4.0 前言

4.0.1 课程目标

本课程的目标是理解如何使用Python中的Keras深度学习库来定义、拟合和评价LSTM模型。完成这一课之后，你会知道：

- 如何定义一个LSTM模型，包括怎么样将你的数据变型成为所需要的3D输入；
- 如何拟合和评估你的LSTM模型并且将其用来预测新的数据；
- 如何对模型中的内部状态以及当其重置时进行细粒度的控制。

4.0.2 课程概览

本课程分为7个部分，它们是：

1. 定义模型；
2. 编译模型；
3. 拟合模型；
4. 评价模型；
5. 用模型做预测；

6. LSTM状态管理；
7. 准备数据的样例。

让我们开始吧！

4.1 定义模型

第一步是定义你的网络。Keras中的神经网络被定义为一系列的层。这些层的容器是`Sequential`类。第一步是创建`Sequential`类的实例。然后你可以创建你的层，并按照它们应该的连接的顺序添加它们。由存储单元组成的LSTM循环层被称为`LSTM()`。一个全连接层经常跟着一个LSTM层，用于输出一个预测的层被称为`Dense()`。

例如，我们可以定义一个具有两个存储单元的LSTM隐藏层，连接着带有一个神经元的一个全连接输出层，如下：

```
model = Sequential()
model.add(LSTM(2))
model.add(Dense(1))
```

表 4.1 定义一个LSTM模型的例子

但是我们也可以通过创建一个层的数组，并将其传递给`Sequential`类的构造函数来完成这一步。

```
layers = [LSTM(2), Dense(1)]
model = Sequential(layers)
```

表 4.2 第二个定义一个LSTM模型的例子

网络中的第一个隐藏层必须定义期望输出的数量，例如输入层的形状。输入必须是三维的，由样本、时间步长和特征组成。

- **样本**。它们是你数据中的行。一个样本可以是一个序列。
- **时间步长**。这些是对特征的过去观察值，例如滞后的变量。
- **特征**。这些是数据中的列。

假设你的数据被加载为了一个NumPy数组，你可以使用NumPy中的thereshape() 函数将1D或者2D的数据集转换为3D数据集。您可以调用您的NumPy数组中的NumPy函数，并传递给一个元组的维数给它，该维数就是你需要转成多少维的值。假设我们在NumPy数组中有两个列的输入数据（X），我们可以把这两个列当做两个时间步长来对待，将其改写如下：

```
data = data.reshape((data.shape[0], data.shape[1], 1))
```

表 4.4 变换有1个时间步长的NumPy数组的例子

如果你想让你的2D数据中的列成为一个时间步长的特征，你可以变形，如下：

```
data = data.reshape((data.shape[0], 1, data.shape[1]))
```

表 4.4 变换有1个时间步长的NumPy数组的例子

你可以指定输入形状参数，该参数期望是一个包含时间步长和特征数量的元祖。例如，如果我们有一个单变量序列的两个时间步长和一个特征，每行具有两个之后的观测值，则具体如下：

```
model = Sequential()
model.add(LSTM(5, input_shape=(2,1)))
model.add(Dense(1))
```

表 4.5 定义LSTM模型的输入的例子

样本的数量不必是指定的。模型假定一个或者多个样本，留给你定义的只有时间步长以及特征。本课的章节提供了为LSTM模型准备输入数据的附加示例。

把`Sequential`模型想象成为一个管道，一端输入原始数据，另外一端输出预测结果。在Keras中这是一个很有用的容器，它传统上是和一层相联系的，可以可以拆分并添加为一个单独的层。这样可以清楚地表明，它们在数据从输入到预测的转换中的作用。例如，激活函数，即变换一个从来自于一层的每一个神经元的求和的信号，可以被提取出来，并作为一个像层一样的对象，叫做Activition,添加到`Sequential`中。

```
model = Sequential()
model.add(LSTM(5, input_shape=(2,1)))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

表 4.6 在输出层带有sigmoid激活函数LSTM模型的例子

激活函数的选择对于输出层来说是最重要的，因为它将决定预测所采用的格式。例如，下面是一些常见的预测建模问题类型和结构以及在输出层中使用的标准激活函数：

- **回归**。线性激活函数，或者线性，神经元的数量和输出的数量相匹配。这是用于全连接层（Dense）层的默认激活函数。
- **二分类（2类）**。逻辑激活函数，或者sigmoid，一个神经元的输出层。
- **多分类（大于2类）**。Softmax激活函数，或者softmax，假设是one hot 编码输出模式每个类值一个输出神经元。

4.2 编译模型

一旦开发了我们的网络，我们就必须编译它。编译是一个有效的步骤。它将我们所定义的简单层序列转换成一系列高效的矩阵变换格式，以便在GPU或者CPU上执行，这取决于Keras是如何配置的。将编译看做是一个网络的预计算步骤。一个模型建立之后总是需要编译的。

编译需要多个参数被指定，尤其是那些需要调整来训练你网络的。具体来说，用于网络训练的优化函数以及用于评价网络的损失函数，它被优化函数最小化了。

例如，下面是编译指定梯度下降（sgd）和均方差（mse）的损失函数模型的情况，用于解决回归类型的问题。

```
model.compile(optimizer= 'sgd' , loss= 'mse' )
```

表 4.7 编译LSTM模型的例子

或者，在提供编译不走的参数之前创建和配置 `optimizer` 。

```
algorithm = SGD(lr=0.1, momentum=0.3)
model.compile(optimizer=algorithm, loss= 'mse' )
```

表 4.8 用SGD优化算法编译一个LSTM模型的例子

预测建模问题的类型可对所使用的损失函数进行约束。例如，下面是不同的预测模型类型的一些标准损失函数：

- **回归。**均方误差，或者 `mean squared error` ，简称 `mse` 。
- **二分类（2类）。**对数损失，也叫做交叉熵或者 `binary crossentropy` 。
- **多分类（大于2类）。**多分类的对数损失， `categorical crossentropy` 。

最常见的优化算法是经典的随机梯度下降算法，但是Keras还支持一套其他扩展的经典优化算法，这种算法表现很好、配置很少。由于它们通常的性能更好，也许最常用的优化算法是：

- **Stochastic Gradient Descent**， 或者sgd。
- **Adam**， 或者adam。
- **RMSprop**， 或者rmsprop。

最后，除了损失函数外，你也可以指定性能指标(Metrics)来收集拟合你模型时候的信息。总的来说，最有用的附加性能指标(Metrics)来收集的是分类问题的准确性（例如 ‘accuracy’ 或者简称 ‘acc’ ）。用来收集的性能指标(Metrics)可以通过性能指标(Metrics)数组中的名称或者损失函数的名字来指定。例如：

```
model.compile(optimizer= 'sgd' , loss= 'mean_squared_error' , metrics=[ 'accuracy' ])
```

表 4.9 用一个性能指标(Metrics)编译LSTM模型的例子

4.2 拟合模型

一旦网络被编译，它就可以拟合，这意味着适应训练数据集上的权重。拟合网络需要指定训练数据集，包括输入模式的性能指标(Metrics)X以及和输出类型匹配的数组 y。网络采用基于时间的反向传

播算法进行训练，并根据优化算法和损失函数特性对模型进行优化。

反向传播算法要求对网络进行训练数据集中的所有序列进行特定数量的周期或者暴露训练集中的所有序列。每个周期（epoch）可以被分为成组的输入-输出模式对，这被称为批次（batches）。这定义了网络在一个时代内更新权重之前所暴露的模式的数量。它也是一种有效的优化，确保在一段时间内没有太多的输入模式被加载到存储器中。

- **Epoch**。遍历训练数据集中的所有样本，并更新网络权重。LSTM可以训练及时、几百或者数千个周期（epoch）。
- **Batch**。通过训练数据集中的样本子集，然后更新网络权值。一个周期（epoch）是由一个或者多个批次（batch）组成的。

下面是batch size一些常用的配置：

- batch_size=1。在每个样本之后更新权重，并且该过程被称为随机梯度下降。
- batch_size=32。通常的值是32,64和128，调整来符合预期的效率和模型更新速率。如果批大小（batch size）不是一个周期（epoch）中一个样本数量的因素，那么在周期（epoch）的结尾，则在结束时运行剩余样本的一个额外的批大小（batch size）。
- batch_size=n。其中，n是训练数据集的样本数。权重在每个周期（epoch）中被更新，这个过程被称为批梯度下降（batch gradient descent）。

batch size为32的小批量梯度下降法（Mini-batch gradient descent）是LSTM的一个常见配置。拟合网络的一个例子如下：

```
model.fit(X, y, batch_size=32, epochs=100)
```

表 4.10 拟合LSTM模型的例子

一旦拟合，返回一个历史对象，该对象在训练期间提供对模型姓名的总结。当编译模型的时候，这包含损失和任何的额外的性能指标(Metrics)被指定，并在每个周期（epoch）被记录。这些性能指标(Metrics)可以被记录、绘制和分析，以了解网络在训练数据集上是过拟合（overfitting）还是欠拟合（underfittting）。

根据网络规模和训练数据的大小，训练可能会花费很长的时间，从秒到数天。默认情况下，进度条显示在每个epoch的命令行上。这可能会对你造成很多的噪音，或者会对你的环境造成问题，比如你在一个交互式的笔记本或者IDE中。你可以通过将 `verbose` 参数设置为2来将显示的信息量减少到仅损失的每个周期（epoch）。你可以通过设置 `verbose` 为0。例如：

```
history = model.fit(X, y, batch_size=10, epochs=100, verbose=0)
```

表 4.11 拟合一个LSTM模型并检索无 `verbose` 输出的历史的例子

4.4 评价模型

一旦网络被训练，它就可以被评估。网络可以对训练数据进行评估，但是这不会作为预测模型提供网络性能的有用指示，因为它以前见过所有这些数据。我们可以在一个单独的数据集上评估网络的性能，在测试期间看不见。在未来对于看不见的数据，这将提供网络性能的估计。

该模型评估了所有测试模式的损失，以及模型编译时的任何其他度量，如分类精度。返回一个评估的度量表。如，对于使用精度度量编译模型，我们可以在新的数据集上对其进行评估如下：

```
loss, accuracy = model.evaluate(X, y)
```

表 4.12 衡量一个LSTM模型的例子

与网络一样，提出`verbose`输出的概念是为了评价模型的进度。我们可以通过将`verbose`参数设置为0来关闭这个。

```
loss, accuracy = model.evaluate(X, y, verbose=0)
```

表 4.13 衡量一个没有`verbose`输出的LSTM的例子

4.5 预测模型

一旦对我们的拟合模型的性能感到满意，我们可以用它来预测新的数据。这就在带有新输入模式数组的模型上调用`predict()`函数一样简单。例如：

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

表 4.14 在拟合LSTM模型上预测的例子

预测将以网络的输出层提供的格式返回。在回归问题的情况下，这些预测可以直接和线性激活函数提供问题的格式一样。

对于一个二分类问题，预测可以使第一类的概率数组，它可以通过舍入转换为1或0。对于一个多分类问题，结果可能是一个概率数组的形式（假设是一个one hot编码输出变量），它可能需要使用NumPy的argmax()函数转换成单类输出预测。另外，对于分类问题，我们可以使用`predict_classes()`函数来自动地将不清晰的预测转换为清晰的整数类值。

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

表 4.15 拟合LSTM模型上预测类别的例子

随着对网络的拟合和评估，`verbose`被输出以便用于反映模型预测的进度。我们可以通过将`verbose`参数设置为0来关闭这个。

```
predictions = model.predict(X, verbose=0)
```

表 4.16 没有`verbose`输出时预测的例子

根据拟合LSTM模型来预测将会在第十三章中详细的讲述。

4.6 LSTM状态管理

每个LSTM存储单元都保持着积累的内部状态。这种内部状态可能需要在网络训练和预测时对序列问题进行详细的管理。默认情况下，网络中的所有LSTM存储单元的内部状态在每个批次之后被重置，例如，当网络权重被更新时。这意味着批次大小的配置在三个时期上施加了张力：

- 学习的正确性，或更新前处理多少个样本；
- 学习的速度，或者权重的更新频率；
- 内部状态，或内部状态重置频率。

通过顶一个LSTM层作为状态，Keras提供了将内部状态更新重置的很灵活的方式。这可以通过将LSTM层上的状态参数设置为true来实现。当使用状态LSTM层时，还必须通过设置输入形状参数来确定网络中输入形状的批大小（batch size），并且批处理大小必须是训练数据集中样本数量的一个因素。

批输入（batch input）形状参数需要一个定义为批处理大小、时间步长和特征的三维数组。

例如，我们可以定义一个状态LSTM，在训练数据集上训练100个样本，10个批次大小（batch size），1个特征的5个时间步长，如下：

```
model.add(LSTM(2, stateful=True, batch_input_shape=(10, 5, 1)))
```

表 4.17 定义一个带状态的LSTM层的例子

状态LSTM不会在每个批次结束时重置内部传状态。相反，您可以通过调用`reset_states()`函数对何时重置内部状态进行细粒度控制。例如，我们可能希望在每个周期（epoch）结束时重置内部状态，我们可以这样做：

```
for i in range(1000):
    model.fit(X, y, epochs=1, batch_input_shape=(10, 5, 1))
    model.reset_states()
```

表 4.18 一个状态LSTM手动迭代训练批次的例子

在进行预测时，也必须使用相同状态的LSTM中的相同批次大小（batch size）。

```
predictions = model.predict(X, batch_size=10)
```

例 4.19 用状态LSTM预测的例子

LSTM层的内部状态也在评估网络和进行预测时积累。因此，如果使用的是状态LSTM，则必须在验证数据集或者预测之后对网络进行重置状态。

在默认情况下，一个周期（epoch）的状态被洗牌。在使用神经网络的多层感知机进行工作的时候这是一个很好的做法。如果您尝试在样本间保存状态，那么训练数据集中的样本顺序可能是重要的并且必须保留。这可以通过设置 `shuffle` 参数为False来完成，例如：

```
for i in range(1000):
    model.fit(X, y, epochs=1, shuffle=False, batch_input_shape=(10, 5, 1))
    model.reset_states()
```

表 4.20 拟合一个状态LSTM时使样本不shuffle的例子

为了使这个更具体，下面是管理状态的3个常见的例子：

- 在每个序列结束的时候进行预测，并且序列是独立的。状态应该在每个序列之后重置，通过将批次大小设置为1。
- 长序列被分割成多个子序列（每个样本具有许多的时间步长）。状态应该在网络暴露于整个序列之后通过LSTM状态化，关闭子序列的洗牌（shuffling），并在每个周期（epoch）之后重置状态之后被重置。
- 一个非常长的子序列被分成多个子序列（每个样本有许多时间步长）。训练质量比长期内部状态更重要，使用128个样本的批次大小（batch size），然后更新网络权重和状态重置。

我鼓励你头脑风暴你的序列预测问题和网络配置的许多不同的框架，测试和选择那些在预测误差方面最有希望的模型。

4.7 准备数据的例子

很难理解如何准备您的序列数据以输入到LSTM模型。通常，围绕如何定义LSTM模型的输入层存在混淆。关于如何将您的序列数据转换为一维或二维的数字矩阵到LSTM输入层所需的3D格式也存在混淆。在这一节中，您将通过两个示例来修改序列数据，并将输入层改为LSTM模型。

4.7.1 单个输入样本的LSTM的例子

考虑一个具有多个时间步长和一个特征的序列。例如，这可以是10个值的序列：

```
0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
```

表 4.21 序列的例子

我们可以将这个序列的数字定义为NumPy数组。

```
from numpy import array
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
```

表 4.22 将一个序列定义为NumPy数组的例子

我们可以使用NumPy的`reshape()`函数来将这个一维数组变成三维数组，每个时间步长上有1个样本、10个时间步长和一个特征。在数组上调用时，`reshape()`函数采用一个参数，它是数组的新的形状。我们不能传递任何一组数字，变换必须要均匀地重新排列数组中的数据。

```
data = data.reshape((1, 10, 1))
```

表 4.23 变换一个数组的例子

一旦变换了，我们可以输出数组新的形状。

```
print(data.shape)
```

表 4.24 打印序列新形状的例子

将所有的都放在一起，完整的例子展示如下：

```
from numpy import array
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
data = data.reshape((1, 10, 1))
print(data.shape)
print(data)
```

表 4.25 变换一个样本的例子

运行例子打印单个样本的3D形状。

```
(1, 10, 1)
[[[ 0.1]
  [ 0.2]
  [ 0.3]
  [ 0.4]
  [ 0.5]
  [ 0.6]
  [ 0.7]
  [ 0.8]
  [ 0.9]
  [ 1. ]]]
```

表 4.26 单个样本变型输出的例子

这个数据通过设置`inpput_shape=(10,1)`现在已经准备好被用于输入（X）到LSTM模型中。

```
model = Sequential()
model.add(LSTM(32, input_shape=(10, 1)))
...
```

表 4.27 定义LSTM模型输入层的例子

4.7.2 多输入特征的LSTM例子

考虑到你的模型可能有多个并行序列作为输入的情况。例如，下面是有10个值的两个并行的序列：

```
series 1: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 series 2: 1.0, 0.9, 0.8, 0.7,
0.6, 0.5, 0.4, 0.3, 0.2, 0.1
```

表 4.28 并行序列的例子

我们可以将这些数据定义为2列10行的数据：

```
from numpy import array
data = array([ [0.1, 1.0], [0.2, 0.9], [0.3, 0.8], [0.4, 0.7], [0.5, 0.6], [0.6, 0.5],
[0.7, 0.4], [0.8, 0.3], [0.9, 0.2], [1.0, 0.1]])
```

表 4.29 将并行序列定义为一个NumPy数组的例子

这个数据可以被化为具有10个时间步长和2个特征的1个样本，它可以变形为3D数组如下：

```
data = data.reshape(1, 10, 2)
```

表 4.30 变形一个序列的例子

将所有这些放在一起，完整的例子显示如下：

```
from numpy import array
data = array([ [0.1, 1.0], [0.2, 0.9], [0.3, 0.8], [0.4, 0.7], [0.5, 0.6], [0.6, 0.5],
[0.7, 0.4], [0.8, 0.3], [0.9, 0.2], [1.0, 0.1]])
data = data.reshape(1, 10, 2)
print(data.shape)
```

表 4.31 变形并行序列的例子

运行例子，打印单个样本新的3D形状如下：

```
(1, 10, 2)
```

表 4.32 变换并行序列输出的例子

这个数据现在已经通过设置 `input_shape=(10,2)` 准备好被用作输入（X）到LSTM模型中。

```
model = Sequential()
model.add(LSTM(32, input_shape=(10, 2)))
...
```

表 4.33 定义LSTM模型输入层的例子

4.7.3 LSTM输入的建议

本章节列出了一些帮助您准备LSTM输入数据的提示：

- LSTM输入层必须是3D的；
- 3个输入维度的含义是：样本、时间步长和特征。
- LSTM输入层在输入隐藏层上由输入形状参数决定。
- 输入形状参数采用两个值的元组，以减少时间步长和特征的数量。
- 假设样本的数量是1个或者更多。
- NumPy数组中的 `reshape()` 函数可以用来将1D或者2D数据变换为3D的。
- `reshape()` 函数将元组作为新的形状的参数。

4.8 扩展阅读

本章节提供一些用于扩展阅读的资料。

4.8.1 Keras APIs

- Keras API for Sequential Models.
- Keras API for LSTM Layers.
- Keras API for optimization algorithms.
- Keras API for loss functions.

4.8.2 其它APIs

- NumPy reshape() API.
- NumPy argmax() API.

4.9 扩展

你想深入了解Keras中LSTMs的生命周期吗？这个章节列出了本课程中一些具有挑战性的扩展。

- 列出5个序列预测的问题并重点说明怎么样将数据分割成为样本、时间步长以及特征；
- 列出5个序列预测问题并单独指出每个输出层的激活函数；
- 研究Keras矩阵以及损失函数，并列出5个可以用于回归序列预测问题和分类序列预测问题的5个矩阵；
- 研究Keras历史对象并编写示例代码来创建Matplotlib的线图，该图是从拟合一个LSTM模型捕获的矩阵；
- 列出5个序列预测问题，以及如何使得网络更好地管理每个节点的内部状态。

在网上发布你的扩展并与我分享链接。我很想知道你是怎么样想的！

4.10 总结

在本课程中，你发现了使用Keras库的LSTM循环神经网络的5步生命周期。特别地，你学到了：

- 怎么样定义一个LSTM模型，包括怎么样去将你的数据变型为所需要的3D输入；
- 怎么样定义和评价你的LSTM模型，以及怎么样使用它在新数据上做预测；
- 怎么样对模型中的内部状态以及当其重置时进行细粒度的控制。

在下面的章节中，我们将会发现4个主要类型的序列预测模型以及怎么样在Keras中运用。