使用 RNN 进行文本分类

原创 zzzkkkk 要努力的小正 2020-11-16

此文本分类教程将在 IMDB 大型电影评论数据集上训练循环神经网络，以进行情感分析。

设置

```
import tensorflow_datasets as tfds
import tensorflow as tf
```

导入 `matplotlib` 并创建一个辅助函数来绘制计算图：

```
import matplotlib.pyplot as plt

def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
  plt.show()

```

设置输入流水线

IMDB 大型电影评论数据集是一个*二进制分类*数据集——所有评论都具有*正面*或*负面*情绪。

使用 TFDS 下载数据集。

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

```
WARNING:absl:TFDS datasets with text encoding are deprecated and will be removed in a fu

Downloading and preparing dataset imdb_reviews/subwords8k/1.0.0 (download: 80.23 MiB, ge
```

```
Shuffling and writing examples to /home/kbuilder/tensorflow_datasets/imdb_reviews/subwor
Shuffling and writing examples to /home/kbuilder/tensorflow_datasets/imdb_reviews/subwor
Shuffling and writing examples to /home/kbuilder/tensorflow_datasets/imdb_reviews/subwor
Dataset imdb_reviews downloaded and prepared to /home/kbuilder/tensorflow_datasets/imdb_
```

数据集 `info` 包括编码器（`tfds.features.text.SubwordTextEncoder`）。

```
encoder = info.features['text'].encoder
print('Vocabulary size: {}'.format(encoder.vocab_size))
```

```
Vocabulary size: 8185
```

此文本编码器将以可逆方式对任何字符串进行编码，并在必要时退回到字节编码。

```
sample_string = 'Hello TensorFlow.'

encoded_string = encoder.encode(sample_string)
print('Encoded string is {}'.format(encoded_string))

original_string = encoder.decode(encoded_string)
print('The original string: "{}"'.format(original_string))
```

```
Encoded string is [4025, 222, 6307, 2327, 4043, 2120, 7975]
The original string: "Hello TensorFlow."
```

```
assert original_string == sample_string
```

```
for index in encoded_string:
  print('{} ----&gt; {}'.format(index, encoder.decode([index])))
```

```
4025 ----&gt; Hell
222 ----&gt; o
6307 ----&gt; Ten
2327 ----&gt; sor
4043 ----&gt; Fl
```

```
2120 ----&gt; ow
7975 ----&gt; .
```

准备用于训练的数据

接下来，创建这些编码字符串的批次。使用 `padded_batch` 方法将序列零填充至批次中最长字符串的长度：

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
```

```
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.padded_batch(BATCH_SIZE)

test_dataset = test_dataset.padded_batch(BATCH_SIZE)
```

创建模型

构建一个 `tf.keras.Sequential` 模型并从嵌入向量层开始。嵌入向量层每个单词存储一个向量。调用时，它会将单词索引序列转换为向量序列。这些向量是可训练的。（在足够的数据上）训练后，具有相似含义的单词通常具有相似的向量。

与通过 `tf.keras.layers.Dense` 层传递独热编码向量的等效运算相比，这种索引查找方法要高效得多。

循环神经网络 (RNN) 通过遍历元素来处理序列输入。RNN 将输出从一个时间步骤传递到其输入，然后传递到下一个步骤。

`tf.keras.layers.Bidirectional` 包装器也可以与 RNN 层一起使用。这将通过 RNN 层向前和向后传播输入，然后连接输出。这有助于 RNN 学习长程依赖关系。

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

请注意，我们在这里选择 Keras 序贯模型，因为模型中的所有层都只有单个输入并产生单个输出。如果要使用有状态 RNN 层，则可能需要使用 Keras 函数式 API 或模型子类化来构建模型，以便可以检索和重用 RNN 层状态。有关更多详细信息，请参阅 Keras RNN 指南。

编译 Keras 模型以配置训练过程：

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

训练模型

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
Epoch 1/10
391/391 [==============================] - 41s 105ms/step - loss: 0.6363 - accuracy: 0
Epoch 2/10
391/391 [==============================] - 41s 105ms/step - loss: 0.3426 - accuracy: 0
Epoch 3/10
391/391 [==============================] - 42s 107ms/step - loss: 0.2520 - accuracy: 0
Epoch 4/10
391/391 [==============================] - 41s 105ms/step - loss: 0.2103 - accuracy: 0
Epoch 5/10
391/391 [==============================] - 42s 106ms/step - loss: 0.1803 - accuracy: 0
Epoch 6/10
391/391 [==============================] - 42s 106ms/step - loss: 0.1589 - accuracy: 0
Epoch 7/10
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================] - 17s 43ms/step - loss: 0.4305 - accuracy: 0.84
Test Loss: 0.43051090836524963
Test Accuracy: 0.8476799726486206
```

上面的模型没有遮盖应用于序列的填充。如果在填充序列上进行训练并在未填充序列上进行测试，则可能导致倾斜。理想情况下，您可以使用遮盖来避免这种情况，但是正如您在下面看到的那样，它只会对输出产生很小的影响。

如果预测 >= 0.5，则为正，否则为负。

```
def pad_to_size(vec, size):
  zeros = [0] * (size - len(vec))
  vec.extend(zeros)
  return vec
```

```
def sample_predict(sample_pred_text, pad):
  encoded_sample_pred_text = encoder.encode(sample_pred_text)

  if pad:
    encoded_sample_pred_text = pad_to_size(encoded_sample_pred_text, 64)
  encoded_sample_pred_text = tf.cast(encoded_sample_pred_text, tf.float32)
  predictions = model.predict(tf.expand_dims(encoded_sample_pred_text, 0))

  return (predictions)
```

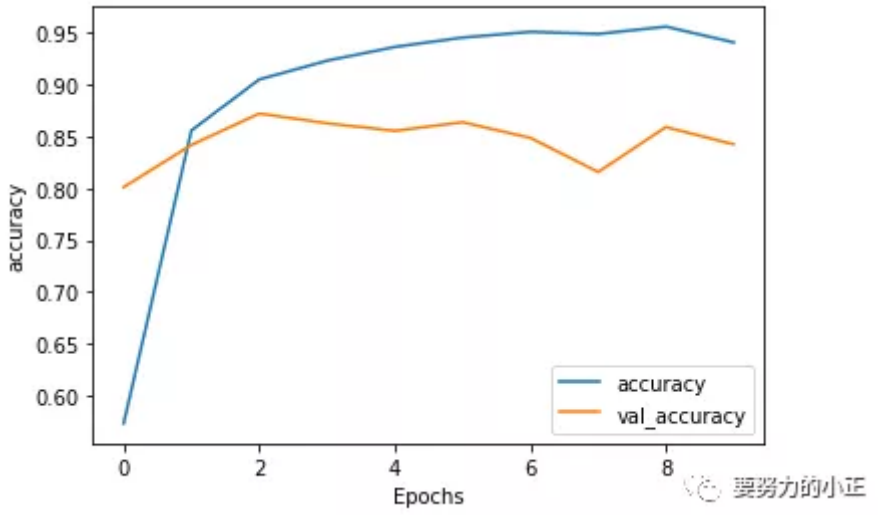
```
# predict on a sample text without padding.

sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
```

```
[[-0.11829309]]
```

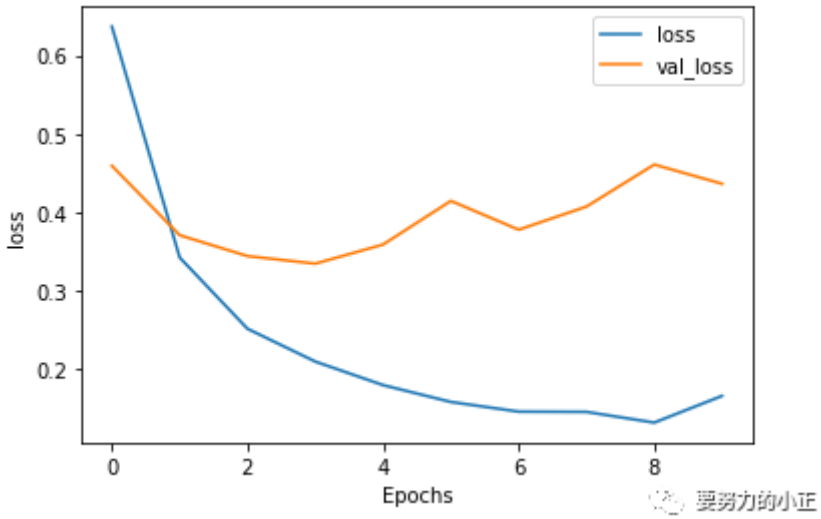
```
# predict on a sample text with padding

sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=True)
print(predictions)
```

```
[[-1.162545]]
```

```
plot_graphs(history, 'accuracy')
```




```
plot_graphs(history, 'loss')
```



堆叠两个或更多 LSTM 层

Keras 循环层有两种可用的模式，这些模式由 `return_sequences` 构造函数参数控制：

- 返回每个时间步骤的连续输出的完整序列（形状为 `(batch_size, timesteps, output_features)` 的 3D 张量）。

- 仅返回每个输入序列的最后一个输出（形状为 (batch_size, output_features) 的 2D 张量）。

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=Tr
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
Epoch 1/10
391/391 [==============================] - 75s 192ms/step - loss: 0.6484 - accuracy: (
Epoch 2/10
391/391 [==============================] - 74s 190ms/step - loss: 0.3603 - accuracy: (
Epoch 3/10
391/391 [==============================] - 75s 191ms/step - loss: 0.2666 - accuracy: (
Epoch 4/10
391/391 [==============================] - 75s 193ms/step - loss: 0.2151 - accuracy: (
Epoch 5/10
391/391 [==============================] - 76s 194ms/step - loss: 0.1806 - accuracy: (
Epoch 6/10
391/391 [==============================] - 75s 193ms/step - loss: 0.1623 - accuracy: (
Epoch 7/10
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================] - 30s 78ms/step - loss: 0.5205 - accuracy: 0.85
Test Loss: 0.5204932689666748
Test Accuracy: 0.857200026512146
```

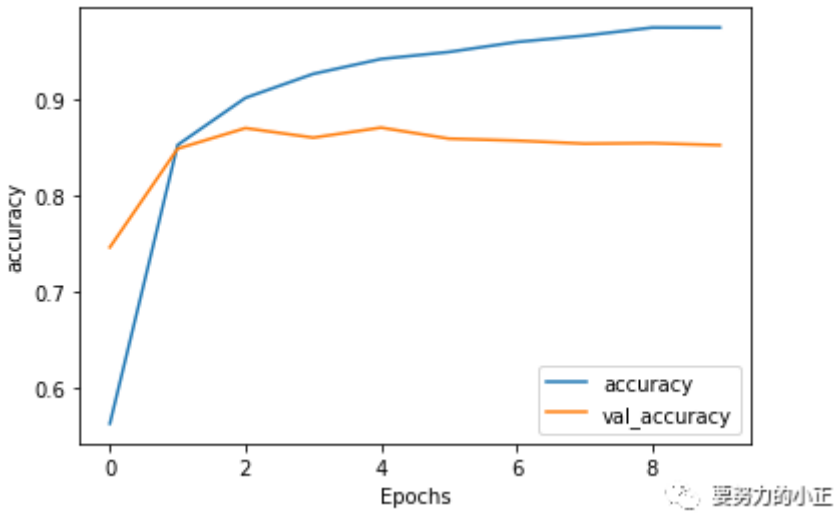
```
# predict on a sample text without padding.

sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
```

```
[[-2.6377363]]
```

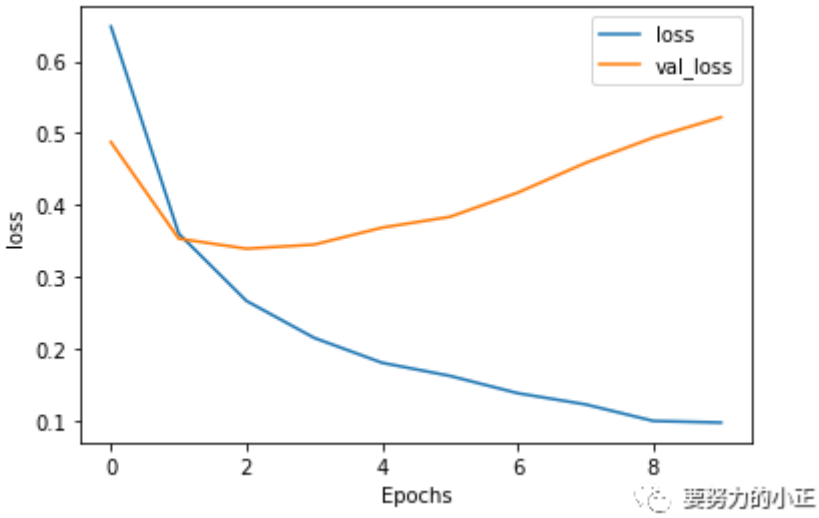
```
# predict on a sample text with padding

sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=True)
print(predictions)
```

```
[[-3.0502243]]
```

```
plot_graphs(history, 'accuracy')
```




```
plot_graphs(history, 'loss')
```



检查其他现有循环层，例如 GRU 层。

完毕！！！！！！！！！！