3.0 前言

3.0.1 课程目标

本课的目的是教您如何准备用于LSTM模型的序列预测数据。完成这一课后，你会知道：

- 如何对数字数据进行缩放，以及如何转换分类数据。
- 如何填充和截断不同长度的输入序列。
- 如何将输入序列转换成有监督的学习问题。

3.0.2 课程概览

本课程被划分为4个部分，它们是：

1. 准备数值数据；

2. 准备类别数据；
3. 准备不同长度的序列；
4. 基于监督学习的序列预测。

让我们开始吧！

3.1 准备数值数据

在训练神经网络时，可能需要对序列问题的数据进行缩放，例如长短时记忆循环神经网络。当一个网络对一个具有一定范围的值的数据（例如10s到100s的数量）时，大的输入可以减缓网络的学习和收敛，并且在某些情况下阻止网络有效学习你的问题。

您可能需要考虑的系列有两种类型的缩放：归一化和标准化。这些都可以使用Python中的scikit-learn学习机器学习库来实现。

3.1.1 归一化序列数据

归一化是从原始范围的数据重新缩放，使得所有的值都在0到1的范围之内。规范化要求您指导或能够准确地最小和最大可观测值。您可以从可用的数据中估计这些值。如果您可以从可用的数据中估计这些值。如果您的序列是呈增长趋势的或者下降趋势的，估计这些期望值可能是困难的，并且规范化可能不是解决您问题的最好办法。

如果要缩放的值超过最小值和最大值的边界，则所得到的值的范围不在0到1的范围之内。在进行预测之前，你可以先检查下这些观测值，或者将它们从数据集中删除，或者将它们限制在一个预先定义的最大值或者最小值的范围内。可以使用scikit-learn对象的MinMaxScaler来归一化数据集。使用MinMaxScaler缩放技术以及其他缩放技术的的一个好的例子如下：

- **使用可用的训练数据拟合缩放器。**对于归一化，这意味着训练数据将被用来估计最小和最大的可观测值。这可以通过调用 `fit()` 函数来完成。
- **将尺度运用于训练数据。**这意味着可以使用归一化数据来训练模型。这可以通过调用 `transform()` 函数来完成。
- **将尺度应用于后面的数据。**这意味着你可以在未来准备你想要用于预测的新的数据。

如果需要的话，变换可以反过来。这将有助于将预测转化为原始尺度的报告或者绘图。这个可以通过调用反向的 `transform()` 函数来实现。下面是对10个单位的人为归一化的序列的例子。缩放器对象需要提供数据作为行和列的矩阵。加载的时间序列数据作为Pandas Series。

```
from pandas import Series
from sklearn.preprocessing import MinMaxScaler
# define contrived series
data = [10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
series = Series(data)
print(series)
# prepare data for normalization
values = series.values
values = values.reshape((len(values), 1))
```

```
# train the normalization
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(values)
print('Min: %f, Max: %f' % (scaler.data_min_, scaler.data_max_))
# normalize the dataset and print
normalized = scaler.transform(values)
print(normalized)
# inverse transform and print
inversed = scaler.inverse_transform(normalized)
print(inversed)
```

列表 3.1 正则化序列的例子

运行该示例并打印序列，打印的最小和最大值估计序列，打印同样归一化了的序列，然后打印利用反向变换回来的原来的值。我们可以看到，最小值和最大值的数据分别为10和100。

```
0     10.0
1     20.0
2     30.0
3     40.0
4     50.0
5     60.0
6     70.0
7     80.0
8     90.0
9    100.0
dtype: float64
Min: 10.000000, Max: 100.000000
[[ 0.        ]
 [ 0.11111111]
 [ 0.22222222]
 [ 0.33333333]
 [ 0.44444444]
 [ 0.55555556]
 [ 0.66666667]
 [ 0.77777778]
 [ 0.88888889]
 [ 1.        ]]
[[  10.]
 [  20.]
 [  30.]
 [  40.]
 [  50.]
 [  60.]
 [  70.]
 [  80.]
 [  90.]
 [ 100.]]
```

表3.2 正则化一个序列输出的例子

3.1.2 标准化序列数据

标准化一个数据集涉及重新调整值的分布，以便使得观测值的平均值为0，标准差为1。这可以被认为是数据都减去了一个平均数或者中心化数据。

与归一化一样，标准化可能是有用的，甚至在某些机器学习算法中，当您的数据具有不同尺度的输入值时也是需要的。标准化假设您的观察值符合高斯分布（贝尔曲线），具有良好的均值和标准差。不满足这个期望，你仍然可以标准化你的时间序列数据，但是可能不会得到可靠的结果。

标准化需要你知道或者能够准确地估计观测值的均值和标准差。你可以从训练数据中估计这些值。数据集的均值和标准差估计对新数据的鲁棒性比最小和最大值更高。可以使用scikit-learn对象的StandardScaler来标准化你的数据集。

```
from pandas import Series
from sklearn.preprocessing import StandardScaler
from math import sqrt
# define contrived series
data = [1.0, 5.5, 9.0, 2.6, 8.8, 3.0, 4.1, 7.9, 6.3]
series = Series(data)
print(series) # prepare data for normalization
values = series.values
values = values.reshape((len(values), 1))
# train the normalization
scaler = StandardScaler()
scaler = scaler.fit(values)
print('Mean: %f, StandardDeviation: %f' % (scaler.mean_, sqrt(scaler.var_)))
# normalize the dataset and print
standardized = scaler.transform(values)
print(standardized)
# inverse transform and print
inversed = scaler.inverse_transform(standardized)
print(inversed)
```

列表 3.3 标准化一个序列的例子

运行打印的序列的例子，打印从序列估计得均值和标准差，打印标准化之后的值，然后以原始的比例打印值。我们可以看出，估计得均值和标准差分别是5.3和2.7。

```
0    1.0
1    5.5
2    9.0
3    2.6
4    8.8
5    3.0
6    4.1
7    7.9
8    6.3
dtype: float64
Mean: 5.355556, StandardDeviation: 2.712568
[[-1.60569456]
 [ 0.05325007]
 [ 1.34354035]
 [-1.01584758]
 [ 1.26980948]
 [-0.86838584]
```

```
  [-0.46286604]
  [ 0.93802055]
  [ 0.34817357]]
 [[ 1. ]
  [ 5.5]
  [ 9. ]
  [ 2.6]
  [ 8.8]
  [ 3. ]
  [ 4.1]
  [ 7.9]
  [ 6.3]]
```

列表 3.4：序列标准化输出的例子

3.1.3 缩放时考虑的实际问题

在缩放序列数据的时候有一些实际问题需要考虑：

- **估计系数。**如果你可以从训练数据中估计系数（归一化的最大值或者最小值，以及标准化的均值和标准差）。检查这些第一次估计，并使用领域知识或者领域专家来帮助改善这些估计，以便他们在后面能够有效地纠正所有的数据。
- **保存系数。**你需要在后面以完全相同的方式重新缩放新数据。保存用于文件的系数，然后在预测时需要缩放新的数据。
- **数据分析**：使用数据分析来帮助你更好的理解数据，例如，一个简单的直方图可以帮助你快速获得数量分布的感觉，以便来看看标准化是否有意义。
- **缩放每个序列**：如果你的问题有多个序列，则将每个变量视为单独的变量，然后分别对它们进行缩放。这里，缩放是指缩放过程的选择，例如归一化或标准化。
- **在正确的时间缩放。**在正确的时间应用任何缩放变化是很重要的。例如，如果你有一系列非平稳的单位，那么在使数据静止后，可以适当地缩放。将该序列转换成有监督的学习问题是不合适的，因为每个列都将被不同地处理，这将是不正确的。
- **如果有疑问则缩放。**你可能需要重新调整输入和输出变量。如果有疑问，最起码归一化你的数据。

3.2 准备类别数据

类别数据是包含的是标签值而不是数值的变量。可能值的数目通常限定于一个固定的集合。类别变量通常被称为名词性的。一些例子包括：

- 宠物的种类的值：狗和猫；
- 颜色值的变量：红、绿和蓝；
- 序列变量的值：第一、第二和第三。

每一个只代表一个不同的类别。文本中的词可以被认为是类别数据，文本中每一个词可以被认为是一个不同的类别。文本输入或者输出的序列预测问题可以被认为是类别数据。

某些类别可能彼此具有天然的联系，例如自然排序。上面的位置变量确实有一个值的自然排序。这种类型的变量称为序数变量。在使用LSTM时，必须将分类数据转换为数字。

3.2.1 怎么样把类别数据转化为数值数据

这包含两个步骤：

- 数字编码；
- 独热编码(One-Hot Encoding)；

数值编码 作为一个步骤，每个唯一的类别值被分配了一个整数值。例如，红色是1，绿色是2，以及蓝色是3。这被称为变迁编码或者整数编码，并且很容易可逆。对于一些变量，这可能是足够的。

整数值彼此之间具有自然的有序关系，机器学习算法可以理解和利用这种关系。例如，上面的位置变量之类的序数变量将是一个很好的例子，其中标签编码将是充分的。

独热编码(One-Hot Encoding) 对于没有这样顺序关系的分类变量，整数编码是不够的。事实上，使用这种编码并允许模型假设类别之间的自然排序可能会导致性能不佳或者意外结果（类别间的预测）。

在这种情况下，可以将一个独热编码应用于整数表示。这是删除整数编码变量并为每个唯一整数值添加新的二进制变量的地方。在颜色可变的例子中，有三个类别，因此需要三个二进制变量。当该颜色存在的时候该处放1值，不存在的时候放0值，例如：

```
red, green, blue
1,   0,     0
0,   1,     0
0,   0,     1
```

表3.5： 独热编码颜色的例子

3.2.2 用scikit-learn进行one hot编码

在本例子中，我们将假定你有以下的三个标签的输出序列：cold，warm， hot。10个时间步长的示例序列可以是：

```
cold, cold, warm, cold, hot, hot, warm, cold, warm, hot
```

表3.6 天气类别序列的例子

这将需要一个整数编码，例如1,2,3。这之后是对一个具有3个值的二进制向量的整数的一个one hot编码，例如[1,0,0]。该序列提供了关于序列中每个可能值的至少一个示例。因此，我们可以使用自动方法来定义标签到数值以及数值到二进制向量的映射。

在这个例子中，我们将使用scikit-learn库中的编码器。具体来说，`LabelEncoder`创建一个标签的整数编码，`OneHotEncoder`为整数编码的值创建一个one hot encoding。下面列出了完整的例子。

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
# define example
data = ['cold', 'cold', 'warm', 'cold', 'hot', 'hot', 'warm', 'cold', 'warm', 'hot']
values = array(data)
print(values)
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)
# binary encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
# invert first example
inverted = label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])
print(inverted)
```

表3.7 序列的one hot encoding的例子

运行示例，首先打印标签序列，这之后是标签的整数编码，然后是一个one hot编码。训练数据包含所有可能的示例集合，因此我们可以依靠整数和一个one hot编码变换来创建标签搭配编码的完整映射。

默认情况下，`OneHotEncoder`类将返回一个更有效的稀疏编码。这可能不适用于某些应用程序，例如与Keras深度学习库一起使用。在这种情况下，我们通过设置`sparse = False`参数来禁用稀疏返回类型。如果我们在这个由3个值组成的one hot编码中收到一个预测，我们就可以很容易地将变换反转回原来的标签。

首先，我们使用NumPy的argmax()函数来定位具有最大值的列的索引。然后，可以将其喂给`LabelEncoder`来进行逆变换计算回文本标签。在示例结束时，将第一个one hot编码实例逆变换为标签值'cold'。再次注意，输入为了可读性被格式化了。

```
['cold' 'cold' 'warm' 'cold' 'hot' 'hot' 'warm' 'cold' 'warm' 'hot']
[0 0 2 0 1 1 2 0 2 1]
[[ 1.  0.  0.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 0.  1.  0.]]
['cold']
```

表3.87 one hot编码天气输出的例子

3.3 准备可变长度的序列

深度学习库假定对数据进行矢量化表示。在可变长序列预测问题的情况下，这需要对数据进行变换，以便使得每个序列具有相同的长度。对于你所选择的深度学习算法来说，矢量化允许代码以批量的方式执行矩形运算。

3.3.1 序列插补

Keras深度学习库中的插补函数 `sequences()` 可以用来插补可变长的序列。默认的填充值为0，尽管可以通过参数值指定首选值来改变此值，但是这是适用于大多数应用程序的。例如，应用于序列开始或者结束的填充，称为前序填充或者后序填充，可以由填充参数指定，如下所示。

前序列插补 前序列插补是是默认项（padding='pre'）。下面的例子显示了用0来前插补由3个值组成的序列。

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [ [1, 2, 3, 4], [1, 2, 3], [1] ]
# pad sequence
padded = pad_sequences(sequences)
print(padded)
```

表3.9 前向插补的例子

运行例子，打印3个由零值插补的序列。

```
[[1 2 3 4]
 [0 1 2 3]
 [0 0 0 1]]
```

后序插补 插补也可以应用于序列的结尾，在某些问题领域可能更合理一些。后序插补可以通过指定插补参数来设置。

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [ [1, 2, 3, 4], [1, 2, 3], [1] ]
# pad sequence
padded = pad_sequences(sequences, padding= 'post' )
print(padded)
```

表 3.11 后序插补的例子

运行例子，打印同样的以0值插补的序列。

```
[[1 2 3 4]
 [1 2 3 0]
 [1 0 0 0]]
```

表 3.12 后序插补输出的例子

3.3.2 序列截断

序列的长度也可以修剪成期望的长度。序列的期望长度可以用`maxlen`参数指定为多个时间步长。有两种方法可以截断序列：通过从开始或者结尾移除时间步长。

前序截断

默认截断方法是从序列开始去除时间步长，这被称为前序截断。下面的示例是将序列截断为期望长度为2的序列。

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [ [1, 2, 3, 4], [1, 2, 3], [1] ]
# truncate sequence
truncated= pad_sequences(sequences, maxlen=2)
print(truncated)
```

表 3.13： 前序截断的例子

运行该示例，从第一个序列中删除前两个时间步长，从第二个序列删除前面的一个时间步长，并插补最后一个序列。

```
[[3 4]
 [2 3]
 [0 1]]
```

表 3.14 前序截断输出的例子

后序截断

序列也可以通过从末尾去除时间步长来修整。对于某些领域问题，这种方法可能更为理想。通过将缺省参数从'pre'变为'post'，就可以实现如下的后序截断：

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [ [1, 2, 3, 4], [1, 2, 3], [1] ]
# truncate sequence
truncated= pad_sequences(sequences, maxlen=2, truncating= 'post')
print(truncated)
```

表 3.15： 后序截断的例子

运行该示例，从第一个序列中删除后面两个时间步长，从第二个序列中删除最后一个时间步长，第三个时间步长再次填充。

```
[[1 2]
 [1 2]
 [0 1]]
```

表 3.16： 后序截断输出的例子

对于何时进行插补何时进行截断不同长度的输入序列没有经验法则。例如，在情感分析中截断非常长的文本可能是很有意义的，或者对短文本进行填充也可能是有意义的，并且让模型去学习忽略或者输入零值覆盖来确保没有数据丢失。

我建议为你的序列预测问题测试一系列不同的表达，并记录上哪些结果很好的模型。

3.4 监督学习的序列预测

序列预测问题必须重新构建为监督学习问。也就是说，数据必须转化成输入和输出的对。

3.4.1 序列VS监督学习

在我们开始之前，让我们花一点时间来好好理解下原始输入序列的形式和监督学习数据。考虑到一个。这可以看做是有序值的列表或者列，例如：

```
0
1
2
3
4
5
6
7
8
9
```

表 3.17 序列的例子

监督学习问题由输入（X）和输出（y）组成，使得算法可以可以学习如何从输入模式中预测输出模式。例如：

```
X, y
1, 2
2, 3
```

```
3, 4
4, 5
5, 6
6, 7
7, 8
8, 9
```

表 3.18 监督学习问题输入输出对例子

这将表示序列的1-lag变型，使得必须给定序列的先前时间步长来预测当前是的时间步长。

3.4.2 Pandas shift()函数

将时间序列数据转换成有监督学习问题的一个关键是Pandas的shift()函数。给定一个`DataFrame`，`shift()`函数可以用来创建列的拷贝，这些列被向前推（含有NaN值的行被添加到前面）或者向后拉（含有NaN值的列被添加到末尾）。

这是创建之后观测的列所需要的行为，也是监督学习格式中时间序列数据集上的预测观测列。让我们来看看实际中的`shift()`函数的一些例子。我们可以将模拟时间序列数据集作为10个数字序列，在这种情况下`DataFrame`的单个列如下：

```
from pandas import DataFrame
# define the sequence
df = DataFrame()
df['t'] = [x for x in range(10)]
print(df)
```

表 3.20 创建一个序列并打印的例子

运行样例打印每个观察值带有索引行的时间序列数据。

```
   t
0  0
1  1
2  2
3  3
4  4
5  5
6  6
7  7
8  8
9  9
```

表 3.21 创建序列的输出例子

我们可以通过在顶部插入一个新的行来将所有观测值向下移动一个时间步长。因为新行没有数据，所以我们可以使用NaN来表示没有数据。`shift()`函数可以为我们这样做，并且我们可以在我们原始列旁边插入这个移位列。

```
from pandas import DataFrame
# define the sequence
df = DataFrame()
df['t'] = [x for x in range(10)]
# shift forward
df['t-1'] = df['t'].shift(1)
print(df)
```

表 3.21 向前转换序列的例子

运行这个例子给了我们数据集中的两列：用原始的观测值和一个新的位移列。我们可以看到，把一个时间序列移向一个原始的监督学习问题，尽管X和Y的顺序不对。忽略行标签的列。因为有NaN值，所以第一行必须被丢弃。第二行显示了第二列（输入或者X）中的0的输入值和第一列（输出或y）中的1的值。

```
   t  t-1
0  0  NaN
1  1  0.0
2  2  1.0
3  3  2.0
4  4  3.0
5  5  4.0
6  6  5.0
7  7  6.0
8  8  7.0
9  9  8.0
```

表 3.22 向前转换序列输出的例子

我们可以看到，如果我们可以通过2次,3次以及更多的位移来重复这个过程，我们可以创建长的输入序列（X），它可以用来预测输出值（y）。

移位运算符也可以接受负整数值。这是通过在末尾插入新行来拉动观察结果的。下面是一个例子：

```
from pandas import DataFrame
# define the sequence
df = DataFrame()
df['t'] = [x for x in range(10)]
# shift backward
df['t+1'] = df['t'].shift(-1)
print(df)
```

表 3.23 从后位移序列的例子

运行示例，显示一个新的列，其中的NaN值为最后一个值。我们可以看到，预测列可以作为输入（X），第二列作为输出值（y）。也就是说，输入值的0可以用来预测1的输出值。

```
   t  t+1
0  0  1.0
```

```
1  1  2.0
2  2  3.0
3  3  4.0
4  4  5.0
5  5  6.0
6  6  7.0
7  7  8.0
8  8  9.0
9  9  NaN
```

表 3.24 从后位移位移输出的例子

从技术上讲，在时间序列预测术语中，当前时间（t）和未来时间（t+1, t+n）是预测时间，过去的观察值（t-1, t-n）用于进行预测。我们可以看到将shit()函数用于监督学习中在一个带有输入输出序列的时间序列中创建新的DataFrame有多少正面影响以及多少负面影响。

这不仅允许经典的 `X->y` 预测，也允许 `X->Y`，其中输入和输出都可以是序列。此外，`shift()` 函数也适用于所谓多变量时间序列问题。这里时间序列不是只有一组观察值，我们可以有多个（例如温度和压力）。时间序列中所有的变量可以镶嵌或者向后移位以创建多变量输入和输出序列。

3.5 扩展阅读

这个章节提供了资源以便进一步的阅读。

3.5.1 数值缩放的APIs

- MinMaxScaler API in scikit-learn.
- StandardScaler API in scikit-learn.
- Should I normalize/standardize/rescale the data? Neural Nets FAQ.

3.5.2 类别编码的APIs

- LabelEncoder API in scikit-learn.
- OneHotEncoder API in scikit-learn.
- NumPy argmax() API.

3.5.3 变化序列长度的APIs

- pad sequences() API in Keras.

3.5.4 监督学习的APIs

- shift() function API in Pandas.

3.6 扩展

你想更深入地学习为LSTMs准备数据吗？这个章节列出了本课程中一些有挑战性的扩展：

- 用一个段落总结什么时候需要规范化数字数据、什么时候需要标准化，以及当你有疑问的时候应该怎么样去做；
- 编写一个函数将ASCII文本进行独热编码(One-Hot Encoding)为类别数据，另外一个函数用来解码编码的格式；
- 列出3个来自填充输入序列和3的序列预测问题的例子，它们可以从截断输入序列中得到；
- 给定一个单变量时间预测问题，数年里每小时进行观察，并给出前一课中截断BPTT的理解，描述3种可以将序列转化为监督学习的方式；
- 开发一个Python函数来自动将序列转化为监督学习问题，并且输出的时间步长的值可以被指定为参数。

在网上张贴你的扩展，并且将其分享给我。我很想知道你的想法。

3.7 总结

在本课程中，你可以发现怎么样准备用于LSTM循环神经网络的序列数据。特别地，你学习到了：

- 怎么样是对数字数据进行缩放，以及如何转换分类数据；
- 如何填充和阶段不同长度的输入序列；
- 如何将输入序列转换成为有监督的学习问题；
- 如何将输入序列转换成有监督的学习问题。 接下来，你将在Keras库中发现LSTM模型的生命周期。