poem_generation_with_RNN-exercise-ch06

November 22, 2025

1

2

[1]:
```
import numpy as np
import tensorflow as tf
import collections
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import layers, optimizers, datasets

start_token = 'bos'
end_token = 'eos'

def process_dataset(fileName):
    examples = []
    with open(fileName, 'r', encoding='utf-8') as fd:
        for line in fd:
            outs = line.strip().split(':')
            content = ''.join(outs[1:])
            ins = [start_token] + list(content) + [end_token]
            if len(ins) > 200:
                continue
            examples.append(ins)

    counter = collections.Counter()
    for e in examples:
        for w in e:
            counter[w]+=1

    sorted_counter = sorted(counter.items(), key=lambda x: -x[1])  #
    words, _ = zip(*sorted_counter)
    words = ('PAD', 'UNK') + words[:len(words)]
    word2id = dict(zip(words, range(len(words))))
    id2word = {word2id[k]:k for k in word2id}

    indexed_examples = [[word2id[w] for w in poem]
```

```
                        for poem in examples]
    seqlen = [len(e) for e in indexed_examples]

    instances = list(zip(indexed_examples, seqlen))

    return instances, word2id, id2word

def poem_dataset():
    instances, word2id, id2word = process_dataset(r'D:
 ↪\code\python\deeplearning\homework6\poems.txt')
    ds = tf.data.Dataset.from_generator(lambda: [ins for ins in instances],
                                        (tf.int64, tf.int64),
                                        (tf.TensorShape([None]),tf.
 ↪TensorShape([])))
    ds = ds.shuffle(buffer_size=10240)
    ds = ds.padded_batch(100, padded_shapes=(tf.TensorShape([None]),tf.
 ↪TensorShape([])))
    ds = ds.map(lambda x, seqlen: (x[:, :-1], x[:, 1:], seqlen-1))
    return ds, word2id, id2word
```

3

[2]:
```
class myRNNModel(keras.Model):
    def __init__(self, w2id):
        super(myRNNModel, self).__init__()
        self.v_sz = len(w2id)
        self.embed_layer = tf.keras.layers.Embedding(self.v_sz, 64)

        self.rnncell = tf.keras.layers.SimpleRNNCell(128)
        self.rnn_layer = tf.keras.layers.RNN(self.rnncell,␣
 ↪return_sequences=True)
        self.dense = tf.keras.layers.Dense(self.v_sz)

    def call(self, inp_ids, training=None):
        '''
               ,    ,  (batch_size, seq_len),  (batch_size, seq_len, self.v_sz)
        '''
        # 1.    token ids  embedding
        # inp_ids shape: (batch_size, seq_len)
        inp_emb = self.embed_layer(inp_ids)  # shape: (batch_size, seq_len, 64)

        # 2.  RNN
        rnn_out = self.rnn_layer(inp_emb, training=training)  # shape:␣
 ↪(batch_size, seq_len, 128)

        # 3.        logits
```

```
        logits = self.dense(rnn_out)  # shape: (batch_size, seq_len, v_sz)

        return logits

    def get_next_token(self, x, state):
        '''
        shape(x) = [b_sz,]
        '''

        inp_emb = self.embed_layer(x) #shape(b_sz, emb_sz)
        h, state = self.rnncell.call(inp_emb, state) # shape(b_sz, h_sz)
        logits = self.dense(h) # shape(b_sz, v_sz)
        out = tf.argmax(logits, axis=-1)
        return out, state
```

3.1 sequence loss

[3]:
```
def mkMask(input_tensor, maxLen):
    shape_of_input = tf.shape(input_tensor)
    shape_of_output = tf.concat(axis=0, values=[shape_of_input, [maxLen]])

    oneDtensor = tf.reshape(input_tensor, shape=(-1,))
    flat_mask = tf.sequence_mask(oneDtensor, maxlen=maxLen)
    return tf.reshape(flat_mask, shape_of_output)


def reduce_avg(reduce_target, lengths, dim):
    """
    Args:
        reduce_target : shape(d_0, d_1,..,d_dim, .., d_k)
        lengths : shape(d0, .., d_(dim-1))
        dim : which dimension to average, should be a python number
    """
    shape_of_lengths = lengths.get_shape()
    shape_of_target = reduce_target.get_shape()
    if len(shape_of_lengths) != dim:
        raise ValueError(('Second input tensor should be rank %d, ' +
                          'while it got rank %d') % (dim, len(shape_of_lengths)))
    if len(shape_of_target) < dim+1 :
        raise ValueError(('First input tensor should be at least rank %d, ' +
                          'while it got rank %d') % (dim+1,␣
 ↪len(shape_of_target)))

    rank_diff = len(shape_of_target) - len(shape_of_lengths) - 1
    mxlen = tf.shape(reduce_target)[dim]
    mask = mkMask(lengths, mxlen)
    if rank_diff!=0:
```

```
        len_shape = tf.concat(axis=0, values=[tf.shape(lengths), [1]*rank_diff])
        mask_shape = tf.concat(axis=0, values=[tf.shape(mask), [1]*rank_diff])
    else:
        len_shape = tf.shape(lengths)
        mask_shape = tf.shape(mask)
    lengths_reshape = tf.reshape(lengths, shape=len_shape)
    mask = tf.reshape(mask, shape=mask_shape)

    mask_target = reduce_target * tf.cast(mask, dtype=reduce_target.dtype)

    red_sum = tf.reduce_sum(mask_target, axis=[dim], keepdims=False)
    red_avg = red_sum / (tf.cast(lengths_reshape, dtype=tf.float32) + 1e-30)
    return red_avg
```

4 loss

[4]:
```
@tf.function
def compute_loss(logits, labels, seqlen):
    losses = tf.nn.sparse_softmax_cross_entropy_with_logits(
            logits=logits, labels=labels)
    losses = reduce_avg(losses, seqlen, dim=1)
    return tf.reduce_mean(losses)

def train_one_step(model, optimizer, x, y, seqlen):
    '''

        ,
    '''
    #  GradientTape
    with tf.GradientTape() as tape:
        #   ,   ,  training=True
        logits = model(x, training=True)
        #
        loss = compute_loss(logits, y, seqlen)

    #
    gradients = tape.gradient(loss, model.trainable_variables)
    #
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    return loss

def train(epoch, model, optimizer, ds):
    loss = 0.0
    accuracy = 0.0
    for step, (x, y, seqlen) in enumerate(ds):
        loss = train_one_step(model, optimizer, x, y, seqlen)
```

```
        if step % 50 == 0:
            print('epoch', epoch, ': loss', loss.numpy())

    return loss
```

5

[5]:
```
optimizer = optimizers.Adam(0.0005)
train_ds, word2id, id2word = poem_dataset()
model = myRNNModel(word2id)

for epoch in range(10):
    loss = train(epoch, model, optimizer, train_ds)
```

```
epoch 0 : loss 8.820581
WARNING:tensorflow:5 out of the last 18 calls to <function compute_loss at
0x0000024BC5E07100> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:5 out of the last 18 calls to <function compute_loss at
0x0000024BC5E07100> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
epoch 0 : loss 6.8395023
epoch 0 : loss 6.8395023
epoch 0 : loss 6.575797
epoch 0 : loss 6.575797
epoch 0 : loss 6.5541577
epoch 0 : loss 6.5541577
epoch 0 : loss 6.534297
epoch 0 : loss 6.534297
epoch 0 : loss 6.589379
epoch 0 : loss 6.589379
epoch 0 : loss 6.4870653
epoch 0 : loss 6.4870653
epoch 0 : loss 6.4831996
```

```
epoch 0 : loss 6.4831996
epoch 0 : loss 6.5280824
epoch 0 : loss 6.5280824
epoch 1 : loss 6.612505
epoch 1 : loss 6.612505
epoch 1 : loss 6.468927
epoch 1 : loss 6.468927
epoch 1 : loss 6.4845104
epoch 1 : loss 6.4845104
epoch 1 : loss 6.293411
epoch 1 : loss 6.293411
epoch 1 : loss 6.2031837
epoch 1 : loss 6.2031837
epoch 1 : loss 6.222295
epoch 1 : loss 6.222295
epoch 1 : loss 6.118825
epoch 1 : loss 6.118825
epoch 1 : loss 6.085
epoch 1 : loss 6.085
epoch 1 : loss 6.012605
epoch 1 : loss 6.012605
epoch 2 : loss 6.067061
epoch 2 : loss 6.067061
epoch 2 : loss 5.918438
epoch 2 : loss 5.918438
epoch 2 : loss 5.9133897
epoch 2 : loss 5.9133897
epoch 2 : loss 5.8428264
epoch 2 : loss 5.8428264
epoch 2 : loss 5.759844
epoch 2 : loss 5.759844
epoch 2 : loss 5.9239407
epoch 2 : loss 5.9239407
epoch 2 : loss 5.8833313
epoch 2 : loss 5.8833313
epoch 2 : loss 5.8582377
epoch 2 : loss 5.8582377
epoch 2 : loss 5.9285808
epoch 2 : loss 5.9285808
epoch 3 : loss 5.8407655
epoch 3 : loss 5.8407655
epoch 3 : loss 5.701578
epoch 3 : loss 5.701578
epoch 3 : loss 5.746084
epoch 3 : loss 5.746084
epoch 3 : loss 5.6953673
epoch 3 : loss 5.6953673
epoch 3 : loss 5.620606
```

```
epoch 3 : loss 5.620606
epoch 3 : loss 5.6671996
epoch 3 : loss 5.6671996
epoch 3 : loss 5.7081456
epoch 3 : loss 5.7081456
epoch 3 : loss 5.686957
epoch 3 : loss 5.686957
epoch 3 : loss 5.517827
epoch 3 : loss 5.517827
epoch 4 : loss 5.6896577
epoch 4 : loss 5.6896577
epoch 4 : loss 5.4840026
epoch 4 : loss 5.4840026
epoch 4 : loss 5.511974
epoch 4 : loss 5.511974
epoch 4 : loss 5.526822
epoch 4 : loss 5.526822
epoch 4 : loss 5.4263744
epoch 4 : loss 5.4263744
epoch 4 : loss 5.4414907
epoch 4 : loss 5.4414907
epoch 4 : loss 5.4572434
epoch 4 : loss 5.4572434
epoch 4 : loss 5.4702063
epoch 4 : loss 5.4702063
epoch 4 : loss 5.458606
epoch 4 : loss 5.458606
epoch 5 : loss 5.5823946
epoch 5 : loss 5.5823946
epoch 5 : loss 5.4075136
epoch 5 : loss 5.4075136
epoch 5 : loss 5.4475355
epoch 5 : loss 5.4475355
epoch 5 : loss 5.4820204
epoch 5 : loss 5.4820204
epoch 5 : loss 5.3538594
epoch 5 : loss 5.3538594
epoch 5 : loss 5.358069
epoch 5 : loss 5.358069
epoch 5 : loss 5.391399
epoch 5 : loss 5.391399
epoch 5 : loss 5.3642855
epoch 5 : loss 5.3642855
epoch 5 : loss 5.347478
epoch 5 : loss 5.347478
epoch 6 : loss 5.4163136
epoch 6 : loss 5.4163136
epoch 6 : loss 5.3750873
```

```
epoch 6 : loss 5.3750873
epoch 6 : loss 5.284243
epoch 6 : loss 5.284243
epoch 6 : loss 5.2260547
epoch 6 : loss 5.2260547
epoch 6 : loss 5.2876134
epoch 6 : loss 5.2876134
epoch 6 : loss 5.2748933
epoch 6 : loss 5.2748933
epoch 6 : loss 5.2770824
epoch 6 : loss 5.2770824
epoch 6 : loss 5.2756104
epoch 6 : loss 5.2756104
epoch 6 : loss 5.3531065
epoch 6 : loss 5.3531065
epoch 7 : loss 5.3173633
epoch 7 : loss 5.3173633
epoch 7 : loss 5.224677
epoch 7 : loss 5.224677
epoch 7 : loss 5.1907587
epoch 7 : loss 5.1907587
epoch 7 : loss 5.1513023
epoch 7 : loss 5.1513023
epoch 7 : loss 5.154849
epoch 7 : loss 5.154849
epoch 7 : loss 5.183319
epoch 7 : loss 5.183319
epoch 7 : loss 5.2058873
epoch 7 : loss 5.2058873
epoch 7 : loss 5.160111
epoch 7 : loss 5.160111
epoch 7 : loss 5.2279787
epoch 7 : loss 5.2279787
epoch 8 : loss 5.3067174
epoch 8 : loss 5.3067174
epoch 8 : loss 5.115646
epoch 8 : loss 5.115646
epoch 8 : loss 5.097439
epoch 8 : loss 5.097439
epoch 8 : loss 5.178859
epoch 8 : loss 5.178859
epoch 8 : loss 5.1519194
epoch 8 : loss 5.1519194
epoch 8 : loss 5.098657
epoch 8 : loss 5.098657
epoch 8 : loss 5.1361647
epoch 8 : loss 5.1361647
epoch 8 : loss 5.129225
```

```
epoch 8 : loss 5.129225
epoch 8 : loss 5.1602244
epoch 8 : loss 5.1602244
epoch 9 : loss 5.206551
epoch 9 : loss 5.206551
epoch 9 : loss 5.039499
epoch 9 : loss 5.039499
epoch 9 : loss 5.1726227
epoch 9 : loss 5.1726227
epoch 9 : loss 5.08385
epoch 9 : loss 5.08385
epoch 9 : loss 5.14501
epoch 9 : loss 5.14501
epoch 9 : loss 5.1043973
epoch 9 : loss 5.1043973
epoch 9 : loss 5.052031
epoch 9 : loss 5.052031
epoch 9 : loss 5.150922
epoch 9 : loss 5.150922
epoch 9 : loss 5.111984
epoch 9 : loss 5.111984
```

6

[6]:
```python
def gen_sentence():
    # SimpleRNNCell     ,
    state = tf.random.normal(shape=(1, 128), stddev=0.5)
    cur_token = tf.constant([word2id['bos']], dtype=tf.int32)
    collect = [] #
    '''
        for ,  model.get_next_token ,   50
    '''
    #   50
    for i in range(50):
        #    get_next_token ,   token
        cur_token, state = model.get_next_token(cur_token, state)
        #   token id
        collect.append(cur_token.numpy()[0])

    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

```
      eos      eos      eos      eos
```

[ ]:
```python

```