

Kera 实现 RNN 神经网络

2021年2月13日 21:22

序

没想到春节还是要拜年的，除夕春节本来想接着干活没想到还是给自己放了两天假，到今天愣是没转过来，差不多相当于三天在摸鱼了。。。不过直至 2/13 日晚已经将状态慢慢调整回来了（也就是写这篇博客的日期）。

什么是 Kera ?

Keras是一个高层神经网络API，Keras由纯Python编写而成并基于Tensorflow、Theano以及CNTK后端。Keras 为支持快速实验而生，能够把你的idea迅速转换为结果，如果你有如下需求，请选择Keras：

- 简易和快速的原型设计（keras具有高度模块化，极简，和可扩充特性）
- 支持CNN和RNN，或二者的结合
- 无缝CPU和GPU切换

来自 <<https://keras-cn.readthedocs.io/en/latest/>>

上面这段话完完整整地摘抄自一个良心网站（在来自中注明）。是一个 Kera 的中文文档网站（非官方，用爱发电维护，不过因为官方中文文档的出现（我还没去看过）现在已经停止维护。）。

Kera 也是我咨询的学长前辈的推荐 API。自己这次实践也是通过它完成，认为还是很好上手的，至于后续算法改进如果有需求再去用 tensorflow 做更底层的实现。

并且，似乎 tensorflow 已经将其作为官方 API，安装了较新版的 tensorflow 以后，不需要再去额外安装 keras，就自带了它的包。

```
from tensorflow.keras import Sequential, layers
```

就像这样使用就行了。

实现

```
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow as tf
from tensorflow.keras import Sequential, layers
# 利用 matplotlib 可视化训练过程
# 解决中文乱码问题
# 字体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 字体大小
plt.rcParams['font.size'] = 20
def plot_graphs(history, name):
    # 描点
    plt.plot(history.history[name])
    plt.plot(history.history['验证集 - ' + name])
    # 横坐标为迭代次数
    plt.xlabel("Epochs")
    # 纵坐标为设置的参数（如准确率 accuracy）
    plt.ylabel(name)
    # 备注版
    plt.legend([name, '验证集 - ' + name])
# 呈现
```

```

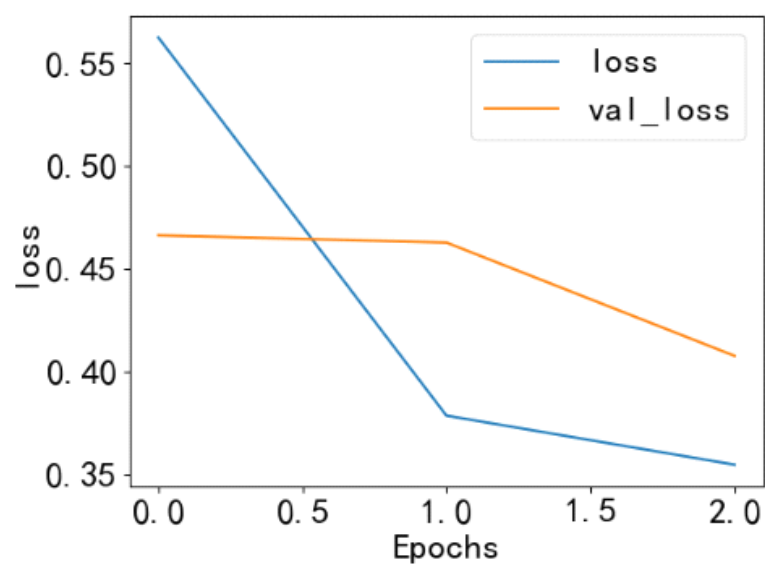
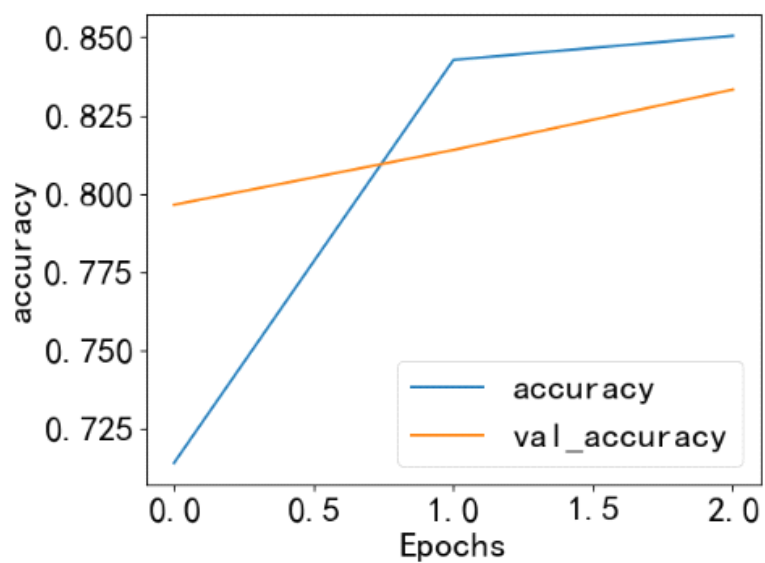
plt.show()
# 获取数据集。默认返回形式为字典，可以设置为以元组返回 ( as_supervised )
ds, info = tfds.load('imdb_reviews/subwords8k', with_info=True,
                    as_supervised=True)

# 获取训练集，测试集
train_ds, test_ds = ds['train'], ds['test']
# 设置缓冲区大小，批大小
BUFFER_SIZE, BATCH_SIZE = 10000, 64
# 打乱 BUFFER_SIZE 缓冲区大小内的数据，并始终保持缓冲区大小的乱序数据（不精确的理解）
train_ds = train_ds.shuffle(BUFFER_SIZE)
# TODO 读取数据。padded_batch 这个函数还不是很了解，文档也说得云里雾里
train_ds = train_ds.padded_batch(BATCH_SIZE, tf.compat.v1.data.get_output_shapes(train_ds))
test_ds = test_ds.padded_batch(BATCH_SIZE, tf.compat.v1.data.get_output_shapes(test_ds))
# TODO 获取 tokenizer。用进行字符处理级id转换（这里先转换成 subword，再转换为 id）等操作（可能意思是先转换为单词
再给单词打上编号？）
# 总之从结果上来看的话，这里完成对字符文本串的序列化，因为 RNN 或者说所有的 NN 是处理数字的神经网络，需要将文本变
成数字编号才能进行训练
tokenizer = info.features['text'].encoder
print ('词汇个数:', tokenizer.vocab_size)
# 试着将这句话中的单词进行分解编码
sample_str = 'it is write by vanot313.'
tokenized_str = tokenizer.encode(sample_str)
print ('向量化文本:', tokenized_str)
# 打印结果
for ts in tokenized_str:
    print (ts, '-->', tokenizer.decode([ts]))
# 借助 keras 来搭建 RNN 模型
model = Sequential([
    # TODO 输入层。必须位于第一层，将正整数（下标）转换为具有固定大小的向量，如[[4], [20]]->
    [[0.25, 0.1], [0.6, -0.2]]（这个例子什么意思，为什么要这么做）
    layers.Embedding(tokenizer.vocab_size, 64),
    # TODO 用双向 RNN 包装器包装 LSTM 层。（这里需要去补双向 RNN 的知识）
    # 这里还可以添加更多的 LSTM 层来增加性能。
    layers.Bidirectional(layers.LSTM(64)),
    # Dense: 全连接层。
    # 采用 relu 函数来做激活函数。
    # 其他关键参数意义（在这里没有作为参数输入）：
    # kernel: 权值矩阵
    # bias: 偏置（偏差）向量
    layers.Dense(64, activation='relu'),
    # 全连接层（输出），采用 sigmoid 函数来做输出激活函数
    layers.Dense(1, activation='sigmoid')
])
# TODO 这里模型的设计原来不一定全是 LSTM 层吗？全连接层与简单的 RNN 结构一样吗？
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])
# 训练，迭代三次。不过这里相较于之前自己写的 demo 训练的参数只设置了迭代次数。
history1 = model.fit(train_ds, epochs=3, validation_data=test_ds)
# 和之前自己写的简单 demo 一样。evaluate 函数代表估值函数，即输入测试集，进行完整的测试。
loss, acc = model.evaluate(test_ds)
print('准确率:', acc)
print('损失函数值:', loss)
plot_graphs(history1, 'accuracy')
plot_graphs(history1, 'loss')
# 保存模型
model.save("testmodel")

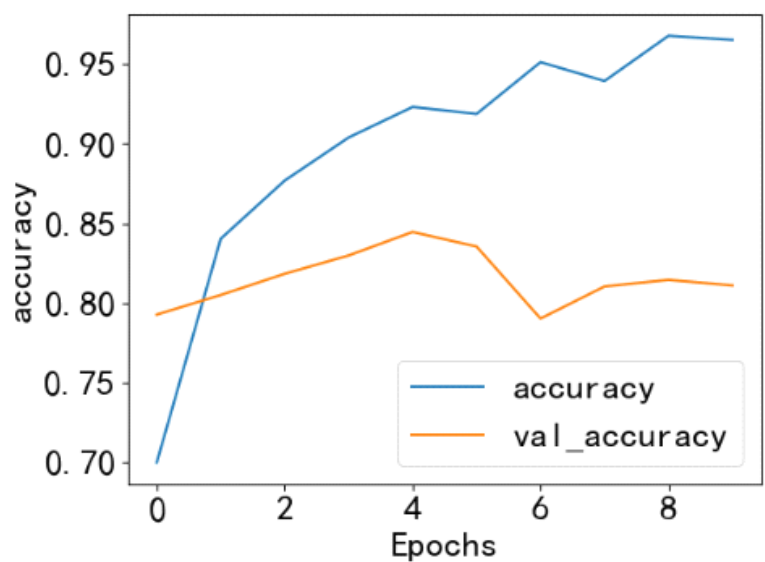
```

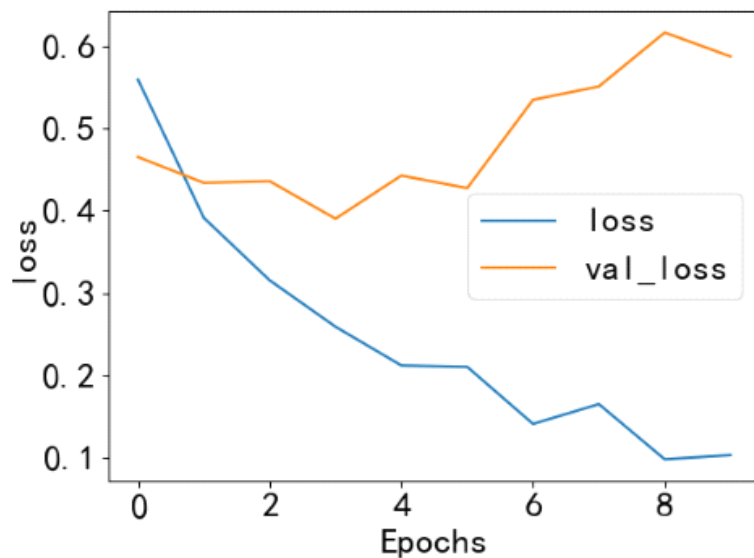
可视化的训练结果

迭代 3 次的结果



迭代 10 次的结果





其实可以看到过多的迭代对于准确率来说没有什么增益，相反是有过拟合（在训练集表现优异，但在测试集表现一般）的趋势。

问题

暂时存在的问题已经在源码中用 TODO 标识出。这里再次汇个总。

- ☑ TODO 读取数据。padded_batch 这个函数还不是很了解，文档也说得云里雾里

简单来说就是将大的样本进行分批操作（回忆一下随机梯度下降 SGD），以及对变长样本进行填充处理。

<https://blog.csdn.net/cqupt0901/article/details/108030260>

- ☑ 获取 tokenizer。用进行字符处理级id转换（这里先转换成 subword，再转换为 id）等操作（可能意思是先转换为单词再给单词打上编号？）

我的想法是对的。

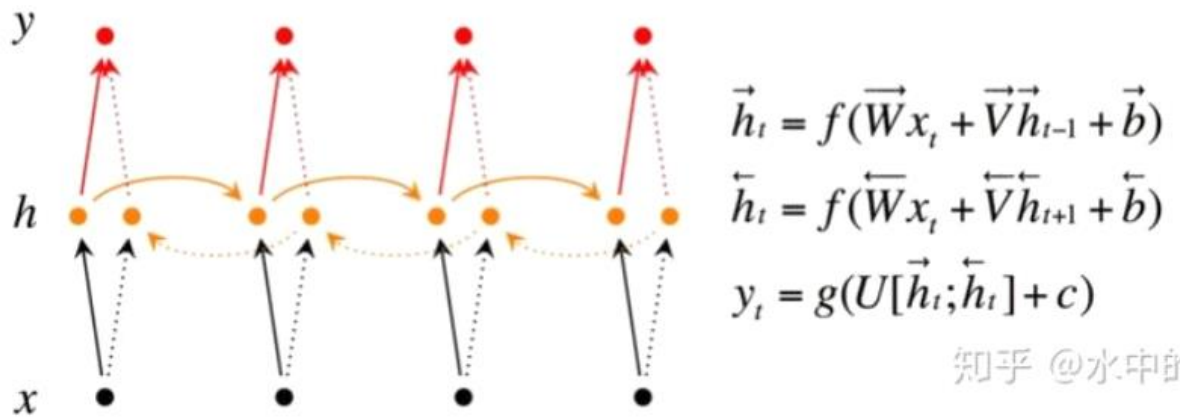
- ☑ TODO 输入层。必须位于第一层，将正整数（下标）转换为具有固定大小的向量，如[[4],[20]]->[[0.25, 0.1],[0.6, -0.2]]（这个例子什么意思，为什么要这么做）

首先纠正一点，embedding 层并不是传统意义上的输入层，他的任务是将下标（如该 demo 中将单词序列化得到后的单词 id）转化为固定长度的向量，其本身也属于可训练的一层，其单词到向量的映射可以被训练，在经过训练后，同意义的单词的向量应当趋同。

目前对 embedding 的理解来看比较好的应用场景只有词义解析？似乎在本次项目中不好应用？

- ☑ TODO 用双向 RNN 包装器包装 LSTM 层。（这里需要去补双向 RNN 的知识）

经典的 RNN 中状态的传递是单向的从前往后传递的，在具体问题中体现为其只能结合前面学习到的状态做当前判断，而面对需要结合上下文理解的问题无能为力。在这种情况下双向 RNN 给出了解决方案。



其每一个状态都需要结合过去的状态与未来的状态来做判断。

Q: 当前的状态怎么会结合未来的状态呢? 未来的状态对于当前来说不是未知的吗?

A: 用双向 RNN 包裹的层做了两边计算, 一遍正序, 一遍倒序, 以此实现。

☑ TODO 这里模型的设计原来不一定全是 LSTM 层吗? 全连接层与简单的 RNN 结构一样吗?

不一定全是 LSTM 层, LSTM 层只是提供了长期记忆功能的一层罢了。不过全连接层 Dense 在结构上与 RNN 结构类似, 应该作用上也差不多。

☑ TODO 这次的 demo 是从远程下载的数据集直接投入测试, 如果是本地的数据集要如何操作?

直接读取本地数据文件再经过必要处理得到输入所需的数据类型即可。

末

经由这个 demo 了解了成熟的网络模型的搭建步骤, 同时也发现了很多问题。

另外这个模型训练的时间要比我想象的久太多了。。。没想到情人节到来的夜晚是这个东西陪伴在我身边。

2/15 日-记

因为昨天还遗留了许多问题没有解决, 今天回过头来彻底理解了一遍。

首先, 明确这个 demo 解决的是什么问题: 它通过对 IMBD 影评数据的学习, 实现对影评的情绪二元判断 (好 或 坏)。

来一步步解析这个过程。

- Tensorflowdatasets 获取目标训练集测试集 -> 训练集与测试集

正文	评价
It is awful	0
Oh I love it	1
...

- 对正文中的单词进行解析, 将其转化为数据序列, 其中编号规则依照出现频率制定 -> 序列化正文内容的训练集与测试集

该步在获取到数据时就已经完成了预处理, 但是还是应当了解这个预处理的过程。

正文	评价
19 8 752	0
33 20 567 90	1
...

- padding_batch 划分批次与处理变长数据 (因为每篇评论长度都不相同, 需要将其长度同一才能进行正确的批处理操作)

```
(array([[ 135, 7968,    8, ...,    0,    0,    0],
       [2016,  131,  258, ...,    0,    0,    0],
       [ 701, 1296, 7961, ...,    0,    0,    0],
       ...,
       [ 135, 7968,    8, ...,    0,    0,    0],
       [6000,  105,  579, ...,    0,    0,    0],
       [ 62,   9,   4, ...,    0,    0,    0]], dtype=int64), array([1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1,
1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1],
dtype=int64))
*****
```

结果类似于这样，每一批之间在这里体现为用 * 号分隔开。

○ 投入到神经网络中训练

根据目前的判断，也根据 RNN 的功能推测，神经网络每次的输入只是一个单词，当一个句子中的单词全部识别完成后，才算是完成对一个样本的训练。（如上图中一个数字代表一个序列化的单词，包裹单词的 list [] 代表一个句子，包裹一个个句子的 list 代表一个批次）

▪ 对单词的额外处理

也就是 embedding 层做的工作将单词序列转化为一个定长的向量。

这差不多就是整个的 demo 数据结构的变化过程。当然这也只是 demo 的流程。似乎今天路走偏了，不应该在这个 demo 上花太多时间。以目前的进度而言，应当尽早开始针对项目的开发。

参考博文

<https://keras-cn.readthedocs.io/en/latest/>

<https://geektutu.com/post/tf2doc-rnn-lstm-text.html>

https://blog.csdn.net/qg_31456593/article/details/89923645

https://blog.csdn.net/qg_31456593/article/details/89923645

<https://www.pythonheidong.com/blog/article/312534/b731a3b5748fa6ab9d5c/>

https://blog.csdn.net/qg_43203949/article/details/108571290