RNN Sentiment Analysis

February 7, 2021

[1]:
```
import os
import tensorflow as tf
import numpy as np
from tensorflow import keras
```

[2]:
```
tf.random.set_seed(22)
np.random.seed(22)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
assert tf.__version__.startswith('2.')
```

Fix random seed and reproducibility

[3]:
```
np.random.seed(7)
```

Load dataset but only keep the top n words, zero the rest

[4]:
```
top_words = 10000
```

Truncate and pad input sequences

[5]:
```
max_review_length = 80
(X_train, y_train), (X_test, y_test) = keras.datasets.imdb.load_data(num_words␣
 ↪= top_words)

# X_train = tf.convert_to_tensor(X_train)
# y_train = tf.one_hoy(y_train, depth = 2)
```

[6]:
```
x_train = keras.preprocessing.sequence.pad_sequences(X_train, maxlen =␣
 ↪max_review_length)
x_test = keras.preprocessing.sequence.pad_sequences(X_test, maxlen =␣
 ↪max_review_length)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

```
x_train shape: (25000, 80)
x_test shape: (25000, 80)
```

[7]:
```
class RNN(keras.Model):
    def __init__(self, units, num_classes, num_layers):
        super(RNN, self).__init__()

        # self.ceels = [keras.layers.LSTMCell(units) for _ in range(num_layers)]

        # self.rnn = keras.layers.RNN(self.cells, unroll = True)

        self.rnn = keras.layers.LSTM(units,return_sequences = True)
        self.rnn2 = keras.layers.LSTM(units)

        # self.cells = (keras.layers.LSTMCell(units) for _ in range(num_layers))

        # self.rnn = keras.layers.RNN(self.cells, return_sequences=True,␣
↪return_state=True)
        # self.rnn = keras.layers.LSTM(units, unroll=True)
        # self.rnn = keras.layers.StackedRNNCells(self.cells)

        # have 1000 words totally, every word will be embedding into 100 length␣
↪vector
        # the max sentence lenght is 80 words

        self.embedding = keras.layers.Embedding(top_words, 100,␣
↪input_length=max_review_length)
        self.fc = keras.layers.Dense(1)

    def call(self, inputs, training=None, mask=None):

        # print('x', inputs.shape)
        # [b, sentence len] => [b, sentence len, word embedding]

        x = self.embedding(inputs)
        # print('embedding', x.shape)

        x = self.rnn(x)
        x = self.rnn2(x)
        # print('rnn', x.shape)

        x = self.fc(x)
        print(x.shape)

        return x
```

[8]:
```
def main():

    units = 64
    num_classes = 2
```

```
    batch_size = 32
    epochs = 20

    model = RNN(units, num_classes, num_layers=2)


    model.compile(optimizer=keras.optimizers.Adam(0.001),
                  loss=keras.losses.BinaryCrossentropy(from_logits=True),
                  metrics=['accuracy'])

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

    # evaluate on test set
    scores = model.evaluate(x_test, y_test, batch_size, verbose=1)
    print("Final test loss and accuracy :", scores)
```

[9]:
```
if __name__ == '__main__':
    main()
```

```
Epoch 1/20
(None, 1)
(None, 1)
782/782 [==============================] - ETA: 0s - loss: 0.4217 - accuracy:
0.7921(None, 1)
782/782 [==============================] - 52s 67ms/step - loss: 0.4217 -
accuracy: 0.7921 - val_loss: 0.3665 - val_accuracy: 0.8286
Epoch 2/20
782/782 [==============================] - 64s 81ms/step - loss: 0.2673 -
accuracy: 0.8857 - val_loss: 0.3763 - val_accuracy: 0.8424
Epoch 3/20
782/782 [==============================] - 54s 69ms/step - loss: 0.1887 -
accuracy: 0.9235 - val_loss: 0.4449 - val_accuracy: 0.8334
Epoch 4/20
782/782 [==============================] - 54s 69ms/step - loss: 0.1296 -
accuracy: 0.9508 - val_loss: 0.4835 - val_accuracy: 0.8280
Epoch 5/20
782/782 [==============================] - 53s 67ms/step - loss: 0.0859 -
accuracy: 0.9678 - val_loss: 0.5468 - val_accuracy: 0.8268
Epoch 6/20
782/782 [==============================] - 52s 67ms/step - loss: 0.0551 -
accuracy: 0.9810 - val_loss: 0.7304 - val_accuracy: 0.8168
Epoch 7/20
782/782 [==============================] - 52s 66ms/step - loss: 0.0515 -
accuracy: 0.9820 - val_loss: 0.7853 - val_accuracy: 0.8173
Epoch 8/20
```

```
782/782 [==============================] - 52s 67ms/step - loss: 0.0310 -
accuracy: 0.9895 - val_loss: 0.8115 - val_accuracy: 0.8206
Epoch 9/20
782/782 [==============================] - 56s 71ms/step - loss: 0.0256 -
accuracy: 0.9914 - val_loss: 0.8566 - val_accuracy: 0.8260
Epoch 10/20
782/782 [==============================] - 52s 66ms/step - loss: 0.0263 -
accuracy: 0.9921 - val_loss: 0.8208 - val_accuracy: 0.8172
Epoch 11/20
782/782 [==============================] - 52s 67ms/step - loss: 0.0184 -
accuracy: 0.9941 - val_loss: 1.0357 - val_accuracy: 0.8124
Epoch 12/20
782/782 [==============================] - 53s 67ms/step - loss: 0.0209 -
accuracy: 0.9934 - val_loss: 0.8785 - val_accuracy: 0.8042
Epoch 13/20
782/782 [==============================] - 55s 70ms/step - loss: 0.0144 -
accuracy: 0.9953 - val_loss: 1.0250 - val_accuracy: 0.8152
Epoch 14/20
782/782 [==============================] - 54s 69ms/step - loss: 0.0183 -
accuracy: 0.9936 - val_loss: 0.8989 - val_accuracy: 0.8136
Epoch 15/20
782/782 [==============================] - 58s 74ms/step - loss: 0.0103 -
accuracy: 0.9968 - val_loss: 1.0755 - val_accuracy: 0.8245
Epoch 16/20
782/782 [==============================] - 56s 72ms/step - loss: 0.0103 -
accuracy: 0.9964 - val_loss: 1.0958 - val_accuracy: 0.8187
Epoch 17/20
782/782 [==============================] - 56s 71ms/step - loss: 0.0129 -
accuracy: 0.9957 - val_loss: 1.0758 - val_accuracy: 0.8263
Epoch 18/20
782/782 [==============================] - 54s 70ms/step - loss: 0.0036 -
accuracy: 0.9992 - val_loss: 1.1624 - val_accuracy: 0.8244
Epoch 19/20
782/782 [==============================] - 56s 71ms/step - loss: 0.0073 -
accuracy: 0.9976 - val_loss: 1.0436 - val_accuracy: 0.8139
Epoch 20/20
782/782 [==============================] - 57s 73ms/step - loss: 0.0119 -
accuracy: 0.9956 - val_loss: 1.1614 - val_accuracy: 0.8180
782/782 [==============================] - 10s 13ms/step - loss: 1.1614 -
accuracy: 0.8180
Final test loss and accuracy : [1.1613751649856567, 0.8179600238800049]
```