Homework3 - RNN

2025 年 5 月 9 日

SC24219070 朱越

1 导入库与加载数据

程序结构参考了各论坛相关例程内容，把基础 RNN 结构中的 SimpleRNN 块换成了 LSTM 块。

1.1 加载必要的库，然后使用 tensorflow.keras 中的工具加载数据集

[1]:
```
import os
os.environ['KERAS_HOME'] = './datas/keras'  # 设置数据集路径
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Embedding, LSTM, Dense
```

[2]:
```
# 加载数据集
num_words = 10000
(train_data, train_labels), (test_data, test_labels) = imdb.
 ↪load_data(num_words=num_words, seed=42)
class_names = ["Negative", "Positive"]
```

1.2 先打印一部分训练集看一下导入的情况

[3]:
```python
# 获取单词到整数的映射字典
word_index = imdb.get_word_index()
# 反转字典（整数 -> 单词）
word_list = {v: k for k, v in word_index.items()}
# 将整数序列解码为文本（注意索引偏移 3: 0=padding, 1=start, 2=unknown）
decoded_review = ' '.join([word_list.get(i - 3, '?') for i in train_data[0]])
print("训练样本数:", len(train_data))  # 输出: 25000
print("测试样本数:", len(test_data))   # 输出: 25000
print("第一条评论的整数序列:", train_data[0][:500])
print("第一条评论的文本内容:", decoded_review[:500])
print("第一条评论的标签:", class_names[train_labels[0]])  # 0 或 1
```

```
训练样本数: 25000
测试样本数: 25000
第一条评论的整数序列: [1, 11, 4079, 11, 4, 1986, 745, 3304, 299, 1206, 590, 3029,␣
↪1042,
37, 47, 27, 1269, 2, 7637, 19, 6, 3586, 15, 1367, 3196, 17, 1002, 723, 1768,
2887, 757, 46, 4, 232, 1131, 39, 107, 3589, 11, 4, 4539, 198, 24, 4, 1834, 133,
4, 107, 7, 98, 413, 8911, 5835, 11, 35, 781, 8, 169, 4, 2179, 5, 259, 334, 3773,
8, 4, 3497, 10, 10, 17, 16, 3381, 46, 34, 101, 612, 7, 84, 18, 49, 282, 167, 2,
7173, 122, 24, 1414, 8, 177, 4, 392, 531, 19, 259, 15, 934, 40, 507, 39, 2, 260,
77, 8, 162, 5097, 121, 4, 65, 304, 273, 13, 70, 1276, 2, 8, 15, 745, 3304, 5,
27, 322, 2197, 2, 2, 70, 30, 2, 88, 17, 6, 3029, 1042, 29, 100, 30, 4943, 50,
21, 18, 148, 15, 26, 5980, 12, 152, 157, 10, 10, 21, 19, 3196, 46, 50, 5, 4,
1636, 112, 828, 6, 1003, 4, 162, 5097, 2, 517, 6, 2, 7, 4, 9527, 5593, 4, 351,
232, 385, 125, 6, 1693, 39, 2383, 5, 29, 69, 5593, 5670, 6, 162, 5097, 1567,
232, 256, 34, 718, 5612, 2980, 8, 6, 226, 762, 7, 2, 7830, 5, 517, 2, 6, 3242,
7, 4, 351, 232, 37, 9, 1861, 8, 123, 3196, 2, 5612, 188, 5165, 857, 11, 4, 86,
22, 121, 29, 1990, 1495, 10, 10, 1276, 61, 514, 11, 14, 22, 9, 1456, 9533, 14,
575, 208, 159, 9533, 16, 2, 5, 187, 15, 58, 29, 93, 6, 2, 7, 395, 62, 30, 1211,
493, 37, 26, 66, 2, 29, 299, 4, 172, 243, 7, 217, 11, 4, 2, 7106, 22, 4, 2,
1038, 13, 70, 243, 7, 3468, 19, 9533, 11, 15, 236, 1313, 136, 121, 29, 5, 5612,
26, 112, 4382, 180, 34, 3304, 1768, 5, 320, 4, 162, 5097, 568, 319, 4, 3324,
5235, 1456, 269, 8, 401, 56, 19, 5612, 16, 142, 334, 88, 146, 243, 7, 11, 2,
2756, 150, 11, 4, 2, 2550, 10, 10, 7173, 828, 4, 206, 170, 33, 6, 52, 4968, 225,
```

```
55, 117, 180, 58, 11, 14, 22, 48, 50, 16, 101, 329, 12, 62, 30, 35, 6637, 1532,
22, 4079, 11, 4, 1986, 1199, 35, 735, 18, 118, 204, 881, 15, 291, 10, 10, 7173,
82, 93, 52, 361, 7, 4, 162, 5097, 2, 5, 4, 785, 6542, 49, 7, 4, 172, 2572, 7,
665, 26, 303, 343, 11, 23, 4, 2, 11, 192, 4079, 11, 4, 1986, 9, 44, 84, 24, 2,
54, 36, 66, 144, 11, 68, 205, 118, 602, 55, 729, 174, 8, 23, 4, 2, 10, 10, 4079,
11, 4, 1986, 127, 316, 2606, 37, 16, 3445, 19, 12, 150, 138, 426, 2, 7173, 79,
49, 542, 162, 5097, 4413, 84, 11, 4, 392, 555]
第一条评论的文本内容: ? in panic in the streets richard widmark plays u s navy␣
↪doctor who
has his week ? interrupted with a corpse that contains plague as cop paul
douglas properly points out the guy died from two bullets in the chest that's
not the issue here the two of them become unwilling partners in an effort to
find the killers and anyone else exposed to the disease br br as was pointed out
by any number of people for some reason director ? kazan did not bother to cast
the small parts with anyone that sounds li
第一条评论的标签: Positive
```

可以看到数据集已经正确导入了。

另外，因为 keras 中导入的 imdb 库已经执行了文本的预处理——它将数据集中出现的单词根据词频排序并编码——因此不需要再执行文本处理了。

1.3 定义了训练中用到的变量、定义了网络结构、损失函数和优化方法

在训练前需要先对文本切片和标准化。

网络搭建和优化方法、损失函数、评价指标都使用了 tensorflow.keras 库中的集成方法。

感觉仿佛什么都没做，但是已经做完了，哈哈！

[4]:
```python
epochs = 20
batch_size = 128
sequences_len = 500

# 将序列填充/截断为固定长度（例如 500 单词）
train_sequences = sequence.pad_sequences(train_data, maxlen=sequences_len)
test_sequences = sequence.pad_sequences(test_data, maxlen=sequences_len)
model = Sequential([
```

```
    # 嵌入层：将整数索引映射为 32 维向量
    Embedding(input_dim=num_words, output_dim=32),
    # LSTM 层：32 个隐藏单元
    LSTM(32, dropout=0.2, recurrent_dropout=0.2),  # 使用 LSTM 单元
    # 输出层：二分类（sigmoid）
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

打印网络的情况观察一下。

```
[5]: model.build(input_shape=(batch_size, sequences_len))  # (batch_size,␣
  ↪sequence_length)
model.summary()
```

```
Model: "sequential"



 Layer (type)                           Output Shape                              ␣
↪Param #

 embedding (Embedding)                  (128, 500, 32)                            ␣
↪320,000

 lstm (LSTM)                            (128, 32)                                   ␣
↪8,320

 dense (Dense)                          (128, 1)                                      ␣
↪ 33



 Total params: 328,353 (1.25 MB)


 Trainable params: 328,353 (1.25 MB)
```

```
Non-trainable params: 0 (0.00 B)
```

2 开始炼！

训练也只有一句啊！

```
[6]: history = model.fit(train_sequences, train_labels, epochs=epochs,␣
      ↪batch_size=batch_size, validation_split=0.2)
```

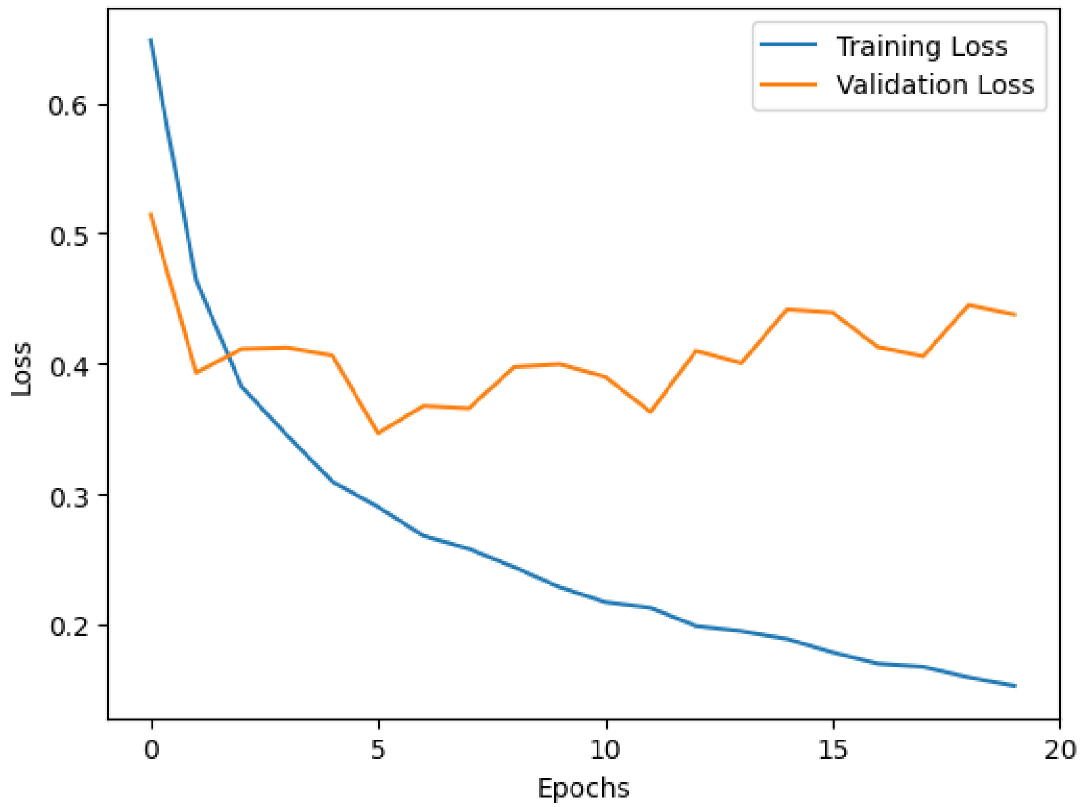
```
Epoch 1/20
157/157                  70s 427ms/step -
accuracy: 0.5692 - loss: 0.6798 - val_accuracy: 0.7746 - val_loss: 0.5147
Epoch 2/20
157/157                  64s 409ms/step -
accuracy: 0.7751 - loss: 0.4957 - val_accuracy: 0.8338 - val_loss: 0.3933
Epoch 3/20
157/157                  67s 427ms/step -
accuracy: 0.8337 - loss: 0.3945 - val_accuracy: 0.8132 - val_loss: 0.4114
Epoch 4/20
157/157                  69s 437ms/step -
accuracy: 0.8574 - loss: 0.3514 - val_accuracy: 0.8130 - val_loss: 0.4125
Epoch 5/20
157/157                  74s 471ms/step -
accuracy: 0.8811 - loss: 0.3080 - val_accuracy: 0.8182 - val_loss: 0.4064
Epoch 6/20
157/157                  69s 440ms/step -
accuracy: 0.8869 - loss: 0.2913 - val_accuracy: 0.8546 - val_loss: 0.3467
Epoch 7/20
157/157                  69s 437ms/step -
accuracy: 0.9012 - loss: 0.2617 - val_accuracy: 0.8386 - val_loss: 0.3677
Epoch 8/20
157/157                  69s 439ms/step -
accuracy: 0.8999 - loss: 0.2620 - val_accuracy: 0.8506 - val_loss: 0.3659
Epoch 9/20
```

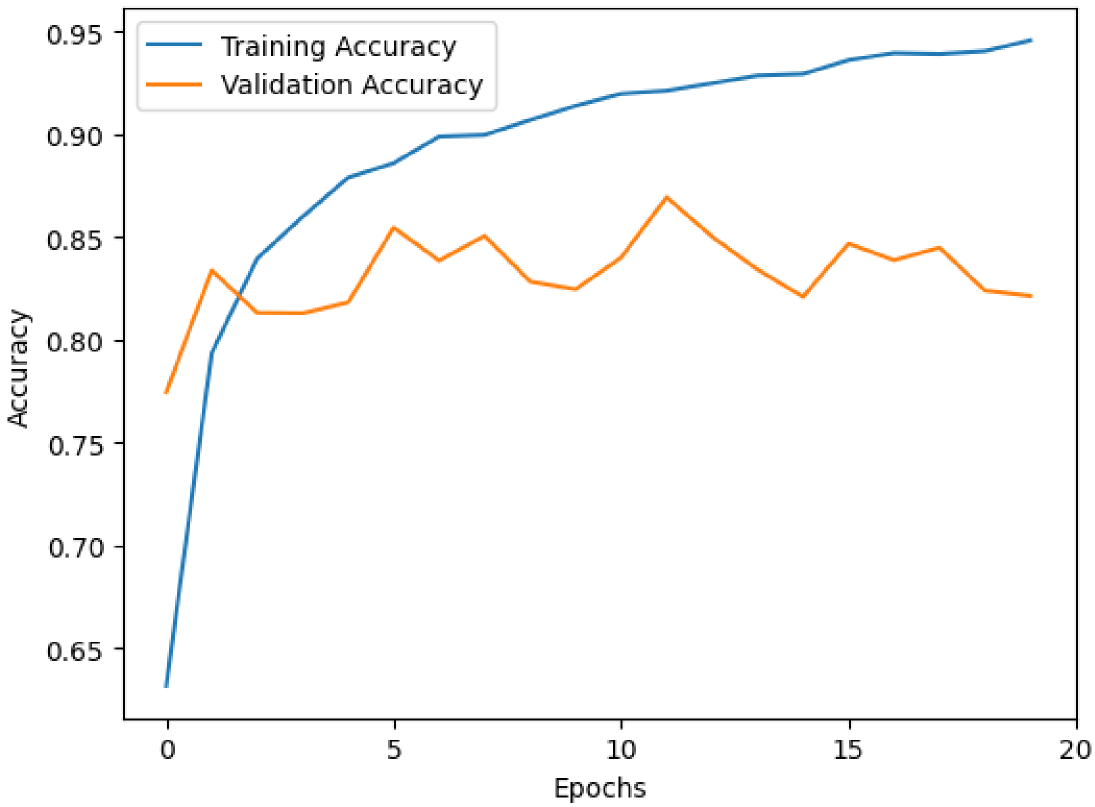
```
157/157              69s 440ms/step -
accuracy: 0.9071 - loss: 0.2387 - val_accuracy: 0.8284 - val_loss: 0.3976
Epoch 10/20
157/157              70s 446ms/step -
accuracy: 0.9147 - loss: 0.2268 - val_accuracy: 0.8246 - val_loss: 0.3997
Epoch 11/20
157/157              70s 444ms/step -
accuracy: 0.9223 - loss: 0.2083 - val_accuracy: 0.8400 - val_loss: 0.3901
Epoch 12/20
157/157              68s 435ms/step -
accuracy: 0.9225 - loss: 0.2111 - val_accuracy: 0.8694 - val_loss: 0.3631
Epoch 13/20
157/157              69s 441ms/step -
accuracy: 0.9237 - loss: 0.2016 - val_accuracy: 0.8502 - val_loss: 0.4100
Epoch 14/20
157/157              69s 441ms/step -
accuracy: 0.9299 - loss: 0.1913 - val_accuracy: 0.8342 - val_loss: 0.4006
Epoch 15/20
157/157              69s 438ms/step -
accuracy: 0.9324 - loss: 0.1841 - val_accuracy: 0.8210 - val_loss: 0.4418
Epoch 16/20
157/157              70s 444ms/step -
accuracy: 0.9367 - loss: 0.1763 - val_accuracy: 0.8468 - val_loss: 0.4394
Epoch 17/20
157/157              70s 445ms/step -
accuracy: 0.9416 - loss: 0.1636 - val_accuracy: 0.8388 - val_loss: 0.4130
Epoch 18/20
157/157              72s 455ms/step -
accuracy: 0.9431 - loss: 0.1570 - val_accuracy: 0.8448 - val_loss: 0.4058
Epoch 19/20
157/157              71s 449ms/step -
accuracy: 0.9397 - loss: 0.1587 - val_accuracy: 0.8240 - val_loss: 0.4451
Epoch 20/20
157/157              87s 554ms/step -
accuracy: 0.9485 - loss: 0.1464 - val_accuracy: 0.8214 - val_loss: 0.4379
```

3 训练结果与后处理

3.1 网络损失和分类正确率的可视化

[7]:
```
# 绘制训练集和验证集的损失
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.xticks(range(0, epochs+1, 5))
plt.ylabel('Loss')
plt.legend()
plt.show()
# 绘制训练集和验证集的准确率
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.xticks(range(0, epochs+1, 5))
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```


Training Loss
Validation Loss
0.6
0.5
0.4
0.3
0.2
Loss
0
5
10
15
20
Epochs



根据曲线可以看出，网络在 5 次训练左右的验证集 Loss 值进入一个低谷。

后续的训练只有 train_acc 升高而 val_acc 不升高，表示开始进入过拟合。

不过 RNN 网络的训练总体来说没有 CNN 平稳，当训练次数达到 10 以上时还出现了非常明显的振荡。

3.2 抽样检查识别结果

[8]:
```
# 1. 抽取 10 个样本
# 因为导入训练集时已经随机提取了，所以这里就不随机了。
indices = list(range(11, 21))
sampled_sequences = train_sequences[indices]
sampled_labels = train_labels[indices]
# 2. 使用模型预测
predictions = model.predict(sampled_sequences)
```

```
predicted_classes = (predictions > 0.5).astype(int)  # 二分类: sigmoid 输出转 0/1
# 3. 打印对比结果
for i in range(len(indices)):
    true_label = class_names[sampled_labels[i]]
    pred_label = class_names[predicted_classes[i][0]]
    print(f"Index: {indices[i]}, True Label: {true_label}, Predicted Label: {pred_label} ({predictions[i][0]:.4f})")  # 同时输出原始概率值
```

```
1/1              1s 594ms/step
Index: 11, True Label: Positive, Predicted Label: Positive (0.9987)
Index: 12, True Label: Positive, Predicted Label: Positive (0.9991)
Index: 13, True Label: Negative, Predicted Label: Negative (0.0054)
Index: 14, True Label: Positive, Predicted Label: Positive (0.9974)
Index: 15, True Label: Negative, Predicted Label: Negative (0.0241)
Index: 16, True Label: Negative, Predicted Label: Negative (0.0763)
Index: 17, True Label: Negative, Predicted Label: Negative (0.4817)
Index: 18, True Label: Positive, Predicted Label: Positive (0.9981)
Index: 19, True Label: Positive, Predicted Label: Positive (0.9797)
Index: 20, True Label: Positive, Predicted Label: Positive (0.9268)
```

可以看到，基本所有的样本都正确识别了，而且原始概率值的区分度也比较大（置信度较高）。

3.3 保存模型、统计准确率

- 执行了一次保存和载入。
- 怎么评价也只有一句!

[9]:
```
model.save('./models/HW3_RNN.keras')  # 保存为 keras 格式
```

[10]:
```
# 评估测试集
net = load_model('./models/HW3_RNN.keras')  # 加载 keras 文件
test_loss, test_acc = net.evaluate(test_sequences, test_labels)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
782/782              60s 75ms/step -
accuracy: 0.8417 - loss: 0.4078
Test Accuracy: 0.8372
```

根据得到的结果可以看出，网络在测试集上的准确率达到了 83.72，已经属于单层 LSTM 不错的水平了！

4 总结与反思

- 本次实验使用了一个 Sequential 结构的简单 RNN 模型，使用了 LSTM 块作为隐藏层单元，实现了 85% 左右的分类准确率。
- 感觉这个 RNN 网络训练比 Alexnet 快多了，准确率也高不少，可能是二分类任务更容易一些叭！