11.0 前言

11.0.1 课程目标

本课程的目标是学习怎么样为开发LSTMs。完成本课程之后，你将会学习到：

- LSTMs怎么样可以被用于生成模型；
- 怎么样把形状绘制作为序列生成问题；
- 怎么开发一个LSTM模型来生成形状。

11.0.2 课程概览

本课程被分为7个部分。它们是：

1. Generative LSTM；
2. 形状生成问题；
3. 定义和编译模型；

4. 拟合模型；
5. 用模型做预测；
6. 评估模型；
7. 完成例子。

让我们开始吧！

11.1 Generative LSTM

11.1.1 生成模型

LSTM可以用作生成模型。给定序列数据的大语料库，例如文本文档，可以设计LSTM模型来学习语料库的一般结构属性，并且当给定种子输入时，可以生成代表原始语料库的新序列。

在自然语言处理领域中，开发一种概括语料库模型的问题称为语言建模。语言模型可以在词语级工作，并学习文档中单词之间的概率关系，以便准确地完成句子并生成完全新的句子。在其最具有挑战性的语言模型中，在字符级工作，从字符序列中学习，并一次生成一个字符的新序列。

> 字符级语言模型的目标是来预测序列中的下一个字符。
>
> — Generating Text with Recurrent Neural Networks, 2011.

尽管有更多的挑战，一个字符级模型添加的灵活性允许新的字符被生成，增加标点符号，并且生成文本数据中可能存在的任何其他结构。

> ...在一个时刻预测一个字符是从序列生成的角度来说是更有趣的，因为它允许网络发明新的单词和字符串。
>
> — Generating Sequences With Recurrent Neural Networks, 2013.

语言建模是目前为止研究最多的Generative LSTM应用，可能是因为使用标准数据集可以对模型性能进行量化和比较。这种方法以及被用来在一组有趣的语言建模问题上生成文本，例如：

- 生成维基百科文章（包括标记）；
- 从生成像莎士比亚这样的伟大的作者的作品片段；
- 生成技术手稿（包括标记）；
- 生成计算机源代码；
- 生成文章标题。

结果的质量各不相同，例如，标记或源代码可能需要人工干预来渲染或编译。然而，结果令人印象深刻。该方法也已应用于不同的领域，其中现在的序列信息的大语料库是可用的，并且可以一步一步生成新的序列，例如：

- 手写体生成；
- 音乐生成；

- 语音生成；

11.1.2 结构和实现

一个Generative LSTM不是真正的体系结构，它更多地是关于LSTM预测模型学习和使用模型的角度的改变。我们可以想象地使用任何LSTM体系结构作为生成模型。在这种情况下，我们使用一个简单的Vanilla LSTM。

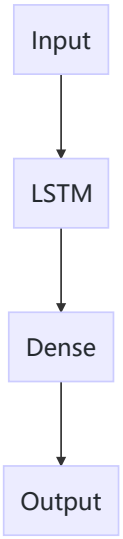

图 1.1 Vanilla LSTM情况下，Generative LSTTM结构

在字符级语言模型的情况下，所有可能字符的字母都是固定的。一个one hot编码既用于学习输入序列，又用于预测输出序列。一个一对一的模型用于预测每个输入时间步长的一个步骤。这意味着输入序列可能需要专门的处理，以便被矢量化或格式化，以便有意识地训练有监督的模型。例如，给定序列：

```
“hello world”
```

表 11.1 字符序列的例子

一个数据集可能需要被构建成例如：

```
'h' => 'e'
'e' => 'l'
'l' => 'l'
...
```

表 11.2 如one-to-one模型的字符序列的例子

这可以作为一个时间步长样本的数据集来呈现，这可以相当限制网络（例如，没有BPTT）。或者，它可以被矢量化为many-to-one个时间步长模型的固定长度的输入序列，例如：

```
[ 'h' , 'e' , 'l' ] => 'l'
[ 'e' , 'l' , 'l' ] => 'o'
[ 'l' , 'l' , 'o' ] => ' '
...
```

表 11.3 many-to-one模型的字符序列的例子

或者，一个one-to-many时间步长模型的一个固定长度的输出序列。

```
'h' => [ 'e' , 'l' , 'l' ]
'e' => [ 'l' , 'l' , 'o' ]
'l' => [ 'l' , 'o' , ' ']
...
```

表 11.4 one-to-many模型的字符序列的例子

或者这些方法的一些变化。注意，在进行预测时，需要相同的矢量化表示，这意味着预测的字符将需要作为后续样本的输入。这在实现上可能是相当笨拙的。网络的内部形状可能需要仔细管理，可能在输入序列中的选择位置（例如段落、页面或章节结束）重置，而不是在每个输入序列的末尾重置。

11.2 形状生成问题

我们可以将产生随机形状的问题分解为序列生成问题。我们可以把一个矩形化成顺时针方向的点序列，在二维空间中有4个点：

- **Bottom Left**(BL):[0, 0]
- **Bottom Right**(BR):[1, 0]
- **Top Right**(TR):[1, 1]
- **Top Left**(TL):[0, 1]

每个坐标可以作为一个时间步长，x轴和y轴中的每一个代表单独的特征。从[0, 0]开始，任务是绘制具有一致宽度和高度的矩形的其余4个点。我们将把这个问题看做是一个坐标生成问题，例如one-to-one序列预测问题。给定坐标，预测下一个坐标。然后给定在最后一步预测的坐标，预测下一个坐标等托。

```
[0,0] => [x1, y1]
[x1,y1] => [x2, y2]
[x2,y2] => [x3, y3]
```

表 11.5 作为一个one-to-one模型做坐标预测的例子

 图 11.2 基于one-to-one预测模型的形状生成问题

我们可以通过生成随机矩阵来训练模型，并使模型预测后续坐标。从[0, 0]预测第一坐标是不可能的，并且从宽度预测高度也是不可能的，但是我们相信，只要重复暴露于整个矩形，模型就可以学习如何绘制具有一致宽度和高度的新矩形。在Python中实现这一过程涉及三个步骤：

1. 生成随机矩形；
2. 绘制矩形；
3. 矩形到序列；

11.2.1 生成随机矩形

我们可以使用random()函数来生成0到1之间的随机数。给定一个矩形，我们可以使用random()函数来定义矩形的宽度和高度。

```
width, height = random(), random()
```

表 11.6 生成随机宽度和高度的例子

然后，我们可以使用宽度和高度来定义矩形的4个点，从顺时针方向从x=0，y=0开始。

```
[0.0, 0.0]
[width, 0.0]
[width, height]
[0.0, height]
```

表 11.7 随机矩形的坐标序列的例子

下面的函数，叫做random_rectangle()，产生了一个随机矩阵并返回一个二维点的列表。

```
from random import random

# generate a rectangle with random width and height
def random_rectangle():
    width, height = random(), random()
    points = list()
    # bottom left
    points.append([0.0, 0.0])
    #  bottom right
    points.append([width, 0.0])
    # top right
    points.append([width, height])
    # top left
    points.append([0.0, height])
    return points

rect = random_rectangle()
print(rect)
```

表 11.8 生成随机矩阵的例子

运行示例创建一个随机矩阵并打印出坐标。你生成的特定的随机矩阵将有所不同。

```
[[0.0, 0.0], [0.7541960995182183, 0.0], [0.7541960995182183, 0.9971807822173515], [0.0,
0.9971807822173515]]
```

表 11.9 生成随机矩阵输出的例子

11.2.2 显示矩阵

我们选择一个二维形状生成，主要是因为我们可以可视化结果。因此，我们需要一种容易地绘制矩形的方法，它要么随机生成，要么由模型预测。我们可以使用Matplotlib库来绘制坐标路径。

这涉及为形状定义一个路径，包括坐标列表和路径移动的形状。坐标列表将是我们随机生成的坐标列表，其中添加第一个坐标到列表的末尾以闭合多边形。路径是一系列的运动，如MOVETO指令启动序列，LINETO指令连接点，以及CLOSEPOLY指令来闭合多边形。

```
# close the rectangle path
rect.append(rect[0])
# define path
codes = [Path.MOVETO, Path.LINETO, Path.LINETO, Path.LINETO, Path.CLOSEPOLY]
path = Path(rect, codes)
```

表 11.10 将矩形左边转换成形状的例子

我们可以从路径定义一个PathPatch并将其直接在Matplotlib画图上画出来。

```
axis = pyplot.gca()
patch = PathPatch(path)
# add shape to plot
axis.add_patch(patch)
```

表 11.11 显示一个形状的例子

下面的函数叫做plot_rectangle()将一个4个矩形点的列表作为输入，并绘制并显示该图。轴的边界被调整以确保形状在0-1的范围内很好地被保持。

```
from matplotlib import pyplot
from matplotlib.patches import PathPatch
from matplotlib.path import Path

# plot a rectangle
def plot_rectangle(rect):
    # close the rectangle path
    rect.append(rect[0])
    # define path
```

```
    codes = [Path.MOVETO, Path.LINETO, Path.LINETO, Path.LINETO, Path.CLOSEPOLY]
    path = Path(rect, codes)
    axis = pyplot.gca()
    patch = PathPatch(path)
    # add shape to plot
    axis.add_patch(patch)
    axis.set_xlim(-0.1,1.1)
    axis.set_ylim(-0.1,1.1)
    pyplot.show()
```

表 11.12 显示一个矩形的函数

下面的例子展示了显示随机生成矩形的例子。

```
from random import random
from matplotlib import pyplot
from matplotlib.patches import PathPatch
from matplotlib.path import Path

# generate a rectangle with random width and height
def random_rectangle():
    width, height = random(), random()
    points = list()
    # bottom left
    points.append([0.0, 0.0])
    # bottom right
    points.append([width, 0.0])
    # top right
    points.append([width, height])
    # top left
    points.append([0.0, height])
    return points

# plot a rectangle
def plot_rectangle(rect):
    # close the rectangle path
    rect.append(rect[0])
    # define path
    codes = [Path.MOVETO, Path.LINETO, Path.LINETO, Path.LINETO, Path.CLOSEPOLY]
    path = Path(rect, codes)
    axis = pyplot.gca()
    patch = PathPatch(path)
    # add shape to plot
    axis.add_patch(patch)
    axis.set_xlim(-0.1,1.1)
    axis.set_ylim(-0.1,1.1)
    pyplot.show()

rect = random_rectangle()
plot_rectangle(rect)
```

表 11.13 生成和显示一个矩形的例子

运行例子生成一个随机矩形并将其绘制到屏幕上。生成的特定矩形将在每次代码运行时发生变化。

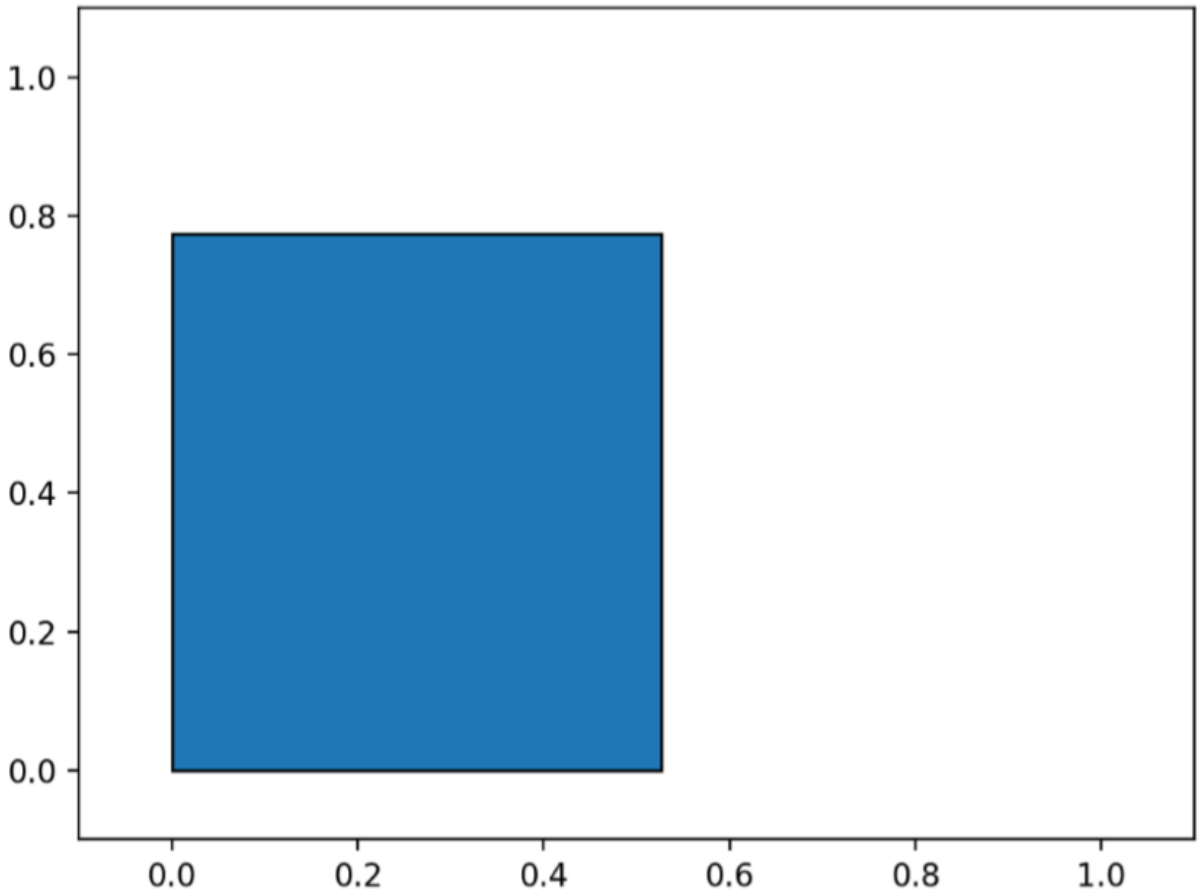

图 11.3 显示随机矩阵

11.2.3 矩阵到序列

最后，我们需要将2D坐标的序列从随机生成的矩形转换成我们可以用来训练LSTM模型的东西。给定矩形的4个点的序列：

```
[BL, BR, TR, TL]
```

表 11.14 矩形的点序列的例子

我们可以生成具有1个时间步长和一个特征的3个例子，如下：

```
[BL] => [BR]
[BR] => [TR]
[TR] => [TL]
```

表 11.15 one-to-one模型的一个序列的点的例子

下面的函数叫做get_samples()将会生成一个新的随机矩形并将其转换成一个数据集的3个例子：

```
# generate input and output sequences for one random rectangle
def get_samples():
    # generate rectangle
    rect = random_rectangle()
    X, y = list(), list()
    # create input output pairs for each coordinate
    for i in range(1, len(rect)):
        X.append(rect[i-1])
        y.append(rect[i])
    # convert input sequence shape to have 1 time step and 2 features
    X, y = array(X), array(y)
    X = X.reshape((X.shape[0], 1, 2))
    return X, y
```

表 11.16 生成一个随机矩形并返回一个序列点的函数

下面的例子显示了这个函数。

```
from random import random
from numpy import array

# generate a rectangle with random width and height
def random_rectangle():
    width, height = random(), random()
    points = list()
    # bottom left
    points.append([0.0, 0.0])
    # bottom right
    points.append([width, 0.0])
    # top right
    points.append([width, height])
    # top left
    points.append([0.0, height])
    return points

# generate input and output sequences for one random rectangle
def get_samples():
    # generate rectangle
    rect = random_rectangle()
    X, y = list(), list()
    # create input output pairs for each coordinate
    for i in range(1, len(rect)):
        X.append(rect[i-1])
        y.append(rect[i])
    # convert input sequence shape to have 1 time step and 2 features
    X, y = array(X), array(y)
    X = X.reshape((X.shape[0], 1, 2))
    return X, y

X, y = get_samples()
for i in range(X.shape[0]):
    print(X[i][0], '=>', y[i])
```

表 11.17 生成一个矩形并将其准备为LSTM模型的例子

运行这个例子显示了输入和输出坐标的每个例子。每次运行示例时，特定坐标将变化。

```
[ 0.  0.] => [ 0.44680225  0.        ]
[ 0.44680225  0.        ] => [ 0.44680225  0.87292771]
[ 0.44680225  0.87292771] => [ 0.          0.87292771]
```

表 11.18 生成一个矩形并将其准备给LSTM模型的输出的例子

11.3 定义和编译模型

现在我们已经准备好了一个LSTM模型来解决形状预测的问题。该模型将预期1个时间步长输入，每一个具有2个特征的x坐标轴和y坐标轴。我们将在LSTM隐藏层中使用10个存储单元。大容量则不需要管这个问题，10个单元的设定是从一次次的尝试和错误中来的。

```
model = Sequential()
model.add(LSTM(10, input_shape=(1, 2)))
```

表 11.19 添加输入层的例子

网络的输出是由x和y值组成的单个坐标。我们将使用具有2个神经元和线性激活函数的Dense层。

```
model.add(Dense(2, activation= 'linear' ))
```

表 11.20 添加输出层的例子

该模型最小化了平均绝对误差（mae）损失函数，并利用Adam优化算法对网络权值进行了计算。下面列出了完成的模型的定义。

```
# define model
model = Sequential()
model.add(LSTM(10, input_shape=(1, 2)))
model.add(Dense(2, activation= linear ))
model.compile(loss= 'mae' , optimizer= adam )
print(model.summary())
```

表 11.21 定义和编译Generative LSTM模型的例子

运行例子打印模型结构的总结。我们可以看到模型确实是很小的。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 10)                520
```

```
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 22
=================================================================
Total params: 542
Trainable params: 542
Non-trainable params: 0
_________________________________________________________________
None
```

表 11.22 从定义和编译Generative LSTM输出的例子

11.4 拟合模型

该模型可以通过一次生成一个序列并使用它们来更新模型权重来拟合。每个序列由3个样本组成，在该模型的末尾，模型权重将被更新，并且每个LSTM单元的内部状态将被重置。这允许模型的内存集中在每个二进制序列的特定输入值上。生成的样本数代表训练周期epoch的代理；在这里，我们将使用25000个随机生成的矩形点序列。这种配置是在经过一次次的实验和反复尝试后发现的。样本的顺序很重要，因此样本的混洗被关闭。

```
# fit model
for i in range(25000):
    X, y = get_samples()
    model.fit(X, y, epochs=1, verbose=2, shuffle=False)
```

表 11.23 拟合一个编译了的Generative LSTM的例子

在最后一个周期的时候拟合模型打印损失。

```
Epoch 1/1
 - 0s - loss: 0.3404
Epoch 1/1
 - 0s - loss: 0.2813
Epoch 1/1
 - 0s - loss: 0.1133
Epoch 1/1
 - 0s - loss: 0.1091
Epoch 1/1
 - 0s - loss: 0.0847
Epoch 1/1
 - 0s - loss: 0.1837
Epoch 1/1
 - 0s - loss: 0.2099
Epoch 1/1
 - 0s - loss: 0.2659
 Epoch 1/1
 - 0s - loss: 0.1562
Epoch 1/1
 - 0s - loss: 0.1360
Epoch 1/1
 - 0s - loss: 0.2088
```

```
Epoch 1/1
 - 0s - loss: 0.3155
Epoch 1/1
 - 0s - loss: 0.1470
```

表 11.24 拟合一个编译了的Generative LSTM输出的例子

11.5 用模型进行预测

一旦模型拟合，我们就可以使用它来生成新的矩形。我们可以用通过将矩形的起点定义为[0, 0]，以此作为输入来获得下一个预测坐标。这是矩形的宽度。

```
# use [0,0] to seed the generation process
last = array([0.0,0.0]).reshape((1, 1, 2))
# predict the next coordinate
yhat = model.predict(last, verbose=0)
```

表 11.25 预测矩形宽度的例子

然后，我们可以使用预测坐标作为输入来预测下一点。这将定义矩形的高度。

```
# use this output as input for the next prediction
last = yhat.reshape((1, 1, 2))
# predict the next coordinate
yhat = model.predict(last, verbose=0)
```

表 11.26 预测矩形高度的例子

然后，该过程再重复一次，以产生剩余的坐标，该坐标预期与已经在第三点中定义的宽度和高度保持一致。下面的函数名generate_rectangle()包裹了这个函数，并使用拟合模型生成一个新的矩形，然后返回点列表。

```
# use a fit LSTM model to generate a new rectangle from scratch
def generate_rectangle(model):
    rect = list()
    # use [0,0] to seed the generation process
    last = array([0.0,0.0]).reshape((1, 1, 2))
    rect.append([[y for y in x] for x in last[0]][0])
    # generate the remaining 3 coordinates
    for i in range(3):
        # predict the next coordinate
        yhat = model.predict(last, verbose=0)
        # use this output as input for the next prediction
        last = yhat.reshape((1, 1, 2))
        # store coordinate
        rect.append([[y for y in x] for x in last[0]][0])
    return rect
```

表 11.27 函数从零开始生成一个矩形

我们可以使用这个函数来进行预测，然后使用先前定义的函数plot_rectangle()来显示结果。

```
# generate new shapes from scratch
rect = generate_rectangle(model)
plot_rectangle(rect)
```

表 11.28 从零开始生成一个矩形的函数

运行例子生成一个矩形并将其显示在屏幕上。我们可以看到，矩形在宽度和高度的比例保持一致上确实做了很合理的工作。生成的特定矩形将在每次代码运行时发生变化。

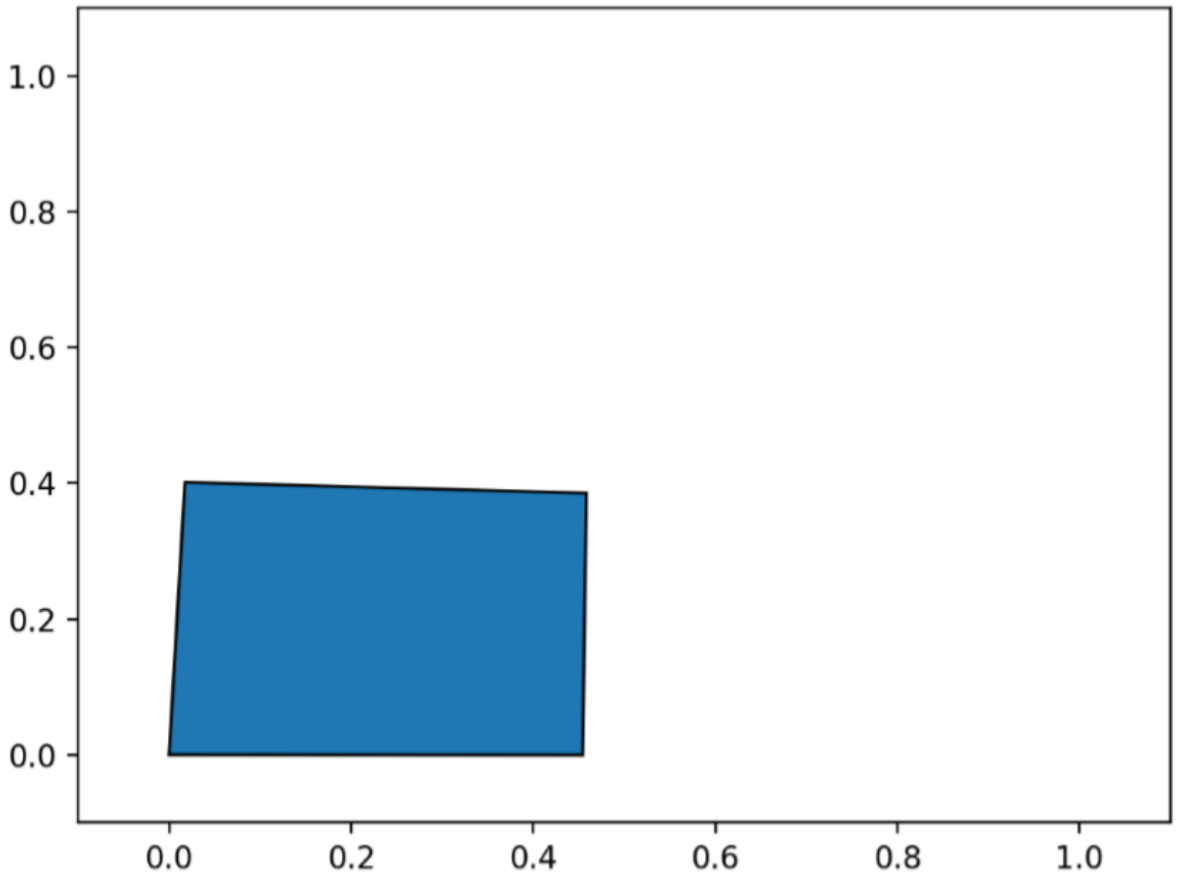

图 11.4 显示由Generative LSTM模型生成的矩形

11.6 评估模型

评估一个Generative LSTM模型是困难的。在这种情况下，我们对生成的形状（矩形）有很好的期望，可以定量和定性地进行评价。在许多问题中，情况困难并非如此，你可能不得不依靠定性的评价。

我们可以通过可视化实现对矩形的定性评价。一下是在不同的数量的训练之后生成的矩形的示例，以显示随着训练例子的增肌，模型的学习能力的定性改进。

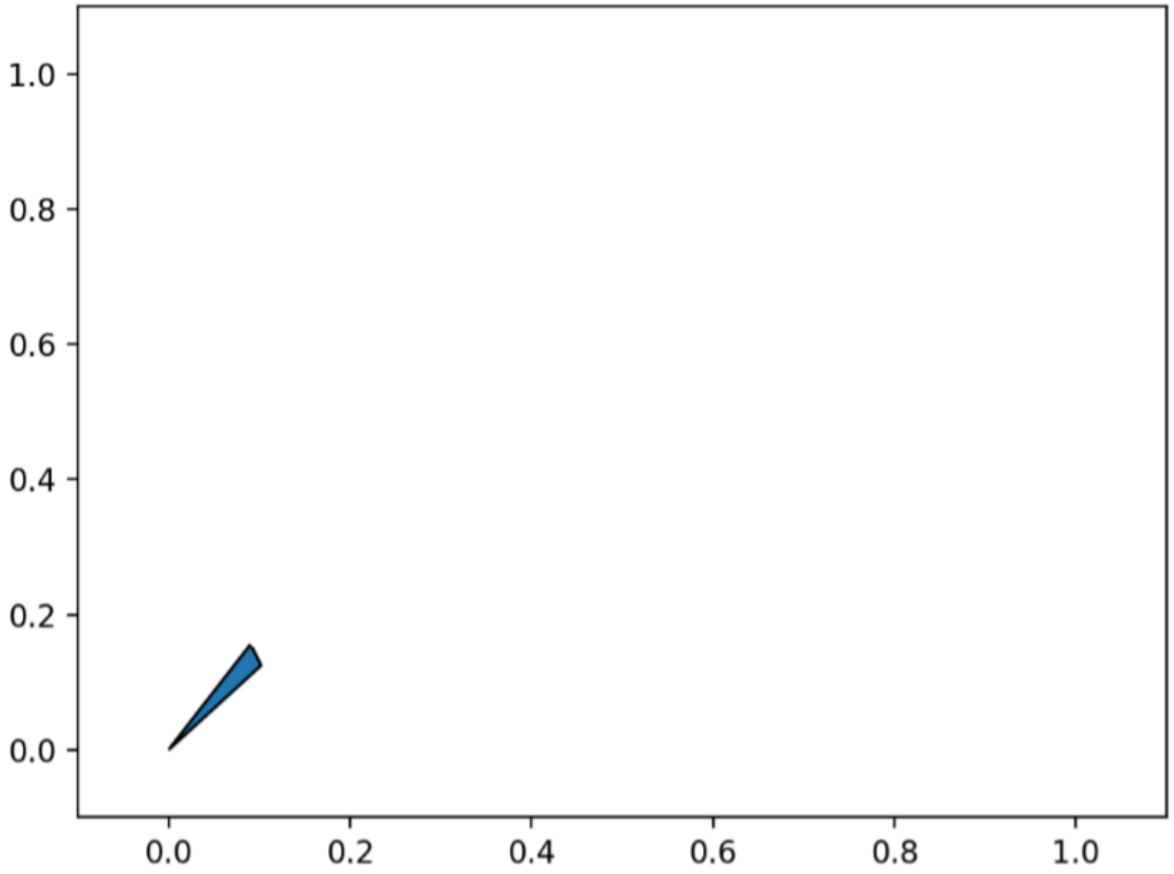

图 11.5 通过100个实例生成LSTM模型生成的矩形图

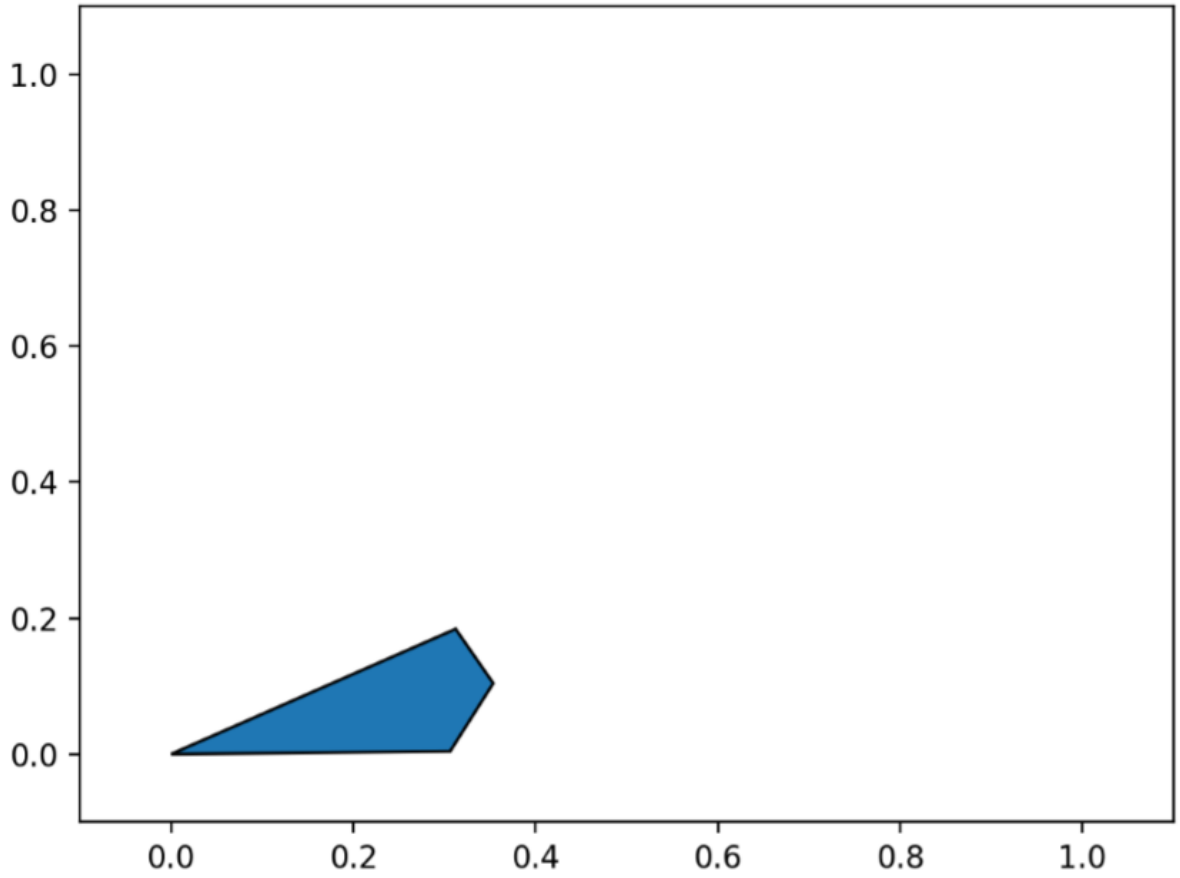

图 11.6 通过500个实例生成LSTM模型生成的矩形图

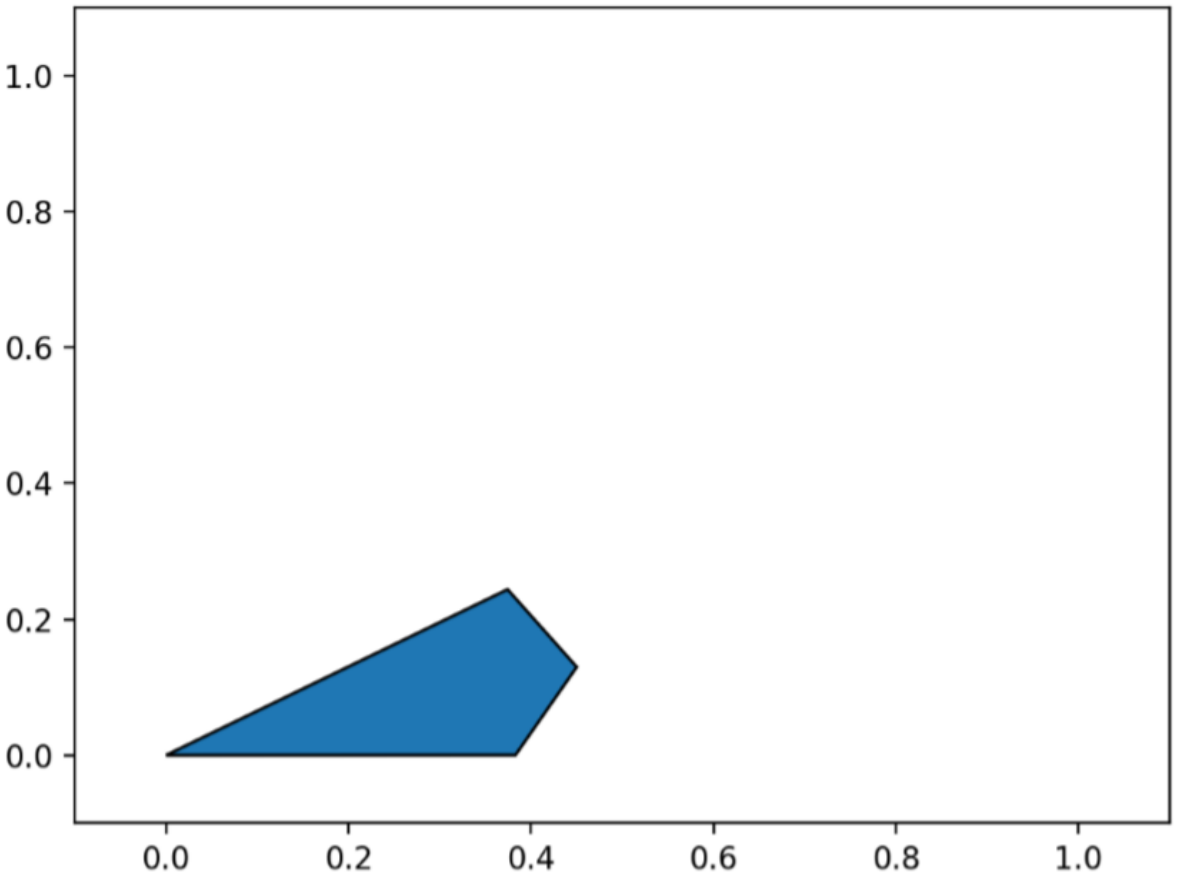

图 11.7 通过1000个实例生成LSTM模型生成的矩形图

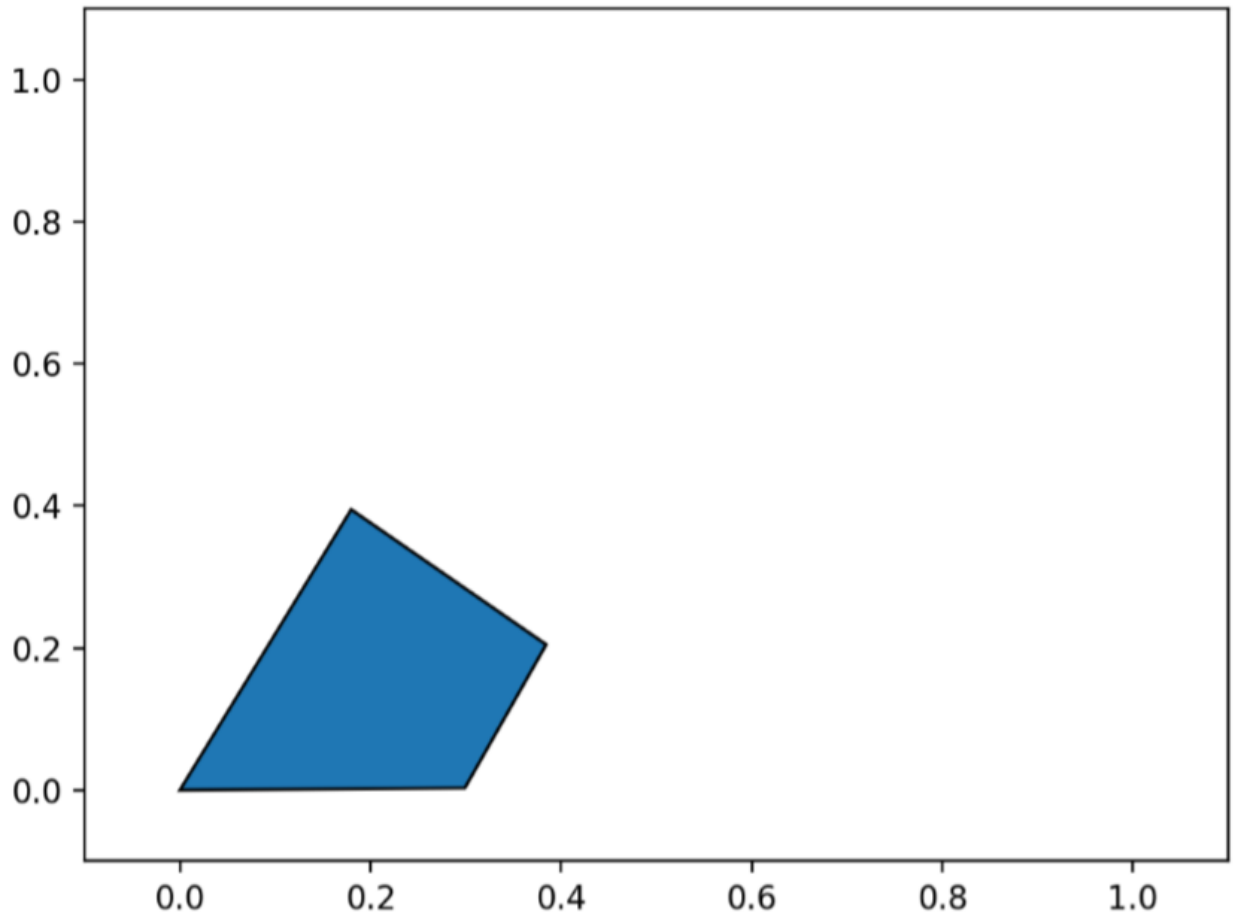

图 11.7 通过5000个实例生成LSTM模型生成的矩形图

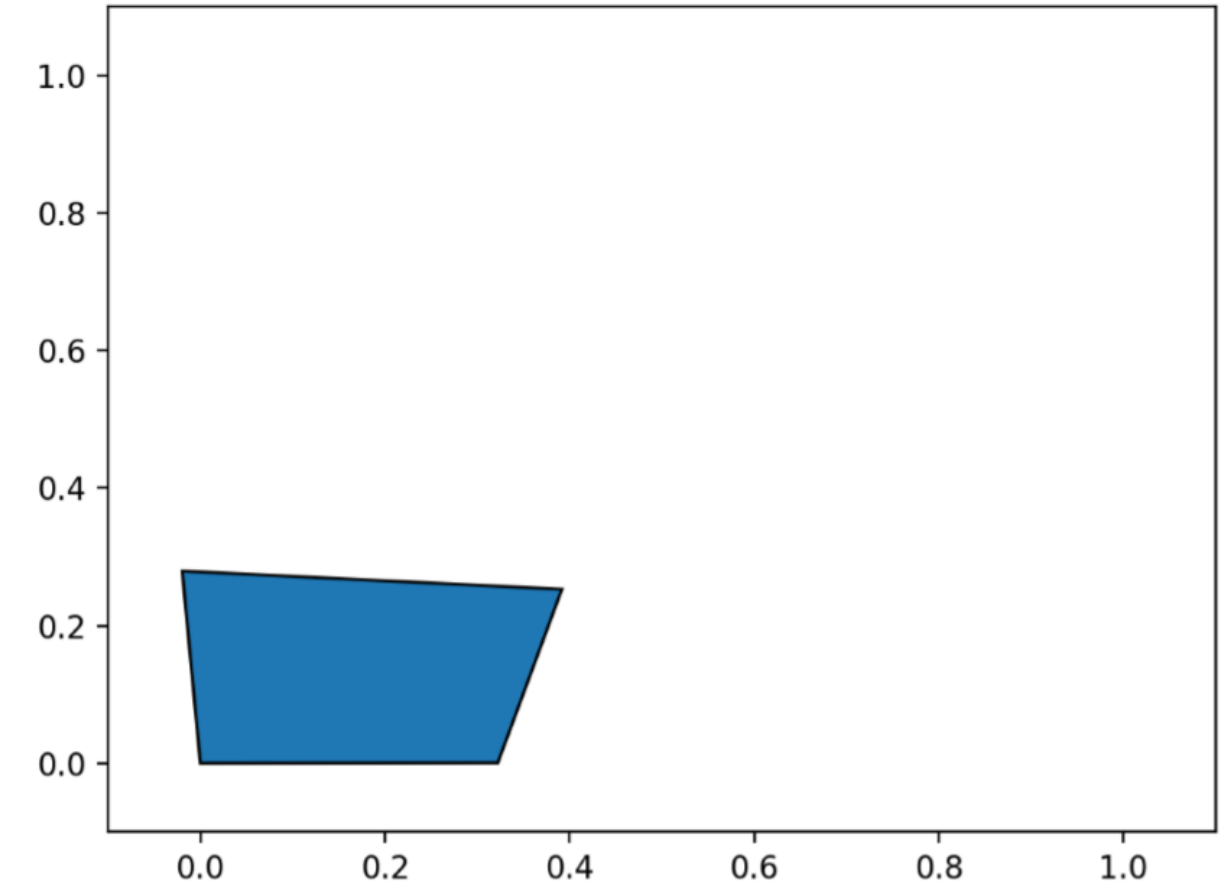

图 11.7 通过10000个实例生成LSTM模型生成的矩形图

11.7 完整例子

完整的代码列表提供如下给你做参考：

```
from random import random
from numpy import array
from matplotlib import pyplot
from matplotlib.patches import PathPatch
from matplotlib.path import Path
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# generate a rectangle with random width and height
def random_rectangle():
    width, height = random(), random()
    points = list()
    # bottom left
    points.append([0.0, 0.0])
    # bottom right
    points.append([width, 0.0])
    # top right
    points.append([width, height])
    # top left
    points.append([0.0, height])
    return points

# plot a rectangle
def plot_rectangle(rect):
    # close the rectangle path
    rect.append(rect[0])
    # define path
    codes = [Path.MOVETO, Path.LINETO, Path.LINETO, Path.LINETO, Path.CLOSEPOLY]
    path = Path(rect, codes)
    axis = pyplot.gca()
    patch = PathPatch(path)
    # add shape to plot
    axis.add_patch(patch)
    axis.set_xlim(-0.1,1.1)
    axis.set_ylim(-0.1,1.1)
    pyplot.show()

# generate input and output sequences for one random rectangle
def get_samples():
    # generate rectangle
    rect = random_rectangle()
    X, y = list(), list()
    # create input output pairs for each coordinate
    for i in range(1, len(rect)):
        X.append(rect[i-1])
        y.append(rect[i])
    # convert input sequence shape to have 1 time step and 2 features
    X, y = array(X), array(y)
    X = X.reshape((X.shape[0], 1, 2))
    return X, y
```

```
# use a fit LSTM model to generate a new rectangle from scratch
def generate_rectangle(model):
    rect = list()
    # use [0,0] to seed the generation process
    last = array([0.0,0.0]).reshape((1, 1, 2))
    rect.append([[y for y in x] for x in last[0]][0])
    # generate the remaining 3 coordinates
    for i in range(3):
        # predict the next coordinate
        yhat = model.predict(last, verbose=0)
        # use this output as input for the next prediction
        last = yhat.reshape((1, 1, 2))
        # store coordinate
        rect.append([[y for y in x] for x in last[0]][0])
    return rect

# define model
model = Sequential()
model.add(LSTM(10, input_shape=(1, 2)))
model.add(Dense(2, activation= 'linear' ))
model.compile(loss= 'mae' , optimizer= 'adam' )
print(model.summary())

# fit model
for i in range(25000):
    X, y = get_samples()
    model.fit(X, y, epochs=1, verbose=2, shuffle=False)

# generate new shapes from scratch
rect = generate_rectangle(model)
plot_rectangle(rect)
```

表 11.29 生成模型在矩形生成问题上的例子

11.8 扩展阅读

本章节提供了一些用于扩展阅读的资源。

11.8.1 论文

-Generating Text with Recurrent Neural Networks, 2011.

- Generating Sequences With Recurrent Neural Networks, 2013.
- TTS Synthesis with Bidirectional LSTM based Recurrent Neural Networks, 2014.
- A First Look at Music Composition using LSTM Recurrent Neural Networks, 2002.
- Jazz Melody Generation from Recurrent Network Learning of Several Human Melodies, 2005.

11.8.2 文章

- The Unreasonable Effectiveness of Recurrent Neural Networks, 2015.

11.8.3 APIs

- Python random API.
- Matplotlib Path API.
- Matplotlib Patch API.

11.9 扩展

你想更加深入地了解Generative LSTM吗？本章节列出了本课中一些具有挑战性的扩展。

- 列出5个其他问题的例子，而不是其他的语言模型，Generative LSTM可以被使用。
- 设计并执行实验以比较模型大小（神经元数）与定性模型学习能力（图像）。
- 更新示例，使得随机矩阵由更多的点组成（例如，在中间宽度的点和矩形的中高度），然后调谐LSTM以获得良好的技能。
- 开发一个函数来估计矩形误差在宽度和高度不一致的情况下，在拟合生成LSTM时用作损失函数和度量。
- 更新示例以学习不同的形状，例如圆、星型或十字。

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！

11.10 总结

在本课程中，我们学习到了怎么样开发一个Generative LSTM模型。特别地，你学习到了：

- LSTMs怎么样可以被用于生成模型；
- 怎么样把形状绘制作为序列生成问题；
- 怎么开发一个LSTM模型来生成形状。

在下一课中，你将会学到怎么样充分利用LSTM模型。