_scope(vs.get_variable_scope(),
    179                              custom_getter=self._rnn_get_variable):
--> 180         return super(RNNCell, self).__call__(inputs, state)
    181
    182   def _rnn_get_variable(self, getter, *args, **kwargs):
```

```
/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/l
448         # Check input assumptions set after layer building, e.g. input shape.
449         self._assert_input_compatibility(inputs)
--> 450         outputs = self.call(inputs, *args, **kwargs)
451
452         # Apply activity regularization.


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
399       c, h = array_ops.split(value=state, num_or_size_splits=2, axis=1)
400
--> 401     concat = _linear([inputs, h], 4 * self._num_units, True)
402
403     # i = input_gate, j = new_input, f = forget_gate, o = output_gate


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
1037         _WEIGHTS_VARIABLE_NAME, [total_arg_size, output_size],
1038         dtype=dtype,
-> 1039         initializer=kernel_initializer)
1040     if len(args) == 1:
1041       res = math_ops.matmul(args[0], weights)


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
1063       collections=collections, caching_device=caching_device,
1064       partitioner=partitioner, validate_shape=validate_shape,
-> 1065       use_resource=use_resource, custom_getter=custom_getter)
1066 get_variable_or_local_docstring = (
1067     """%s


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
960           collections=collections, caching_device=caching_device,
961           partitioner=partitioner, validate_shape=validate_shape,
--> 962           use_resource=use_resource, custom_getter=custom_getter)
963
964   def _get_partitioned_variable(self,


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/c
358           reuse=reuse, trainable=trainable, collections=collections,
359           caching_device=caching_device, partitioner=partitioner,
--> 360           validate_shape=validate_shape, use_resource=use_resource)
361     else:
362       return _true_getter(
```

```
/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
181
182   def _rnn_get_variable(self, getter, *args, **kwargs):
--> 183     variable = getter(*args, **kwargs)
184     trainable = (variable in tf_variables.trainable_variables() or
185                  (isinstance(variable, tf_variables.PartitionedVariable) and


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
350           trainable=trainable, collections=collections,
351           caching_device=caching_device, validate_shape=validate_shape,
--> 352           use_resource=use_resource)
353
354     if custom_getter is not None:


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/tensorflow/python/
662                           " Did you mean to set reuse=True in VarScope? "
663                           "Originally defined at:\n\n%s" % (
--> 664                               name, "".join(traceback.format_list(tb))))
665       found_var = self._vars[name]
666       if not shape.is_compatible_with(found_var.get_shape()):


ValueError: Variable rnn/basic_lstm_cell/kernel already exists, disallowed. Did you mea

  File "<ipython-input-15-0a0e3f362050>", line 50, in RNN
    outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)
  File "<ipython-input-15-0a0e3f362050>", line 55, in <module>
    logits = RNN(X, weights, biases)
  File "/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/IPython/core/i
    exec(code_obj, self.user_global_ns, self.user_ns)
```

In [10]:

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

ImportError                               Traceback (most recent call last)

<ipython-input-10-e959f633d1e8> in <module>()
  7 from keras.models import Sequential
  8 from keras.layers.core import Dense, Activation
----> 9 from keras.initializations import normal, identity, one
 10 from keras.layers.recurrent import SimpleRNN, LSTM
 11 from keras.optimizers import RMSprop
```

```
/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/keras/initializati
  2 import numpy as np
  3 from . import backend as K
----> 4 from .utils.generic_utils import get_from_module
  5
  6


ImportError: cannot import name 'get_from_module'
```

In [24]:
```
import random
import numpy as np
import tensorflow as tf
from tensorflow.contrib import rnn
serieses = []
labels = []
for _ in range(10000):
    series = np.random.randint(0,9,30)
    serieses.append(series)
    if(sum(series)>=100):
        labels.append([0,1])
    else:
        labels.append([1,0])
train_input = serieses[0:8000]
test_input = serieses[8000:10000]
train_output = labels[0:8000]
test_output = labels[8000:10000]

x_train=np.array(train_input)
y_train=np.array(train_output)

x_test=np.array(test_input)
y_test=np.array(test_output)
```

In [25]:
```
print(x_train.shape)
print(y_train.shape)
```

```
(8000, 30)
(8000, 2)
```

In [29]:
```
import keras
from keras.layers import LSTM
from keras.layers import Dense, Activation
from keras.datasets import mnist
from keras.models import Sequential
```

```
from keras.optimizers import Adam

learning_rate = 0.001
training_iters = 60
batch_size = 50
display_step = 10

n_input = 8000
n_step = 30
n_hidden = 128
n_classes = 2


#x_train = x_train.reshape(-1, n_step, n_input)
#x_test = x_test.reshape(-1, n_step, n_input)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')


y_train = keras.utils.to_categorical(y_train, n_classes)
y_test = keras.utils.to_categorical(y_test, n_classes)

model = Sequential()
model.add(LSTM(n_hidden,
               batch_input_shape=(None, n_step, n_input),
               unroll=True))

model.add(Dense(n_classes))
model.add(Activation('softmax'))

adam = Adam(lr=learning_rate)
model.summary()
model.compile(optimizer=adam,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=training_iters,
          verbose=1,
          validation_data=(x_test, y_test))

scores = model.evaluate(x_test, y_test, verbose=0)
print('LSTM test score:', scores[0])
print('LSTM test accuracy:', scores[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
lstm_2 (LSTM)                (None, 128)               4162048
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
_________________________________________________________________
activation_2 (Activation)    (None, 2)                 0
=================================================================
Total params: 4,162,306
Trainable params: 4,162,306
Non-trainable params: 0
_________________________________________________________________
```

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

ValueError                                Traceback (most recent call last)

<ipython-input-29-7875f662a4a5> in <module>()
     44           epochs=training_iters,
     45           verbose=1,
---> 46           validation_data=(x_test, y_test))
     47
     48 scores = model.evaluate(x_test, y_test, verbose=0)


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/keras/models.py in
    865             validation_data: this can be either
    866                 - a generator for the validation data
--> 867                 - a tuple (inputs, targets)
    868                 - a tuple (inputs, targets, sample_weights).
    869             nb_val_samples: only relevant if `validation_data` is a generator.


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/keras/engine/traini
   1520                     generator_output = None
   1521                     while enqueuer.is_running():
-> 1522                         if not enqueuer.queue.empty():
   1523                             generator_output = enqueuer.queue.get()
   1524                             break


/media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/keras/engine/traini
   1388         """Fits the model on data generated batch-by-batch by
   1389         a Python generator.
-> 1390         The generator is run in parallel to the model, for efficiency.
   1391         For instance, this allows you to do real-time data augmentation
   1392         on images on CPU in parallel to training your model on GPU.
```

```
    /media/zhanwang/data/data/lab/anaconda3/lib/python3.6/site-packages/keras/engine/traini
    239     if isinstance(metrics, list):
    240         # we then apply all metrics to all outputs.
--> 241         return [copy.copy(metrics) for _ in output_names]
    242     elif isinstance(metrics, dict):
    243         nested_metrics = []


    ValueError: Input arrays should have the same number of samples as target arrays. Found
```