Chapter_16_Modeling_Sequential_Data_Using_Recurrent_Neural_Networ

March 20, 2024

[1]:
```
import pyprind
import pandas as pd
from string import punctuation
import re
import numpy as np
df = pd.read_csv('movie_data.csv', encoding='utf-8')
```

[2]:
```
## Separate words and
## count each word's occurrence
from collections import Counter
counts = Counter()
pbar = pyprind.ProgBar(len(df['review']), title='Counting words occurrences')
for i,review in enumerate(df['review']):
    text = ''.join([c if c not in punctuation else ' '+c+' ' for c in review]).
 ↪lower()
    df.loc[i,'review'] = text
    pbar.update()
    counts.update(text.split())
## Create a mapping
## Map each unique word to an integer
word_counts = sorted(counts, key=counts.get, reverse=True)
print(word_counts[:5])
word_to_int = {word: ii for ii, word in enumerate(word_counts, 1)}
mapped_reviews = []
pbar = pyprind.ProgBar(len(df['review']), title='Map reviews to ints')
for review in df['review']:
    mapped_reviews.append([word_to_int[word] for word in review.split()])
    pbar.update()
```

```
Counting words occurrences
0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:04:03
Map reviews to ints
```

```
['the', '.', ',', 'and', 'a']
```

```
0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:02
```

[3]:
```
## Define same-length sequences
## if sequence length < 200: left-pad with zeros
## if sequence length > 200: use the last 200 elements
sequence_length = 200 ## (Known as T in our RNN formulas)
sequences = np.zeros((len(mapped_reviews), sequence_length),dtype=int)
for i, row in enumerate(mapped_reviews):
    review_arr = np.array(row)
    sequences[i, -len(row):] = review_arr[-sequence_length:]
```

[4]:
```
X_train = sequences[:25000,:]
y_train = df.loc[:25000, 'sentiment'].values
X_test = sequences[25000:,:]
y_test = df.loc[25000:, 'sentiment'].values
```

[5]:
```
#Define a function to generate mini-batches:
def create_batch_generator(x, y=None, batch_size=64):
    n_batches = len(x)//batch_size
    x = x[:n_batches*batch_size]
    if y is not None:
        y = y[:n_batches*batch_size]
    for ii in range(0, len(x), batch_size):
        if y is not None:
            yield x[ii:ii+batch_size], y[ii:ii+batch_size]
        else:
            yield x[ii:ii+batch_size]
```

[14]:
```
def build(self):
    ## Define the placeholders
    tf_x = tf.placeholder(tf.int32,shape=(self.batch_size, self.
 ↪seq_len),name='tf_x')
    tf_y = tf.placeholder(tf.float32,shape=(self.batch_size),name='tf_y')
    tf_keepprob = tf.placeholder(tf.float32,name='tf_keepprob')
    ## Create the embedding layer
    embedding = tf.Variable(tf.random_uniform((self.n_words, self.
 ↪embed_size),minval=-1, maxval=1),name='embedding')
    embed_x = tf.nn.embedding_lookup(embedding, tf_x,name='embeded_x')
    ## Define LSTM cell and stack them together
    cells = tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.DropoutWrapper(tf.
 ↪contrib.rnn.BasicLSTMCell(self.lstm_size),output_keep_prob=tf_keepprob) for␣
 ↪i in range(self.num_layers)])
    ## Define the initial state:
    self.initial_state = cells.zero_state(self.batch_size, tf.float32)
    print(' << initial state >> ', self.initial_state)
    lstm_outputs, self.final_state = tf.nn.dynamic_rnn(cells,␣
 ↪embed_x,initial_state=self.initial_state)
    ## [batch_size, max_time, cells.output_size]
    print('\n << lstm_output >> ', lstm_outputs)
```

```
    print('\n << final state >> ', self.final_state)
    logits = tf.layers.dense(inputs=lstm_outputs[:, -1],units=1,␣
 ↪activation=None,name='logits')
    logits = tf.squeeze(logits, name='logits_squeezed')
    print ('\n << logits >> ', logits)
    y_proba = tf.nn.sigmoid(logits, name='probabilities')
    predictions = {
    'probabilities': y_proba,
    'labels' : tf.cast(tf.round(y_proba), tf.int32,
    name='labels')
    }
    print('\n << predictions >> ', predictions)
    ## Define the cost function
    cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf_y,␣
 ↪logits=logits),name='cost')

    ## Define the optimizer
    optimizer = tf.train.AdamOptimizer(self.learning_rate)
    train_op = optimizer.minimize(cost, name='train_op')
```

[15]:
```
import tensorflow as tf
class SentimentRNN(object):
    def __init__(self, n_words, seq_len=200,lstm_size=256, num_layers=1,␣
 ↪batch_size=64,learning_rate=0.0001, embed_size=200):
        self.n_words = n_words
        self.seq_len = seq_len
        self.lstm_size = lstm_size ## number of hidden units
        self.num_layers = num_layers
        self.batch_size = batch_size
        self.learning_rate = learning_rate
        self.embed_size = embed_size
        self.g = tf.Graph()
        with self.g.as_default():
            tf.set_random_seed(123)
            self.build()
            self.saver = tf.train.Saver()
            self.init_op = tf.global_variables_initializer()
```

[11]:
```
```

[16]:
```
def train(self, X_train, y_train, num_epochs):
    with tf.Session(graph=self.g) as sess:
        sess.run(self.init_op)
        iteration = 1
        for epoch in range(num_epochs):
            state = sess.run(self.initial_state)
```

```
            for batch_x, batch_y in create_batch_generator(X_train, y_train,␣
 ↪self.batch_size):
                feed = {'tf_x:0': batch_x,'tf_y:0': batch_y,'tf_keepprob:0': 0.
 ↪5,self.initial_state : state}
                loss, _, state = sess.run(['cost:0', 'train_op',self.
 ↪final_state],feed_dict=feed)
                if iteration % 20 == 0:
                    print("Epoch: %d/%d Iteration: %d | Train loss: %.5f" %␣
 ↪(epoch + 1, num_epochs,iteration, loss))
                iteration +=1
            if (epoch+1)%10 == 0:
                self.saver.save(sess,"model/sentiment-%d.ckpt" % epoch)
```

[17]:
```
def predict(self, X_data, return_proba=False):
    preds = []
    with tf.Session(graph = self.g) as sess:
        self.saver.restore(sess, tf.train.latest_checkpoint('./model/'))
        test_state = sess.run(self.initial_state)
        for ii, batch_x in enumerate(create_batch_generator(X_data, None,␣
 ↪batch_size=self.batch_size), 1):
            feed = {'tf_x:0' : batch_x,'tf_keepprob:0' : 1.0,self.initial_state␣
 ↪: test_state}
        if return_proba:
            pred, test_state = sess.run(['probabilities:0', self.
 ↪final_state],feed_dict=feed)
        else:
            pred, test_state = sess.run(['labels:0', self.
 ↪final_state],feed_dict=feed)
        preds.append(pred)

    return np.concatenate(preds)
```

[18]:
```
n_words = max(list(word_to_int.values())) + 1
rnn =␣
 ↪SentimentRNN(n_words=n_words,seq_len=sequence_length,embed_size=256,lstm_size=128,num_layer
 ↪001)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-18-44839a8bfb28> in <module>
      1 n_words = max(list(word_to_int.values())) + 1
----> 2 rnn =␣
 ↪SentimentRNN(n_words=n_words,seq_len=sequence_length,embed_size=256,lstm_size=128,num_layer
 ↪001)

<ipython-input-15-2d0b76e3b5bc> in __init__(self, n_words, seq_len, lstm_size,␣
 ↪num_layers, batch_size, learning_rate, embed_size)
```

```
     12         with self.g.as_default():
     13             tf.set_random_seed(123)
---> 14             self.build()
     15             self.saver = tf.train.Saver()
     16             self.init_op = tf.global_variables_initializer()

AttributeError: 'SentimentRNN' object has no attribute 'build'
```

[19]:
```
rnn.train(X_train, y_train, num_epochs=40)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-87206b1a98ec> in <module>
----> 1 rnn.train(X_train, y_train, num_epochs=40)

NameError: name 'rnn' is not defined
```

[20]:
```
preds = rnn.predict(X_test)
y_true = y_test[:len(preds)]
print('Test Acc.: %.3f' % (np.sum(preds == y_true) / len(y_true)))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-0da02dfb0db6> in <module>
----> 1 preds = rnn.predict(X_test)
      2 y_true = y_test[:len(preds)]
      3 print('Test Acc.: %.3f' % (np.sum(preds == y_true) / len(y_true)))

NameError: name 'rnn' is not defined
```

[21]:
```
proba = rnn.predict(X_test, return_proba=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-fee6f23fbdc1> in <module>
----> 1 proba = rnn.predict(X_test, return_proba=True)

NameError: name 'rnn' is not defined
```

[23]:
```
import numpy as np
## Reading and processing text
with open('pg2265.txt', 'r', encoding='utf-8') as f:
    text=f.read()
```

```
text = text[15858:]
chars = set(text)
char2int = {ch:i for i,ch in enumerate(chars)}
int2char = dict(enumerate(chars))
text_ints = np.array([char2int[ch] for ch in text],dtype=np.int32)
```

[24]:
```
def reshape_data(sequence, batch_size, num_steps):
    tot_batch_length = batch_size * num_steps
    num_batches = int(len(sequence) / tot_batch_length)
    if num_batches*tot_batch_length + 1 > len(sequence):
        num_batches = num_batches - 1
    ## Truncate the sequence at the end to get rid of
    ## remaining charcaters that do not make a full batch
    x = sequence[0: num_batches*tot_batch_length]
    y = sequence[1: num_batches*tot_batch_length + 1]
    ## Split x & y into a list batches of sequences:
    x_batch_splits = np.split(x, batch_size)
    y_batch_splits = np.split(y, batch_size)
    ## Stack the batches together
    ## batch_size x tot_batch_length
    x = np.stack(x_batch_splits)
    y = np.stack(y_batch_splits)
    return x, y
```

[25]:
```
def create_batch_generator(data_x, data_y, num_steps):
    batch_size, tot_batch_length = data_x.shape
    num_batches = int(tot_batch_length/num_steps)
    for b in range(num_batches):
        yield (data_x[:, b*num_steps:(b+1)*num_steps],data_y[:, b*num_steps:
 ↪(b+1)*num_steps])
```

[26]:
```
import tensorflow as tf
import os
class CharRNN(object):
    def __init__(self, num_classes, batch_size=64,num_steps=100,␣
 ↪lstm_size=128,num_layers=1, learning_rate=0.001,keep_prob=0.5,␣
 ↪grad_clip=5,sampling=False):
        self.num_classes = num_classes
        self.batch_size = batch_size
        self.num_steps = num_steps
        self.lstm_size = lstm_size
        self.num_layers = num_layers
        self.learning_rate = learning_rate
        self.keep_prob = keep_prob
        self.grad_clip = grad_clip
        self.g = tf.Graph()
        with self.g.as_default():
```

```
            tf.set_random_seed(123)
            self.build(sampling=sampling)
            self.saver = tf.train.Saver()
            self.init_op = tf.global_variables_initializer()
```

[27]:
```
def build(self, sampling):
    if sampling == True:
        batch_size, num_steps = 1, 1
    else:
        batch_size = self.batch_size
        num_steps = self.num_steps
    tf_x = tf.placeholder(tf.int32,shape=[batch_size, num_steps],name='tf_x')
    tf_y = tf.placeholder(tf.int32,shape=[batch_size, num_steps],name='tf_y')
    tf_keepprob = tf.placeholder(tf.float32,name='tf_keepprob')
    # One-hot encoding:
    x_onehot = tf.one_hot(tf_x, depth=self.num_classes)
    y_onehot = tf.one_hot(tf_y, depth=self.num_classes)
    ### Build the multi-layer RNN cells
    cells = tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.DropoutWrapper(tf.
 ↪contrib.rnn.BasicLSTMCell(self.lstm_size),output_keep_prob=tf_keepprob) for␣
 ↪_ in range(self.num_layers)])
    ## Define the initial state
    self.initial_state = cells.zero_state(batch_size, tf.float32)
    ## Run each sequence step through the RNN
    lstm_outputs, self.final_state = tf.nn.dynamic_rnn(cells,␣
 ↪x_onehot,initial_state=self.initial_state)
    print(' << lstm_outputs >>', lstm_outputs)
    seq_output_reshaped = tf.reshape(lstm_outputs,shape=[-1, self.
 ↪lstm_size],name='seq_output_reshaped')
    logits = tf.layers.dense(inputs=seq_output_reshaped,units=self.
 ↪num_classes,activation=None,name='logits')
    proba = tf.nn.softmax(logits,name='probabilities')
    y_reshaped = tf.reshape(y_onehot,shape=[-1, self.
 ↪num_classes],name='y_reshaped')
    cost = tf.reduce_mean(tf.nn.
 ↪softmax_cross_entropy_with_logits(logits=logits,labels=y_reshaped),name='cost')
    # Gradient clipping to avoid "exploding gradients"
    tvars = tf.trainable_variables()
    grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars),self.grad_clip)
    optimizer = tf.train.AdamOptimizer(self.learning_rate)
    train_op = optimizer.apply_gradients(zip(grads, tvars),name='train_op')
```

[28]:
```
def train(self, train_x, train_y,num_epochs, ckpt_dir='./model/'):
    ## Create the checkpoint directory
    ## if it does not exists
    if not os.path.exists(ckpt_dir):
        os.mkdir(ckpt_dir)
```

```
    with tf.Session(graph=self.g) as sess:
        sess.run(self.init_op)
        n_batches = int(train_x.shape[1]/self.num_steps)
        iterations = n_batches * num_epochs
        for epoch in range(num_epochs):
            # Train network
            new_state = sess.run(self.initial_state)
            loss = 0
            ## Mini-batch generator:
            bgen = create_batch_generator(
            train_x, train_y, self.num_steps)
            for b, (batch_x, batch_y) in enumerate(bgen, 1):
                iteration = epoch*n_batches + b
                feed = {'tf_x:0': batch_x,'tf_y:0': batch_y,'tf_keepprob:0' :␣
↪self.keep_prob,self.initial_state : new_state}
                batch_cost, _, new_state = sess.run(['cost:0', 'train_op',self.
↪final_state],feed_dict=feed)
                if iteration % 10 == 0:
                    print('Epoch %d/%d Iteration %d | Training loss: %.4f' %␣
↪(epoch + 1, num_epochs,iteration, batch_cost))
            ## Save the trained model
            self.saver.save(sess, os.path.join(ckpt_dir, 'language_modeling.
↪ckpt'))
```

[29]:
```
def sample(self, output_length,ckpt_dir, starter_seq="The "):
    observed_seq = [ch for ch in starter_seq]
    with tf.Session(graph=self.g) as sess:
        self.saver.restore(sess,tf.train.latest_checkpoint(ckpt_dir))
        ## 1: run the model using the starter sequence
        new_state = sess.run(self.initial_state)
        for ch in starter_seq:
            x = np.zeros((1, 1))
            x[0, 0] = char2int[ch]
            feed = {'tf_x:0': x,'tf_keepprob:0': 1.0,self.initial_state:␣
↪new_state}
            proba, new_state = sess.run(['probabilities:0', self.
↪final_state],feed_dict=feed)
        ch_id = get_top_char(proba, len(chars))
        observed_seq.append(int2char[ch_id])

        ## 2: run the model using the updated observed_seq
        for i in range(output_length):
            x[0,0] = ch_id
            feed = {'tf_x:0': x,'tf_keepprob:0': 1.0,self.initial_state:␣
↪new_state}
            proba, new_state = sess.run(['probabilities:0', self.
↪final_state],feed_dict=feed)
```

```
            ch_id = get_top_char(proba, len(chars))
            observed_seq.append(int2char[ch_id])
    return ''.join(observed_seq)
```

[30]:
```
def get_top_char(probas, char_size, top_n=5):
    p = np.squeeze(probas)
    p[np.argsort(p)[:-top_n]] = 0.0
    p = p / np.sum(p)
    ch_id = np.random.choice(char_size, 1, p=p)[0]
    return ch_id
```

[31]:
```
batch_size = 64
num_steps = 100
train_x, train_y = reshape_data(text_ints,batch_size,num_steps)
rnn = CharRNN(num_classes=len(chars), batch_size=batch_size)
rnn.train(train_x, train_y,num_epochs=100,ckpt_dir='./model-100/')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-31-f3f793253187> in <module>
      2 num_steps = 100
      3 train_x, train_y = reshape_data(text_ints,batch_size,num_steps)
----> 4 rnn = CharRNN(num_classes=len(chars), batch_size=batch_size)
      5 rnn.train(train_x, train_y,num_epochs=100,ckpt_dir='./model-100/')

<ipython-input-26-dcbe8fb175f1> in __init__(self, num_classes, batch_size,␣
 ↪num_steps, lstm_size, num_layers, learning_rate, keep_prob, grad_clip,␣
 ↪sampling)
     14         with self.g.as_default():
     15             tf.set_random_seed(123)
---> 16             self.build(sampling=sampling)
     17             self.saver = tf.train.Saver()
     18             self.init_op = tf.global_variables_initializer()

AttributeError: 'CharRNN' object has no attribute 'build'
```

[32]:
```
del rnn
np.random.seed(123)
rnn = CharRNN(len(chars), sampling=True)
print(rnn.sample(ckpt_dir='./model-100/',output_length=500))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-32-536777f59bf1> in <module>
----> 1 del rnn
      2 np.random.seed(123)
```

```
      3 rnn = CharRNN(len(chars), sampling=True)
      4 print(rnn.sample(ckpt_dir='./model-100/',output_length=500))

NameError: name 'rnn' is not defined
```

[ ]: