13.0 前言

13.0.1 课程目标

本课程的目标是学习怎么样完成一个LSTM模型，并用它在新数据上做预测。完成本课的学习之后，你将会学习到：

- 怎么样为你的工程开发一个最终的LSTM模型；
- 怎么样将LSTM模型保存到文件以便后面使用；
- 怎么样在新的数据上加载LSTM模型来做预测。

13.0.2 课程概览

本课程分为三个部分，它们是：

1. 什么是最终LSTM模型；
2. 将LSTM模型保存到文件；
3. 在新数据上做预测。

让我们开始吧！

13.1 完成一个LSTM模型

在本节中，你将会学习到怎么样去完成你的LSTM模型。

13.1.1 什么是一个最终LSTM模型?

在本章节中，你将会发现怎么样定型你的LSTM模型。

最后一个LSTM模型是用来预测新数据的模型。也就是说，给定输入数据的新示例，你希望在使用该模型来预测预期输出。这可以是一个分类（指定一个标签）或一个回归（一个实值）。你的序列预测项目的目标达到一个最好的模型，其中最好的定义是：

- 数据：你可用的历史数据；
- 时间：工程上你必须花费的时间；
- 过程：数据准备步骤，算法，以及所选择的算法配置；

在你的工程中，你将数据聚在一起，花费你的时间，并发现数据准备过程，使用的算法，以及如何配置它。这个模型是这个过程的顶点，你寻找的目的是为了开始实际的预测。没有完美的模型。你能够发现的只有最好的模型。

13.1.2 使用训练或测试集的目的

创建你数据集的训练和测试拆分是一种快速评估算法对问题性能的方法。训练数据集被用来准备模型，训练它。我们假设测试数据集是新的数据，其中输出值从算法中被保留。我们从训练模型中手机来自测试数据集测输入的预测，并将它们与测试集的保留输出值进行比较。

将测试与预测数据集中的保留输出进行比较，使我们能够在测试数据集上计算模型的性能度量。这是在对未知数据进行预测时对为进行训练的算法的技巧的估计。使用K折交叉验证是一种更稳健和更昂贵的计算方法。我们使用我们在训练数据集上的LSTM模型技能的估计作为代理来估计在新的数据集上做预测时，模型的技能会是什么样的。

这是一个相当大的飞跃，需要：

- 你所使用的程序是非常稳健的，技能的估计接近我们对未知数据的实际期望；
- 性能测量的选择准确地捕获了我们对未知数据的预测感兴趣的测量；
- 数据准备的选择在新数据上是很好理解的并且可重复的，如果预测需要返回到其原始规模或与原始输入值相关则并且是可逆的；
- 模型架构和配置的选择对于其预期使用和操作环境（例如复杂性）是有意义的。

很多骑上在测试集或k折交叉验证程序的整个过程的估计技能。

13.1.3 怎么样最终化LSTM模型?

通过在所有数据上应用所选择的LSTM体系结构和配置来完成模型。没有训练和测试分割，没有交叉验证折叠。把所有的数据重新组合到一个大的训练数据集中，并拟合你的模型。就是这样。用最终的模型，你可以：

- 保存模型用于以后或操作使用；
- 加载模型并对新数据进行预测。

为什么不保持最好的训练模式呢?

你的LSTM模型可能需要数天或者数周来准备。在这种情况下，你可能希望将模型保持在训练数据集上，而不是需要在训练和测试集的组合上。这是对附加数据进行模型训练的可能条件和用于拟合新模型的时间和计算成本之间的权衡。

最终模型上的性能会不一样吗?

使用鲁棒的测试线束的全部想法是估计最终模型的技能。理想情况下，估计与最终模型中观察到的技能之间的差异在测量误差点上是次要的，或者技能被提升为用于弄下的训练实例的数量的函数。你可以通过对模型技能与训练实例的数量进行敏感性分析来测试这两个假设。

每次训练的最后模式不会不同吗?

你用来选择最终模型的技能估计应该在多次运行中进行平均。这样，你知道，平均所选的模型架构和配置是熟练的。你可以尝试通过训练多个最终模型和在实践中使用它们的预测的集合或平均来控制模型的随机性质。同样，你可以设计一个灵敏度分析来测试这是否会导致一组更稳定的预测。

13.2 把LSTM模型保存到文件

Keras提供了一个API来允许你将你的模型保存为文件。有两个选项：

1. 将模型保存为一个单一的文件；
2. 将模型的结构和权重分开保存为文件。

在这两种情况下，使用HDF5文件格式来在磁盘上大量存储大量的数字。你需要确认你安装了h5py的Python库。可以安装如下：

```
sudo pip install h5py
```

表 13.1 安装所需的h5py Python库

13.2.1 保存为单一文件

你可以使用模型的save()函数保存一个拟合的Keras模型到文件。例如：

```
# define model model = Sequential()
model.add(LSTM(...))
# compile model
model.compile(...)
# fit model
model.fit(...)
```

```
# save model to single file
model.save( 'lstm_model.h5' )
```

表 13.2 保存拟合LSTM模型到一个单一文件的例子

这个单一文件包含模型结构和权重。它还包括制定的损失和优化算法的规范，以便你可以恢复训练。该模型可以使用load_model()函数来再次加载（来自于不同的Python会话中的不同脚本）。

```
from keras.models import load_model
# load model from single file
model = load_model( 'lstm_model.h5' )
# make predictions
yhat = model.predict(X, verbose=0)
print(yhat)
```

表 13.3 从单一文件中加载一个保存了的LSTM模型的例子

下面是一个完整的LSTM模型的例子，将它保存到一个文件中，然后加载它。虽然模型的加载在同一个脚本中，但是这个部分可以从另一个Python会话中的另一个脚本运行。运行例子将模型保存到文件lstm_model.h5文件中。

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from numpy import array
from keras.models import load_model


# return training data
def get_train():
    seq = [[0.0, 0.1], [0.1, 0.2], [0.2, 0.3], [0.3, 0.4], [0.4, 0.5]]
    seq = array(seq)
    X, y = seq[:, 0], seq[:, 1]
    X = X.reshape((len(X), 1, 1))
    return X, y


# define model
model = Sequential()
model.add(LSTM(10, input_shape=(1,1)))
model.add(Dense(1, activation= 'linear' ))
# compile model
model.compile(loss= 'mse' , optimizer= 'adam' )
# fit model
X,y = get_train()
model.fit(X, y, epochs=300, shuffle=False, verbose=0)
# save model to single file
model.save( 'lstm_model.h5' )
# snip...
# later, perhaps run from another script

# load model from single file
model = load_model( 'lstm_model.h5' )
```

```
# make predictions
yhat = model.predict(X, verbose=0)
print(yhat)
```

表 13.4 保存一个拟合的LSTM模型到一个单一的文件并稍后再次加载它的例子

13.2.2 保存到分开的文件

你可以保存模型的结构（例如，层和它们间是如何连接的）和权重（数字的数组）到分别的文件中。我推荐这种方法，因为它允许你开发更新的模型权重并替换一个模型，同时确保模型的结构保持不变。

保存结构

Keras提供了两种保存模型结构的格式：JSON和YAML格式。这些格式的好处是它们是人类可读的。选择实际上是一个口味和偏好的问题。可以调用to_json()或者to_yaml()函数来将你的模型分别保存为JSON或者YAML格式的文件。它们返回一个标准的字符串，你可以使用标准的Python文件open()和write()函数将这个字符串保存为一个ASCII文件到硬盘。

```
...
# convert model architecture to JSON format
architecture = model.to_json()
# save architecture to JSON file
with open( 'architecture.json' , 'wt' ) as json_file:
    json_file.write(architecture)
```

表 13.5 保存一个拟合LSTM模型结构到文件的例子

使用 model_from_json()或者 model_from_yaml()函数，模型结构可以被再次加载（在不同的Python会话中从不同的脚本加载）。一旦加载，model_from_json()或者 model_from_yaml()函数可以被用作从结构中产生一个Keras模型。

```
from keras.models import model_from_json

# load architecture from JSON File
json_file = open( architecture.json , rt )
architecture =
json_file.read()
json_file.close()
# create model from architecture
model = model_from_json(architecture)
```

表 13.6 从文件中加载一个保存了的LSTM模型结构的例子

保存权重

Keras提供了一个函数在拟合模型上保存模型权重。save_weights()函数将会保存模型权重到HDF5格式文件。

```
...
# save weights to hdf5 file
model.save_weights( 'weights.h5' )
```

表 13.7 保存一个拟合了的LSTM模型权重到文件的例子

模型权重可以被再次加载（在不同脚本的不同的Python会话中使用load_weights()函数来加载模型对象）。这意味着你必须依据有了一个模型，要么重新创建，要么从加载的结构创建。

```
...
# save weights to hdf5 file
model.load_weights( 'weights.h5' )
```

表 13.8 从文件中加载一个保存了的LSTM模型权重的例子

例子

我们可以把它们组合成一个单独工作的例子。下面的演示合适于一个小型的数据集上的LSTM模型。模型结构采用JSON格式保存，模型权值以HDF5格式存储。如果你愿意，可以很容易地更改例子以保存为YAML格式的结构。然后从这些文件中加载模型并进行预测。

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from numpy import array
from keras.models import model_from_json

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# return training data
def get_train():
    seq = [[0.0, 0.1], [0.1, 0.2], [0.2, 0.3], [0.3, 0.4], [0.4, 0.5]]
    seq = array(seq)
    X, y = seq[:, 0], seq[:, 1]
    X = X.reshape((len(X), 1, 1))
    return X, y


# define model
model = Sequential()
model.add(LSTM(10, input_shape=(1,1)))
model.add(Dense(1, activation= 'linear' ))
# compile model
model.compile(loss= 'mse' , optimizer= 'adam' )
# fit model
X,y = get_train()
model.fit(X, y, epochs=300, shuffle=False, verbose=0)
# convert model architecture to JSON format
architecture = model.to_json()
# save architecture to JSON file
```

```
with open( 'architecture.json' , 'wt' ) as json_file:
    json_file.write(architecture)
    # save weights to hdf5 file
    model.save_weights( 'weights.h5' )
# snip...
# later, perhaps run from another script

# load architecture from JSON File
json_file = open( 'architecture.json' , 'rt' )
architecture = json_file.read()
json_file.close()
# create model from architecture
model = model_from_json(architecture)
# load weights from hdf5 file
model.load_weights( 'weights.h5' )
# make predictions
yhat = model.predict(X, verbose=0)
print(yhat)
```

表 13.9 保存一个拟合LSTM模型到分别的文件并稍后再次加载它的例子

运行这个例子保存模型结构到architecture.json文件并保存权重到weights.h5文件。然后从这些相同的文件加载模型。虽然模型加载在同一个脚本中演示，但是也可以在另外另外一个脚本文件的一个Python会话中轻松地运行该段脚本。

13.3 在新数据上做预测

在进行预测之前，对你的训练数据进行任何数据准备也必须应用于任何新的数据。例如，如果你的训练数据被标准化，那么所有你希望预测的新数据也必须被规范化。反过来，这意味着，用于规范数据（例如，最大值和最小值）的稀疏必须保存为模型化的一部分，并在需要预测时加载。

这适用于其他形式的数据准备，例如对数变换、标准化值和使系列平稳。原始数据被转换成其矢量化形式的特定方式也必须被复制。这包括序列的填充或截断，并且重要的是，将原始序列整形成样本的3D格式、时间步长和特征。

根据问题的框架，预测时的样本数可以指输入输出序列的数量，以便于输出预测。对于单个预测，它可能是1。想想所有在将原始数据转换成用于训练作为管道的模型的数据时所做的数据准备。这个管道也必须在任何时候进行预测。

13.3.2 预测值

预测是很容易的一部分。我们在第4张中介绍了基本API，但是我们将更加详细地复习它。它包括将准备好的输入数据（X）和调用Keras预测方法之一加载模型。记住，用于预测（X）的输入仅由预测所需的输入序列数组成，而不是所有先前的训练数据。在预测一个序列中的下一个值的情况下，输入序列将是1个样本，具有固定的时间步长和在定义和拟合模型时使用的特征。

例如，在输出层的激活函数的形状和尺度上的原始预测可以通过调用模型上的predict()函数来实现：

```
X = ...
model = ...
yhat = model.predict(X)
```

表 13.10 在新数据上用一个拟合的LSTM模型做预测的例子

预测一个类的索引可以通过在模型上调用predict_classes()函数实现。

```
X = ...
model = ...
yhat = model.predict_classes(X)
```

表 13.11 在新数据上用一个拟合的LSTM模型预测分类的例子

预测可能性可以通过在模型上调用predict_proba()函数来实现。

```
X = ...
model = ...
yhat = model.predict_proba(X)
```

表 13.12 在新数据上用一个拟合LSTM预测可能性的例子

13.4 扩展阅读

13.4.1 API

- How can I save a Keras model? in the Keras FAQ.
- Save and Load Keras API.

13.5 扩展

你想更深入地学习一个完成LSTM模型吗？本章节列出了本课程中的一些有挑战性的扩展：

- 列出一个或多个预测问题，在这里你想开发一个LSTM模型并保存它用于以后的预测中；
- 从书中的模型部分更新一个例子，以将模型保存到多个文件中，然后再次从另外一个脚本加载并进行预测；
- 从书中的模型部分更新一个示例，用一个分类输出来预测概率，并以图形方式呈现概率；
- 从书中的模型部分更新一个示例，使用一个分类输出来使用predict()函数，然后使用argmax()函数来将结果解释为类值；
- 从书中模型部分选择一个例子，通过聚焦模型结构（例如，层数、存储单元等）来调优性能。你可能需要增加问题的难度。

- 从书中模型部分选择一个例子，并通过关注模型行为（例如，周期数、训练例子、批次等）来调整性能。你可能需要增加这个问题的难度。

13.6 总结

在本章节中你学习到了在更新你的最终化的LSTM模型中充分利用新数据。特别地，你学习到了：

- 怎么样为你的工程开发一个最终的LSTM模型；
- 怎么样将LSTM模型保存到文件以便后面使用；
- 怎么样在新的数据上加载LSTM模型来做预测。

在下面的课程中，你将会学习到怎么样更新一个完成的LSTM模型来充分利用新数据。