8.0 前言

8.0.1 课程目标

本课程的目标是使用卷积神经网络作为前段开发一个LSTM模型。本课程结束之后，你将会学习到：

- CNN LSTM的原始结构以及它适合什么样类型的问题；
- 怎么样在Keras中应用CNN LSTM结构；
- 怎么样为移动广场视频预测问题开发一个CNN LSTM模型。

8.0.2 课程概览

本课程分为7个部分，它们是：

- CNN LSTM；
- 移动广场视频预测问题；
- 定义和编译模型；

- 拟合模型；
- 用模型进行预测；
- 完成例子。

让我们开始吧！

8.1 CNN LSTM

8.1.1 结构

CNN LSTM结构涉及在输入数据中使用卷积神经网络（CNN）层做特征提取并结合LSTM来支持序列预测。CNN LSTMs开发用来可视化序列预测问题和从图像序列生成文本描述的应用（例如：视频）。特别地，问题包括：

- **活动识别**。对一个序列的图片所显示的活动生成一个文本的描述。
- **图像描述**。对单个图片生成一个文本的描述。
- **视频描述**。对一个序列的图片生成一个文本的描述。

> CNN LSTMs是这样一类模型，它在空间和时间上都很深，并具有适用于各种输入任务和输出的视觉任务的灵活性。
>
> — Long-term Recurrent Convolutional Networks for Visual Recognition and Description, 2015

这种架构最初被称为长期卷积神经网络（Long-term Recurrent Convolutional Network）或者LRCN模型。尽管我们将使用更通用的名为CNNLSTM来指代本课中使用的CNN作为前段的LSTM模型。该体系结构用于生成图像的文本描述的任务。关键是使用CNN，它在具有挑战性的图像分类问题上被预训练，该任务被重新用作字幕生成问题的特征提取器。

> ...使用CNN作为图像的编码器是很自然的，通过对图像分类任何进行预训练，并将最后隐藏层作为输入生成RNN解码器。
>
> — Show and Tell: A Neural Image Caption Generator, 2015

这种体系结构也被用于语音识别和自然语言处理问题，其中CNNs被用作语音和文本输入数据上的LSTM的特征提取器。该体系结构合适于以下的问题：

- 它们的输入中具有空间结构，例如如下中的2D结构或像素或句子、段落或者文档中的一维结构；
- 在其输入中具有时间结构，或者视频中的图像顺序或者文本中的单词或者需要生产具有时间结构的输出，例如文本描述中的单词。

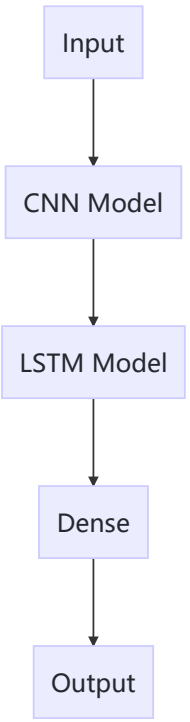

图 8.1 CNN LSTM结构

8.1.2 实现

我们定义了一个CNN LSTM模型来在Keras中共同训练。CNN LSTM可以通过在前端添加CNN层然后紧接着LSTM作为全连接层输出来被定义。

这种体系结构可以被看做是两个子模型：CNN模型做特征提取，LSTM模型帮助教师跨时间步长的特征。在假设输入是图像的一系列的2D输入情况下，让我们来看看这两个子模型的背景：

CNN模型

作为刷新，我们可定义一个2D卷积网络，包括Conv2D和MaxPooling2D层，它们有序的排列在所需深度的堆栈中。Conv2D将解释图像的快照（例如：小方块），池化层将巩固或抽象解释。

例如，下面的片段期望以1个通道（例如：黑和白）读取10X10像素图像。Conv2D将读取2X2快照中的图像并输出一个新的10X10的图像解释。MaxPooling2D将池化解释为2X2的块，将输出减少到5X5合并。Flatten层将采用单个5X5映射，并将其转换成25个元素的矢量，以准备用于其他层处理，例如用于预测输出的Dense。

```
cnn = Sequential()
cnn.add(Conv2D(1, (2,2), activation= 'relu' , padding= 'same' , input_shape=(10,10,1)))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Flatten())
```

表 8.1 CNN LSTM模型的部分CNN的例子

这在图像识别和其他的机器学习任务中是有效果的。

LSTM模型

上面的CNN网络只能处理单个图像，将其输入像素转换为内部矩阵或者矢量表示。我们需要在多个图像删重复该操作，并允许LSTM通过输入图像的内部矢量表示的序列使用BPTT来建立内部状态和更新权重。

CNN可以使用现有的预训练模型如VGG进行图像特征提取。CNN可能不被训练，我们可能希望通过LSTM的错误从多个输入图像反向传播到CNN模型来训练它。在这种情况下，概念上都有一个单一的CNN模型和一个LSTM序列，每个时间步长都有一个模型。我们希望将CNN模型应用于每个输入图像，并将每个输入图像的输出作为单个时间步长传递给LSTM。

我们可以通过将整个CNN输入模型（一层或多层）封装在一个TimeDistributed层中来实现这一点。该层实现了多次应用相同层或层的期望结果。在这种情况下，将其多次应用于多个输入时间步长，并依次向LSTM模型提供一系列图像解释或者图像特征。

```
model.add(TimeDistributed(...)) model.add(LSTM(...)) model.add(Dense(...))
```

表 8.2 CNN LSTM模型的LSTM部分的例子

我们有这个模型的两个部分了，让我们将它们放在一起吧！

CNN LSTM模型

我们可以在Keras中定义一个CNN LSTM模型的层，将它们包含在TimeDistributed层中，然后定义LSTM以及输出层。我们有两个方法来定义模型，这两种定义的方法是等效的，只是在品位上有点区别。你可以首先定义CNN模型，然后通过将CNN层的整个序列包括在TimeDistributed层中的方法将其添加到LSTM模型中，例如：

```
# define CNN model
cnn = Sequential()
cnn.add(Conv2D(...))
cnn.add(MaxPooling2D(...))
cnn.add(Flatten())
# define CNN LSTM model
model = Sequential()
model.add(TimeDistributed(cnn, ...))
model.add(LSTM(..))
model.add(Dense(...))
```

表 8.3 CNN LSTM模型的两个部分的例子

另一种，也许更容易阅读的方法是将CNN模型中的每一层封装在TimeDistributed中，然后将其添加到主模型中。

```
model = Sequential()
model.add(TimeDistributed(Conv2D(...))
model.add(TimeDistributed(MaxPooling2D(...)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(...))
model.add(Dense(...))
```

表 8.4 CNN LSTM模型一部分的例子

第二种方法的好处是所有的层都出现在模型的摘要中，因此这个是优选的。你可以选择你喜欢的方法。

8.2 移动广场视频预测问题

移动广场视频预测问题是为了证明CNN LSTM。这个问题设计生产一系列的帧。在每副图像中，从左到右或者从右到左画出一条线。每帧显示一行像素的扩展。任务是根据线在帧序列中向左或者向右移动为模型进行分类。技术上来说，这个问题是一个具有多对一预测模型的序列分类问题。

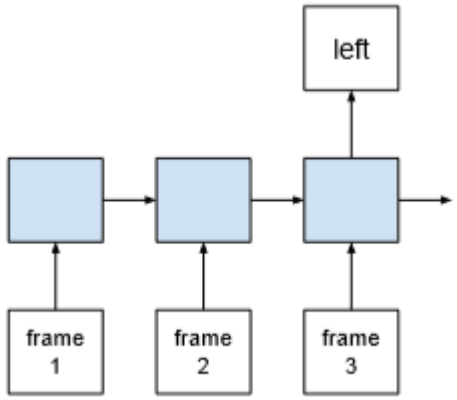

图 8.2 采用many-to-many预测模型的移动广场视频预测问题

测试问题可以分为下面的两个步骤：

1. 图像初始化；
2. 添加步骤；
3. 实例生成器。

8.2.1 图像初始化

我们可以通过定义一个填满0值的2D的NumPy数组来开始。我们可以使得图像对称，在这种情况下，10个像素高10个像素宽。

```
from numpy import zeros
frame = zeros((10,10))
```

图 8.5 生成空的方块图像的例子

下面，我们可以为行的第一步选择行。我们将使用randint()函数来选择一个0到9之间的统一随机整数。

```
from random import randint
step = randint(0, 10-1)
```

表 8.6 选择一个步长的例子

我们现在可以选择是否在图像上画左或右线。我们将使用random()函数来决定。如果是正确的，我们将从左边或第0栏开始，如果离开，我们将从右边开始，或者从第9栏开始。

```
from random import random
right = 1 if random() < 0.5 else 0
col = 0 if right else size-1
```

表 8.7 决定向左或者向右移动的例子

我们可以线的开始。

```
frame[step, col] = 1
```

表 8.8 标记线的开始的例子

8.2.2 添加步长

现在我们需要一个过程来增加步骤。下一步必须是前一步的函数。我们将榆树它在下一列（左或者右），并在同一行中，上面的行或者下面的行。我们将进一步根据图像的边界限制运动。我们可以使用上面相同的randint()函数来选择下一个步骤，并将我们的运动约束价钱在上和下值上。上次选择的不长值存储在最后一步变量中。

```
lower = max(0, last_step-1)
upper = min(10-1, last_step+1)
step = randint(lower, upper)
```

表 8.9 选择下一个步长的例子

下面，我们可以复制最后一幅图像并标记下一列的新位置。

```
column = i if right else size-1-i
frame = last_frame.copy()
frame[step, column] = 1
```

表 8.10 将步骤标记为新帧的示例

根据选定的方向，可以重复这个过程知道达到第一列或者最后一列。

8.2.3 示例生成器

我们可以在两个小函数中铺货所有上述行为。build frames()函数使用一个参数来定义图像的大小，并返回一系列图像，以及该行是向右移动（1）还是向左移动（0）。这个函数调用另一个函数next frame()来创建当先在图像上移动时的每一个后续帧。

为了使问题具体化，我们可以绘制一个序列。我们将产生一个小序列与每个图像5X5像素和5帧并绘制帧。

```
from numpy import zeros
from random import randint
from random import random
from matplotlib import pyplot


# generate the next frame in the sequence
def next_frame(last_step, last_frame, column):
    # define the scope of the next step
    lower = max(0, last_step-1)
    upper = min(last_frame.shape[0]-1, last_step+1)
    # choose the row index for the next step
    step = randint(lower, upper)
    # copy the prior frame
    frame = last_frame.copy()
    # add the new step
    frame[step, column] = 1
    return frame, step


# generate a sequence of frames of a dot moving across an image
def build_frames(size):
    frames = list()
    # create the first frame
    frame = zeros((size,size))
    step = randint(0, size-1)
    # decide if we are heading left or right
    right = 1 if random() < 0.5 else 0
    col = 0 if right else size-1
    frame[step, col] = 1
    frames.append(frame)
    # create all remaining frames
    for i in range(1, size):
        col = i if right else size-1-i
        frame, step = next_frame(step, frame, col)
        frames.append(frame)
    return frames, right


# generate sequence of frames
size = 5
```

```
frames, right = build_frames(size)
# plot all feames
pyplot.figure()
for i in range(size):
    # create a grayscale subplot for each frame
    pyplot.subplot(1, size, i+1)
    pyplot.imshow(frames[i], cmap= 'Greys')
    # turn of the scale to make it cleaer
    ax = pyplot.gca()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
# show the plot
pyplot.show()
```

表 8.11 生成帧序列的示例

运行示例生成一个随机序列并并排绘制帧。你可以看到，线从左到右绕着图像摆动，每一个时间步长一个像素。

图 8.3 线在图像中移动的的一系列帧的例子

8.2.4 准备模型输入

最后，我们将准备一个函数来生成具有正确形状的多个序列，以准备和评估LSTM模型。下面给出一个名为generate_examples()的函数，它将要生成的图像的大小和将要生成的序列的数量作为参数。

生成并存储每个序列。重要的是，模型的输入序列必须调整大小以适应2D CNN。通常情况下可以是：

```
[width, height, channels]
```

表 8.12 2D CNN模型预期输入的例子

在我们的情况中，对于对称的黑白图像来说，它是[size, size, 1]。这是不足够的，因为我们也有多个图像，然后多个序列的图像。因此，对模型的输入必须变型为：

```
[samples, timesteps, width, height, channels]
```

表 8.13 模型期望输入形状的例子

或者，在我们的函数中：

```
[n_patterns, size, size, size, 1]
```

表 8.14 使用来自问题定义的术语作为期望输入形状的例子

这个生成随机视频的新的函数被列出如下：

```
# generate multiple sequences of frames and reshape for network input
def generate_examples(size, n_patterns):
    X, y = list(), list()
    for _ in range(n_patterns):
        frames, right = build_frames(size)
        X.append(frames)
        y.append(right)
        # resize as [samples, timesteps, width, height, channels]
    X = array(X).reshape(n_patterns, size, size, size, 1)
    y = array(y).reshape(n_patterns, 1)
    return X, y
```

表 8.15 生成和变型模型的帧序列的例子

下面，让我们定义并编译模型。

8.3 定义和编译模型

我们可以定义一个CNNLSTM来拟合模型。生成图像的大小决定了问题会多具有挑战性。我们将通过配置图像到50X50像素或者一共2500个二进制值来使得问题不那么具有挑战。

```
# configure problem
size = 50
```

表 8.16 配置问题的例子

我们将会定义一个CNN模型中的每一层都包裹在一个单独的TimeDistributed层中的模型。这是为了确保模型概要清楚地说明网络是如何连接在一起的。我们将定义一个Conv2D层作为输出层，它具有两个filter和一个2X2的核来扫描输入图像。使用2个filter和常规情况下使用小的核是基于实验的经验的。Conv2D将会输出2个49X49像素的输入印象。

卷积层经常直接跟着一个池化层。这里我们使用一个pool大小为2X2的MaxPooling2D池化层，这将有效地减少前一层每一个输出的大小，然后输出2个24X24的映射。

池化层之后跟着的是Flatten层，它用于将来自于MaxPooling的2D层的[24，24，2] 3D输出输出转换成一维具有1152个元素的向量。CNN模型是一个特征提取的模型。希望的是，Flatten层的向量输出是一个被压缩的和/或更显著的表示，而不是原始像素值。

接下来，我们可以定义模型的LSTM模型。我们使用一个具有50个存储单元的LSTM层，这是在一次次尝试和错误之后配置的。在整个CNN模型中使用TimeDistribted wrapper意味着LSTM将会看到50个时间步长，每一个时间步长表示一个1152个元素的向量作为输入。

这是一个二分类的问题，所以我们将使用一个具有一个单一神经元和sigmoid激活函数的Dense输出。该模型被编译以将Adam用作梯度下降来最小化log损失（binary crossentropy），同时二分类的准确度将会被显示出来。下面提供完整的代码列表：

```
# define the model
model = Sequential()
model.add(TimeDistributed(Conv2D(2, (2,2), activation= 'relu'), input_shape=
(None,size,size,1)))
model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(50))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

表 8.17 定义并编译CNN LSTM模型的例子

运行例子，打印编译模型的摘要。我们可以确定每个层输出的期望形状。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_1 (TimeDist (None, None, 49, 49, 2)   10
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 24, 24, 2)   0
_________________________________________________________________
time_distributed_3 (TimeDist (None, None, 1152)        0
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                240600
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 240,661
Trainable params: 240,661
Non-trainable params: 0
_________________________________________________________________
None
```

表 8.18 定义CNN LSTM模型的输出的例子

8.4 拟合模型

我们现在准备好了在随机生成的问题的例子上拟合模型。上面所定义的generate examples()函数准备了一些特定数量的随机序列，这些随机序列我们可以存在内存中，并用于有效地拟合模型。随机生成的示例的数量是训练周期（epoch）的数量的代理，因为我们更喜欢将模型训练到特定的问题实例上，而不是一次又一次地重复同一个随机实例。

在这里，我们将在5000个随机生成序列的单个周期（epoch）上训练模型。理想情况下，LSTM的内部状态将在每个序列的末尾被重置。我们可以通过将批大小（batch size）设置为1来实现这一点。我们将权衡模型的保真度和计算效率，并将批大小（batch size）设置为32。

```
# fit model
X, y = generate_examples(size, 5000)
```

```
model.fit(X, y, batch_size=32, epochs=1)
```

表 8.19 拟合编译了的CNN LSTM模型的例子

运行例子将会显示执行命令行时的进度条，它显示每个批次（batch）结束时的损失和准确度。如果您在IDE或笔记本中运行该示例，可以通过设置verbose=0来打开进度条。

```
5000/5000 [==============================] - 37s - loss: 0.1507 - acc: 0.9208
```

表 8.20 拟合CNN LSTM模型输出的例子

我们可以通过亚牛股不同数量的样本、周期（epoch）和批大小（batch size）来实验。你可以开发一个学习能力更好并且整体训练更少的模型吗?

8.5 评价模型

现在模型被拟合了，我们可以在一个新的随机序列上估计模型的学习能力。这里，我们可以生成100个新的随机序列，并估计模型的准确度。

```
# evaluate model
X, y = generate_examples(size, 100)
loss, acc = model.evaluate(X, y, verbose=0)
print('loss: %f, acc: %f' % (loss, acc*100))
```

表 8.21 衡量拟合了的CNN LSTM模型的例子

运行例子打印拟合模型的损失和精确度。这里，我们可以看到模型达到了100%的准确度。你的结果可能不一样，但是你如果没有看到100%的准确度，尝试着多运行例子几次。

```
loss: 0.001120, acc: 100.000000
```

8.6 用模型进行预测

为了完整性，我们可以开发一个模型来在新的序列上做出预测。这里我们生成了一个新的单个随机序列，并预测线将会向左或者是向右移动。

```
# prediction on new data
X, y = generate_examples(size, 1)
yhat = model.predict_classes(X, verbose=0)
expected = "Right" if y[0]==1 else "Left"
predicted = "Right" if yhat[0]==1 else "Left"
print('Expected: %s, Predicted: %s' % (expected, predicted))
```

表 8.23 使用拟合了的CNN LSTM模型进行预测的例子

运行例子，打印出期望的编码以及预测的值。

```
Expected: Right, Predicted: Right
```

表 8.24 用拟合了的CNN LSTM模型做出预测的输出的例子

8.7 完整例子

为了完整性，下面提供了完整的代码列表供您参考。

```
from random import random
from random import randint
from numpy import array
from numpy import zeros
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import TimeDistributed


# generate the next frame in the sequence
def next_frame(last_step, last_frame, column):
    # define the scope of the next step
    lower = max(0, last_step-1)
    upper = min(last_frame.shape[0]-1, last_step+1)
    # choose the row index for the next step
    step = randint(lower, upper)
    # copy the prior frame
    frame = last_frame.copy()
    # add the new step
    frame[step, column] = 1
    return frame, step


# generate a sequence of frames of a dot moving across an image
def build_frames(size):
    frames = list()
    # create the first frame
    frame = zeros((size,size))
    step = randint(0, size-1)
    # decide if we are heading left or right
    right = 1 if random() < 0.5 else 0
    col = 0 if right else size-1
    frame[step, col] = 1
    frames.append(frame)
    # create all remaining frames
    for i in range(1, size):
```

```
        col = i if right else size-1-i
        frame, step = next_frame(step, frame, col)
        frames.append(frame)
    return frames, right


# generate multiple sequences of frames and reshape for network input
def generate_examples(size, n_patterns):
    X, y = list(), list()
    for _ in range(n_patterns):
        frames, right = build_frames(size)
        X.append(frames)
        y.append(right)
        # resize as [samples, timesteps, width, height, channels]
    X = array(X).reshape(n_patterns, size, size, size, 1)
    y = array(y).reshape(n_patterns, 1)
    return X, y


# configure problem
size = 50
# define the model
model = Sequential()
model.add(TimeDistributed(Conv2D(2, (2,2), activation= 'relu'), input_shape=
(None,size,size,1)))
model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(50))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())

# fit model
X, y = generate_examples(size, 5000)
model.fit(X, y, batch_size=32, epochs=1)

# evaluate model
X, y = generate_examples(size, 100)
loss, acc = model.evaluate(X, y, verbose=0)
print('loss: %f, acc: %f' % (loss, acc*100))

# prediction on new data
X, y = generate_examples(size, 1)
yhat = model.predict_classes(X, verbose=0)
expected = "Right" if y[0]==1 else "Left"
predicted = "Right" if yhat[0]==1 else "Left"
print('Expected: %s, Predicted: %s' % (expected, predicted))
```

表 8.25 移动广场预测问题中CNN LSTM模型的完整例子

8.8 扩展阅读

本章节提供了进行扩展阅读的一些资源。

8.8.1 CNN LSTM论文

- Long-term Recurrent Convolutional Networks for Visual Recognition and Description, 2015.
- Show and Tell: A Neural Image Caption Generator, 2015.
- Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks, 2015.
- Character-Aware Neural Language Models, 2015.
- Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting, 2015.](https://arxiv.org/abs/1506.04214)

8.8.2 Keras API

- Conv2D Keras API. https://keras.io/layers/convolutional/#conv2d MaxPooling2D Keras API.
- Flatten Keras API.
- TimeDistributed Keras API.

8.9 扩展

你想深度地了解CNN LSTM吗？本章节列出了本课程红一些具有挑战性的扩展：

- 列出可以应用CNN LSTM模型的5个机器视觉和5个自然语言处理问题；
- 调整CNN层的数量以及
-

把你的扩展放到网上，并将链接分享给我。我很想知道你是怎么样想的！

8.10 总结

在本课程中，你发现了怎么样去开发一个CNN LSTM模型。特别地，你学到了：

- CNN LSTM的原始结构以及它适合什么样类型的问题；
- 怎么样在Keras中应用CNN LSTM结构；
- 怎么样为移动广场视频预测问题开发一个CNN LSTM模型。

在下一课中，你将会学习到怎么样开发一个Encoder-Decoder LSTM模型。