Comp 543 Assignment 6

Kai-Po Lin (kl72)

Task 0:

【Result】

```
Step 9980 Loss 1.047609 Correct 46 out of 100
Step 9981 Loss 1.1242996 Correct 41 out of 100
Step 9982 Loss 1.0572566 Correct 51 out of 100
Step 9983 Loss 1.1576947 Correct 36 out of 100
Step 9984 Loss 1.2016197 Correct 37 out of 100
Step 9985 Loss 1.077896 Correct 34 out of 100
Step 9986 Loss 1.0788203 Correct 32 out of 100
Step 9987 Loss 1.00177 Correct 52 out of 100
Step 9988 Loss 1.2013291 Correct 29 out of 100
Step 9989 Loss 1.198174 Correct 38 out of 100
Step 9990 Loss 1.0577922 Correct 48 out of 100
Step 9991 Loss 0.9937323 Correct 43 out of 100
Step 9992 Loss 1.0356152 Correct 53 out of 100
Step 9993 Loss 1.0411843 Correct 41 out of 100
Step 9994 Loss 1.0772316 Correct 41 out of 100
Step 9995 Loss 1.1136212 Correct 27 out of 100
Step 9996 Loss 1.0990411 Correct 35 out of 100
Step 9997 Loss 1.0766239 Correct 39 out of 100
Step 9998 Loss 1.0183234 Correct 49 out of 100
Step 9999 Loss 1.1167006 Correct 29 out of 100
student27@b9560e7f4e23:~$
```

Task 1 & 2:

【Code Modify】

```python
import numpy as np
import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

# the number of iterations to train for
numTrainingIters = 10000

# the number of hidden neurons that hold the state of the RNN
hiddenUnits = 500

# the number of classes that we are learning over
numClasses = 3

# the number of data points in a batch
batchSize = 100
```

```python
# this generates a new batch of training data of size batchSize from the
# list of lines of text data. This version of generateData is useful for
# an RNN because the data set x is a NumPy array with dimensions
# [batchSize, 256, maxSeqLen]; it can be unstacked into a series of
# matrices containing one-hot character encodings for each data point
# using tf.unstack(inputX, axis=2)
# Task 2
def generateDataRNN(maxSeqLen, data):
    # randomly sample batchSize lines of text
    myInts = np.random.choice(len(data) - 1, batchSize, replace=False)

    # i is how many line there are in this text, which it the channels of the shape
    # stack all of the text into a matrix of one-hot characters
    x = np.stack(data[i][1] for i in myInts.flat)

    # and stack all of the labels into a vector of labels
    y = np.stack(np.array((data[i][0])) for i in myInts.flat)

    # return the pair
    return (x, y)
```

```python
# Task 1~4
def randomData(randomList, data, train, test):
    i = len(train)
    j = len(test)
    for idx in range(len(data)):
        if idx in randomList:
            test[j] = data[idx]
            j += 1
        else:
            train[i] = data[idx]
            i += 1

    return train, test


# create the data dictionary
maxSeqLen = 0
data = dict()
train = dict()
test = dict()

# load up the three data sets
(maxSeqLen, data) = addToData(maxSeqLen, data, "Holmes.txt", 0, 11000)
randomList = np.random.choice(len(data), 1000, replace=False)
train, test = randomData(randomList, data, train, test)
#print(f"Stage1: Train=>{len(train)}, Test=>{len(test)}")

data.clear()
(maxSeqLen, data) = addToData(maxSeqLen, data, "war.txt", 1, 11000)
randomList = np.random.choice(len(data), 1000, replace=False)
train, test = randomData(randomList, data, train, test)
#print(f"Stage2: Train=>{len(train)}, Test=>{len(test)}")

data.clear()
(maxSeqLen, data) = addToData(maxSeqLen, data, "william.txt", 2, 11000)
randomList = np.random.choice(len(data), 1000, replace=False)
train, test = randomData(randomList, data, train, test)
#print(f"Stage3: Train=>{len(train)}, Test=>{len(test)}")
```

```
data.clear()
(maxSeqLen, data) = addToData (maxSeqLen, data, "william.txt", 2, 10000)
randomList = np.random.choice(len(data), 1000, replace=False)
train, test = randomData(randomList, data, train, test)
#print(f"Stage3: Train=>{len(train)}, Test=>{len(test)}")


# pad each entry in the dictionary with empty characters as needed so
# that the sequences are all of the same length
train = pad(maxSeqLen, train)
test = pad(maxSeqLen, test)

# now we build the TensorFlow computation... there are two inputs,
# a batch of text lines and a batch of labels
# Task 0~2
inputX = tf.placeholder(tf.float32, [batchSize, 256, maxSeqLen])

# Task 3
#inputX = tf.placeholder(tf.float32, [batchSize, 256 * maxSeqLen])

# Task 4
#inputX = tf.placeholder(tf.float32, [batchSize, 256 * 10, maxSeqLen-9])
inputY = tf.placeholder(tf.int32, [batchSize])

# this is the inital state of the RNN, before processing any data
initialState = tf.placeholder(tf.float32, [batchSize, hiddenUnits])

# the weight matrix that maps the inputs and hidden state to a set of values
W = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + 256, hiddenUnits)), dtype=tf.float32)


# Task 2
W_time_wrap = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + hiddenUnits + 256, hiddenUnits)), dtype=tf.float32)
```

```
# Task 2

# Use a queue to store the output for 10 time ticks later in the future
queue = list()
currentState = initialState
tenStateBefore = initialState

# A single timeTick is a character with a shape (batchSize, 256)
for timeTick in sequenceOfLetters:
    if len(queue) >= 10:
        tenStateBefore = queue.pop(0)
    # Concatenate (based on column (axis=1)) the state with the input, then compute the next state
    inputPlusState = tf.concat([timeTick, currentState, tenStateBefore], 1)
    next_state = tf.tanh(tf.matmul(inputPlusState, W_time_wrap) + b)
    currentState = next_state
    queue.append(currentState)
```

```
    _finalState = np.zeros((batchSize, hiddenUnits))
    # and run the training iters
    for epoch in range(numTrainingIters):
        # get some data
        # Task 2
        x, y = generateDataRNN(maxSeqLen, train)
        # Task 3
        #x, y = generateDataFeedForward(maxSeqLen, train)
        # Task 4
        #x, y = generateTimeWrapping(maxSeqLen, train)

        # do the training epoch
        _currentState = np.zeros((batchSize, hiddenUnits))
        _totalLoss, _trainingAlg, _currentState, _predictions, _outputs = sess.run(
                [totalLoss, trainingAlg, currentState, predictions, outputs],
                feed_dict={
                    inputX:x,
                    inputY:y,
                    initialState:_currentState
                })

        if epoch == numTrainingIters - 1:
            _finalState = _currentState
        # just FYI, compute the number of correct predictions
        numCorrect = 0
        for i in range (len(y)):
           maxPos = -1
           maxVal = 0.0
           for j in range (numClasses):
               if maxVal < _predictions[i][j]:
                   maxVal = _predictions[i][j]
                   maxPos = j
           if maxPos == y[i]:
               numCorrect = numCorrect + 1
        # print out to the screen
        print("Step", epoch, "Loss", _totalLoss, "Correct", numCorrect, "out of", batchSize)
```

```
    # Testing
    numCorrect = 0
    avgTestLoss = 0.0

    predictIter = int(len(test) / batchSize)
    for epoch in range(predictIter):
        # Testing Data
        # Task 2
        testX, testY = generateDataRNN(maxSeqLen, test)
        # Task 3
        #testX, testY = generateDataFeedForward(maxSeqLen, test)
        # Task 4
        #testx, testy = generateTimeWrapping(maxSeqLen, test)

        # Do the testing
        _currentState = np.zeros((batchSize, hiddenUnits))
        _totalLoss, _trainingAlg, _currentState, _predictions, _outputs = sess.run(
                [totalLoss, trainingAlg, currentState, predictions, outputs],
                feed_dict={
                    inputX: testX,
                    inputY: testY,
                    initialState: _finalState
                })
        avgTestLoss += _totalLoss

        # Compute the number of correct predictions
        for i in range(len(testY)):
           maxPos = -1
           maxVal = 0.0
           for j in range(numClasses):
               if maxVal < _predictions[i][j]:
                   maxVal = _predictions[i][j]
                   maxPos = j
           if maxPos == testY[i]:
               numCorrect = numCorrect + 1
    print(f"\nLoss for 1000 randomly chosen documents is {avgTestLoss / predictIter}, number correct labels is {numCorrect} out of
    {len(test)}")
```

【Result】

```
Step 9980 Loss 0.5950718 Correct 79 out of 100
Step 9981 Loss 0.4725026 Correct 83 out of 100
Step 9982 Loss 0.39556444 Correct 85 out of 100
Step 9983 Loss 0.38884097 Correct 84 out of 100
Step 9984 Loss 0.33827138 Correct 88 out of 100
Step 9985 Loss 0.4769132 Correct 79 out of 100
Step 9986 Loss 0.35419738 Correct 83 out of 100
Step 9987 Loss 0.38959557 Correct 89 out of 100
Step 9988 Loss 0.4367162 Correct 82 out of 100
Step 9989 Loss 0.36219075 Correct 85 out of 100
Step 9990 Loss 0.28873205 Correct 90 out of 100
Step 9991 Loss 0.3219959 Correct 88 out of 100
Step 9992 Loss 0.23269168 Correct 90 out of 100
Step 9993 Loss 0.38816842 Correct 87 out of 100
Step 9994 Loss 0.2329402 Correct 93 out of 100
Step 9995 Loss 0.373708 Correct 83 out of 100
Step 9996 Loss 0.33101013 Correct 83 out of 100
Step 9997 Loss 0.22485091 Correct 94 out of 100
Step 9998 Loss 0.2639354 Correct 89 out of 100
Step 9999 Loss 0.42247745 Correct 81 out of 100

Loss for 3000 randomly chosen documents is 1.1542970657348632, number correct labels is 1353 out of 3000
```

Task 3:

【Code Modify】

```
# this also generates a new batch of training data, but it represents
# the data as a NumPy array with dimensions [batchSize, 256 * maxSeqLen]
# where for each data point, all characters have been appended.  Useful
# for feed-forward network training
# Task 3
def generateDataFeedForward(maxSeqLen, data):
    # randomly sample batchSize lines of text
    myInts = np.random.choice(len(data) - 1, batchSize, replace=False)

    # stack all of the text into a matrix of one-hot characters
    x = np.stack(data[i][1].flatten() for i in myInts.flat)

    # and stack all of the labels into a vector of labels
    y = np.stack(np.array((data[i][0])) for i in myInts.flat)

    # return the pair
    return (x, y)
```

```
# now we build the TensorFlow computation... there are two inputs,
# a batch of text lines and a batch of labels
# Task 0~2
#inputX = tf.placeholder(tf.float32, [batchSize, 256, maxSeqLen])

# Task 3
inputX = tf.placeholder(tf.float32, [batchSize, 256 * maxSeqLen])

# Task 4
#inputX = tf.placeholder(tf.float32, [batchSize, 256 * 10, maxSeqLen-9])
inputY = tf.placeholder(tf.int32, [batchSize])

# this is the inital state of the RNN, before processing any data
initialState = tf.placeholder(tf.float32, [batchSize, hiddenUnits])

# the weight matrix that maps the inputs and hidden state to a set of values
W = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + 256, hiddenUnits)), dtype=tf.float32)


# Task 2
W_time_wrap = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + hiddenUnits + 256, hiddenUnits)), dtype=tf.float32)

# Task 3
W_feed_forward = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + 256 * maxSeqLen, hiddenUnits)), dtype=tf.float32)

# Task 4 (8 CNN filters)
CNNfilter = tf.Variable(np.random.normal(0, 0.05, (256 * 10, 8)), dtype=tf.float32)
W_CNN = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + 8, hiddenUnits)), dtype=tf.float32)
```

```
# Task 3
inputPlusState = tf.concat([inputX, initialState], 1)
next_state = tf.tanh(tf.matmul(inputPlusState, W_feed_forward) + b)
currentState = next_state
```

```
    _finalState = np.zeros((batchSize, hiddenUnits))
    # and run the training iters
    for epoch in range(numTrainingIters):
        # get some data
        # Task 2
        #x, y = generateDataRNN(maxSeqLen, train)
        # Task 3
        x, y = generateDataFeedForward(maxSeqLen, train)
        # Task 4
        #x, y = generateTimeWrapping(maxSeqLen, train)

        # do the training epoch
        _currentState = np.zeros((batchSize, hiddenUnits))
        _totalLoss, _trainingAlg, _currentState, _predictions, _outputs = sess.run(
                [totalLoss, trainingAlg, currentState, predictions, outputs],
                feed_dict={
                    inputX:x,
                    inputY:y,
                    initialState:_currentState
                })
```

```
    predictIter = int(len(test) / batchSize)
    for epoch in range(predictIter):
        # Testing Data
        # Task 2
        #testX, testY = generateDataRNN(maxSeqLen, test)
        # Task 3
        testX, testY = generateDataFeedForward(maxSeqLen, test)
        # Task 4
        #testx, testy = generateTimeWrapping(maxSeqLen, test)

        # Do the testing
        _currentState = np.zeros((batchSize, hiddenUnits))
        _totalLoss, _trainingAlg, _currentState, _predictions, _outputs = sess.run(
                [totalLoss, trainingAlg, currentState, predictions, outputs],
                feed_dict={
                    inputX: testX,
                    inputY: testY,
                    initialState: _finalState
                })
        avgTestLoss += _totalLoss
```

【Result】

```
Step 9980 Loss 0.08145488 Correct 97 out of 100
Step 9981 Loss 0.09960567 Correct 97 out of 100
Step 9982 Loss 0.119836666 Correct 96 out of 100
Step 9983 Loss 0.068522386 Correct 98 out of 100
Step 9984 Loss 0.13706918 Correct 93 out of 100
Step 9985 Loss 0.1313623 Correct 95 out of 100
Step 9986 Loss 0.098620504 Correct 98 out of 100
Step 9987 Loss 0.1091626 Correct 95 out of 100
Step 9988 Loss 0.123449236 Correct 96 out of 100
Step 9989 Loss 0.10135153 Correct 95 out of 100
Step 9990 Loss 0.108045846 Correct 97 out of 100
Step 9991 Loss 0.08251231 Correct 97 out of 100
Step 9992 Loss 0.14272389 Correct 93 out of 100
Step 9993 Loss 0.113656305 Correct 94 out of 100
Step 9994 Loss 0.10185616 Correct 96 out of 100
Step 9995 Loss 0.08913295 Correct 97 out of 100
Step 9996 Loss 0.1108139 Correct 98 out of 100
Step 9997 Loss 0.17066139 Correct 94 out of 100
Step 9998 Loss 0.11493325 Correct 94 out of 100
Step 9999 Loss 0.10125294 Correct 96 out of 100

Loss for 3000 randomly chosen documents is 0.4034940481185913, number correct labels is 2615 out of 3000
```

Task 4:

【Code Modify】

```
153 # Task 4
154 def generateTimeWrapping(maxSeqLen, data):
155     # Randomly sample batchSize lines of text
156     myInts = np.random.choice(len(data) - 1, batchSize, replace=False)
157 
158     # Stack all of the text into a matrix of one-hot characters
159     X = np.stack(data[i][1] for i in myInts.flat)
160     x = np.zeros((maxSeqLen-9, 100, 2560))
161     for i in range(maxSeqLen-9):
162         tempX = X[:, :, i:i+10]
163         tempX = np.stack(tempX[j].flatten() for j in range(len(tempX)))
164 
165         x[i] = tempX
166     x = np.transpose(x, (1, 2, 0))
167 
168     # and stack all of the labels into a vector of labels
169     y = np.stack(np.array((data[i][0])) for i in myInts.flat)
170 
171     # return the pair
172     return (x, y)
```

```
222 # now we build the TensorFlow computation... there are two inputs,
223 # a batch of text lines and a batch of labels
224 # Task 0~2
225 #inputX = tf.placeholder(tf.float32, [batchSize, 256, maxSeqLen])
226
227 # Task 3
228 #inputX = tf.placeholder(tf.float32, [batchSize, 256 * maxSeqLen])
229
230 # Task 4
231 inputX = tf.placeholder(tf.float32, [batchSize, 256 * 10, maxSeqLen-9])
232 inputY = tf.placeholder(tf.int32, [batchSize])
233
234 # this is the inital state of the RNN, before processing any data
235 initialState = tf.placeholder(tf.float32, [batchSize, hiddenUnits])
236
237 # the weight matrix that maps the inputs and hidden state to a set of values
238 W = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + 256, hiddenUnits)), dtype=tf.float32)
239
240
241 # Task 2
242 W_time_wrap = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + hiddenUnits + 256, hiddenUnits)), dtype=tf.float32)
243
244 # Task 3
245 W_feed_forward = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + 256 * maxSeqLen, hiddenUnits)), dtype=tf.float32)
246
247 # Task 4 (8 CNN filters)
248 CNNfilter = tf.Variable(np.random.uniform(0.01, 1, (256 * 10, 8)), dtype=tf.float32)
249 W_CNN = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits + 8, hiddenUnits)), dtype=tf.float32)
250
251
252 # bias for the hidden values
253 b = tf.Variable(np.zeros((1, hiddenUnits)), dtype=tf.float32)
254
255 # weights and bias for the final classification
256 W2 = tf.Variable(np.random.normal(0, 0.05, (hiddenUnits, numClasses)),dtype=tf.float32)
257 b2 = tf.Variable(np.zeros((1, numClasses)), dtype=tf.float32)
258
259 # For RNN
260 # unpack the input sequences so that we have "a series of matrices",
261 # each of which has a one-hot encoding of the current character from
262 # every input sequence
263 # For Task 2 & 4
264 sequenceOfLetters = tf.unstack(inputX, axis=2)
```

```
# For Task 2 & 4
sequenceOfLetters = tf.unstack(inputX, axis=2)


'''
# Task 2
# Use a queue to store the output for 10 time ticks later in the future
queue = list()
currentState = initialState
tenStateBefore = initialState

# A single timeTick is a character with a shape (batchSize, 256)
for timeTick in sequenceOfLetters:
    if len(queue) >= 10:
        tenStateBefore = queue.pop(0)
    # Concatenate (based on column (axis=1)) the state with the input, then compute the next state
    inputPlusState = tf.concat([timeTick, currentState, tenStateBefore], 1)
    next_state = tf.tanh(tf.matmul(inputPlusState, W_time_wrap) + b)
    currentState = next_state
    queue.append(currentState)
'''

'''
# Task 3
inputPlusState = tf.concat([inputX, initialState], 1)
next_state = tf.tanh(tf.matmul(inputPlusState, W_feed_forward) + b)
currentState = next_state
'''

# Task 4
# now we implement the forward pass
currentState = initialState
for timeTick in sequenceOfLetters:
    eightVal = tf.matmul(timeTick, CNNfilter)
    inputPlusState = tf.concat([eightVal, currentState], 1)
    next_state = tf.tanh(tf.matmul(inputPlusState, W_CNN) + b)
    currentState = next_state
```

```
    _finalState = np.zeros((batchSize, hiddenUnits))
    # and run the training iters
    for epoch in range(numTrainingIters):
        # get some data
        # Task 2
        #x, y = generateDataRNN(maxSeqLen, train)
        # Task 3
        #x, y = generateDataFeedForward(maxSeqLen, train)
        # Task 4
        x, y = generateTimeWrapping(maxSeqLen, train)

        # do the training epoch
        _currentState = np.zeros((batchSize, hiddenUnits))
        _totalLoss, _trainingAlg, _currentState, _predictions, _outputs = sess.run(
                [totalLoss, trainingAlg, currentState, predictions, outputs],
                feed_dict={
                    inputX:x,
                    inputY:y,
                    initialState:_currentState
                })
```

```
    predictIter = int(len(test) / batchSize)
    for epoch in range(predictIter):
        # Testing Data
        # Task 2
        #testX, testY = generateDataRNN(maxSeqLen, test)
        # Task 3
        #testX, testY = generateDataFeedForward(maxSeqLen, test)
        # Task 4
        testX, testY = generateTimeWrapping(maxSeqLen, test)

        # Do the testing
        _currentState = np.zeros((batchSize, hiddenUnits))
        _totalLoss, _trainingAlg, _currentState, _predictions, _outputs = sess.run(
                [totalLoss, trainingAlg, currentState, predictions, outputs],
                feed_dict={
                    inputX: testX,
                    inputY: testY,
                    initialState: _finalState
                })
        avgTestLoss += _totalLoss
```

【Result】

```
Step 9980 Loss 0.8511348 Correct 57 out of 100
Step 9981 Loss 0.85659486 Correct 57 out of 100
Step 9982 Loss 0.843916 Correct 53 out of 100
Step 9983 Loss 0.9280407 Correct 54 out of 100
Step 9984 Loss 0.9476252 Correct 50 out of 100
Step 9985 Loss 0.9172188 Correct 50 out of 100
Step 9986 Loss 0.85106856 Correct 56 out of 100
Step 9987 Loss 0.85628974 Correct 58 out of 100
Step 9988 Loss 0.7843358 Correct 62 out of 100
Step 9989 Loss 0.86767423 Correct 54 out of 100
Step 9990 Loss 0.9800533 Correct 51 out of 100
Step 9991 Loss 0.9019081 Correct 54 out of 100
Step 9992 Loss 0.7746209 Correct 72 out of 100
Step 9993 Loss 0.8683303 Correct 59 out of 100
Step 9994 Loss 0.97564906 Correct 52 out of 100
Step 9995 Loss 0.9497303 Correct 58 out of 100
Step 9996 Loss 0.8149671 Correct 62 out of 100
Step 9997 Loss 0.90209424 Correct 61 out of 100
Step 9998 Loss 0.9849289 Correct 48 out of 100
Step 9999 Loss 0.97914624 Correct 52 out of 100

Loss for 3000 randomly chosen documents is 0.8881920595963796, number correct labels is 1672 out of 3000
```