[转载]使用PyTroch搭建LSTM预测时间序列

> 时间序列就是以时间为自变量的一系列数据。例如, 24小时的温度,各种产品一个月的价格变动, 一个公司一年的股票价格。 现在前沿深度学习模型比如LSTM能够捕捉时间序列的规律，因此可以用来预测数据未来的趋势。在这篇文章中，你可以了解到如何使用LSTM深度学习算法使用时间序列来预测未来。

数据集

我们将会使用的数据来自Python Seaborn包。首先，我们先导入必要的包：

```
import torch
import torch.nn as nn
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

我们先print出Seaborn的所有内置数据集列表：

```
sns.get_dataset_names()
```

```
['anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'iris',
 'mpg',
 'planets',
 'tips',
 'titanic']
```

我们将要使用的数据集是Seaborn中的航班数据集，在jupyter notebook中加载：

```
flight_data = sns.load_dataset("flights")
flight_data.head()
```

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121

这个数据集有个属性，分别是时间（年、月）和乘客数量。乘客数量这列包含了在一个月中乘客的总量。我们来看看数据集的大小：

```
flight_data.shape
```

```
(144, 3)
```

你可以看到这个数据集有144行3列意味着它包含了十二年的乘客记录。

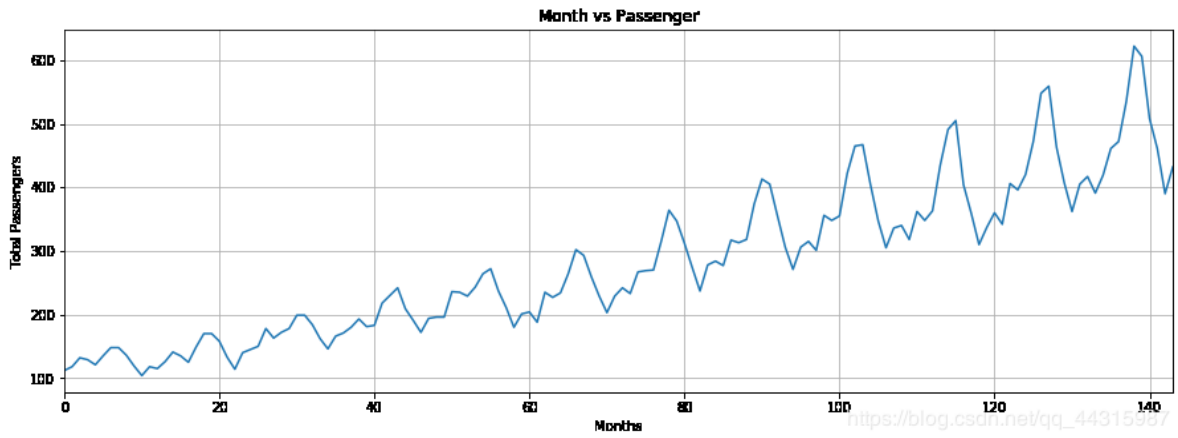
我们的任务是基于十二年的132个月中旅行乘客的数量来预测往后乘客的数量。由上面的输出可以知道，我们有144个的记录，所以我们将把一开始的132个月当成训练集，最后的十二个月当成测试集。

我们先画出每个月旅行乘客数量的直方图。第一步是调整图像大小：

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0]=15
fig_size[1]=5
plt.rcParams["figure.figsize"]= fig_size
```

下一步是画图：

```
plt.title('Month vs Passenger')
plt.ylabel('Total Passengers')
plt.xlabel('Months')
plt.grid(True)
plt.autoscale(axis='x',tight=True)
plt.plot(flight_data['passengers'])
```



从这个输出中可以看出，这十二年的乘机旅行乘客人数在增加。还可以看出乘客数量一年中有波动，可能的原因是暑假和寒假的乘客数量相对别的时期较多。

数据预处理

我们数据集属性的类型用下面这段代码来展示：

```
flight_data.columns

Index(['year', 'month', 'passengers'], dtype='object')
```

第一步我们应该把乘客数改成浮点类型：

```
all_data = flight_data['passengers'].values.astype(float)
```

现在如果你printall_data，你会发现下面的浮点类型值：

```
print(all_data)
```

```
[112. 118. 132. 129. 121. 135. 148. 148. 136. 119. 104. 118. 115. 126.
 141. 135. 125. 149. 170. 170. 158. 133. 114. 140. 145. 150. 178. 163.
 172. 178. 199. 199. 184. 162. 146. 166. 171. 180. 193. 181. 183. 218.
 230. 242. 209. 191. 172. 194. 196. 196. 236. 235. 229. 243. 264. 272.
 237. 211. 180. 201. 204. 188. 235. 227. 234. 264. 302. 293. 259. 229.
 203. 229. 242. 233. 267. 269. 270. 315. 364. 347. 312. 274. 237. 278.
 284. 277. 317. 313. 318. 374. 413. 405. 355. 306. 271. 306. 315. 301.
 356. 348. 355. 422. 465. 467. 404. 347. 305. 336. 340. 318. 362. 348.
 363. 435. 491. 505. 404. 359. 310. 337. 360. 342. 406. 396. 420. 472.
 548. 559. 463. 407. 362. 405. 417. 391. 419. 461. 472. 535. 622. 606.
 508. 461. 390. 432.]
```

下一步，我们将要把数据集分为训练集和测试集。LSTM算法将会在训练集上训练。训练出的模型将会使用在测试集上，用来预测未来。预测结果将会和实际结果作比较。

```
test_data_size =12

train_data = all_data[:-test_data_size]
test_data = all_data[-test_data_size:]
```

现在我们pirnt出训练集和测试集的长度：

```
print(len(train_data))
print(len(test_data))
```

```
132
12
```

我们的数据集还没有标准化。乘客初始的总量远少于最后几年的数量。我们将使用线性函数归一化（min/max scaling）来使所有数据的范围在0~1之间

$$x' = \frac{x - min(x)}{max(x) - min(x)} * (feature\_range[1] - feature\_range[0]) + feature\_range[0]$$

即x'=x−min(x)/max(x)−min(x)∗(feature_range[1]−feature_range[0])+feature_range[0]

我们会使用sklearn.preprocessing模块中的MinMxaScaler类来归一化数据。

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(-1,1))
train_data_normalized = scaler.fit_transform(train_data .reshape(-1,1))
```

我们print出归一化的前五个元素和后五个元素：

```
print(train_data_normalized[:5])
print(train_data_normalized[-5:])
```

```
[[-0.96483516]
 [-0.93846154]
 [-0.87692308]
 [-0.89010989]
 [-0.92527473]]
[[1.        ]
 [0.57802198]
 [0.33186813]
 [0.13406593]
 [0.32307692]]
```

你可以看到数据集的值在-1和1之间，值得注意的是归一化只能用在数据集而不能用在测试集。~~如果归一化用在测试集，会有信息从训练集泄露到测试集中。~~（个人认为此处不对）

下一步是把我们的数据集转换为PyTorch的张量（tensor）：

```
train_data_normalized = torch.FloatTensor(train_data_normalized).view(-1)
train_data_normalized
```

```
tensor([-0.9648, -0.9385, -0.8769, -0.8901, -0.9253, -0.8637, -0.8066, -0.8066,
        -0.8593, -0.9341, -1.0000, -0.9385, -0.9516, -0.9033, -0.8374, -0.8637,
        -0.9077, -0.8022, -0.7099, -0.7099, -0.7626, -0.8725, -0.9560, -0.8418,
        -0.8198, -0.7978, -0.6747, -0.7407, -0.7011, -0.6747, -0.5824, -0.5824,
        -0.6484, -0.7451, -0.8154, -0.7275, -0.7055, -0.6659, -0.6088, -0.6615,
        -0.6527, -0.4989, -0.4462, -0.3934, -0.5385, -0.6176, -0.7011, -0.6044,
        -0.5956, -0.5956, -0.4198, -0.4242, -0.4505, -0.3890, -0.2967, -0.2615,
        -0.4154, -0.5297, -0.6659, -0.5736, -0.5604, -0.6308, -0.4242, -0.4593,
        -0.4286, -0.2967, -0.1297, -0.1692, -0.3187, -0.4505, -0.5648, -0.4505,
        -0.3934, -0.4330, -0.2835, -0.2747, -0.2703, -0.0725,  0.1429,  0.0681,
        -0.0857, -0.2527, -0.4154, -0.2352, -0.2088, -0.2396, -0.0637, -0.0813,
        -0.0593,  0.1868,  0.3582,  0.3231,  0.1033, -0.1121, -0.2659, -0.1121,
        -0.0725, -0.1341,  0.1077,  0.0725,  0.1033,  0.3978,  0.5868,  0.5956,
         0.3187,  0.0681, -0.1165,  0.0198,  0.0374, -0.0593,  0.1341,  0.0725,
         0.1385,  0.4549,  0.7011,  0.7626,  0.3187,  0.1209, -0.0945,  0.0242,
         0.1253,  0.0462,  0.3275,  0.2835,  0.3890,  0.6176,  0.9516,  1.0000,
         0.5780,  0.3319,  0.1341,  0.3231])
```

最后一步是制作我们的训练集的对用的标签

```
train_window =12
```

接下来，我们要定义一个叫create_inout_sequences的函数。这个函数将会接收数据输入的行，然后将会return一个包含元组的列表。

在每一个元组中，第一个元素将会包含十二个月所对应的乘客数，第二个元素包含下一个十二月所对应的乘客数量

```
defcreate_inout_sequences(input_data, tw):
    inout_seq =[]
    L =len(input_data)
    for i inrange(L-tw):
        train_seq = input_data[i:i+tw]
        train_label = input_data[i+tw:i+tw+1]
        inout_seq.append((train_seq ,train_label))
    return inout_seq
```

执行下面代码来创建数据集序列和对应的标签

```
train_inout_seq = create_inout_sequences(train_data_normalized, train_window)
```

如果你print train_inout_seq 列表，你会发现他包含120项。这是因为虽然训练集有132个元素，但是这个序列的长度是12，意味着第一个序列由一开始的12项和第13项组成，其中第十三项就是标签。同样，第二个序列由第二项到第十三项组成，第十四项为他的标签。

```
train_inout_seq[:5]
```

```
[(tensor([-0.9648, -0.9385, -0.8769, -0.8901, -0.9253, -0.8637, -0.8066,
-0.8066,
          -0.8593, -0.9341, -1.0000, -0.9385]), tensor([-0.9516])),
 (tensor([-0.9385, -0.8769, -0.8901, -0.9253, -0.8637, -0.8066, -0.8066,
-0.8593,
          -0.9341, -1.0000, -0.9385, -0.9516]),
  tensor([-0.9033])),
 (tensor([-0.8769, -0.8901, -0.9253, -0.8637, -0.8066, -0.8066, -0.8593,
-0.9341,
          -1.0000, -0.9385, -0.9516, -0.9033]), tensor([-0.8374])),
 (tensor([-0.8901, -0.9253, -0.8637, -0.8066, -0.8066, -0.8593, -0.9341,
-1.0000,
          -0.9385, -0.9516, -0.9033, -0.8374]), tensor([-0.8637])),
 (tensor([-0.9253, -0.8637, -0.8066, -0.8066, -0.8593, -0.9341, -1.0000,
-0.9385,
          -0.9516, -0.9033, -0.8374, -0.8637]), tensor([-0.9077]))]
```

PS：这个切分不好理解，可以画图演示便于理解。

搭建LSTM模型

我们已经做完数据预处理，接下来来训练我们的模型。我们首先定义一个LSTM类，他继承PyTorch的nn.Moudule类。

```
class LSTM(nn.Module):
    def __init__(self, input_size=1, hidden_layer_size=100, output_size=1):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size

        self.lstm = nn.LSTM(input_size, hidden_layer_size)
```

```
        self.linear = nn.Linear(hidden_layer_size, output_size)

        self.hidden_cell = (torch.zeros(1,1,self.hidden_layer_size),
                            torch.zeros(1,1,self.hidden_layer_size)) #
(num_layers * num_directions, batch_size, hidden_size)

    def forward(self, input_seq):
        lstm_out, self.hidden_cell = self.lstm(input_seq.view(len(input_seq) ,1,
-1), self.hidden_cell)
        predictions = self.linear(lstm_out.view(len(input_seq), -1))
        return predictions[-1]
```

我们总结一下上面代码。LSTM的结构包括三个参数：

input_size：类似于输入特征的数量。虽然我们的序列长度是12，但是对于每个月来说，我们仅仅有一个值就是每个月的总乘客量。

hidden_layer_size：隐藏层单元的个数

output_size：输出项的数量，由于我们想预测未来一个月的乘客数量，所以outputsize是1。

接下来我们创建hidden_layer_size, lstm, linear, hidden_cell变量。LSTM算法接收三个输入：上一个隐藏状态，上一个记忆细胞状态和当前输入。hidden_cell这个变量包括上一个隐藏单元和记忆细胞的状态。lstm变量和linear layer变量是用来创建LSTM和线性层的，其中线性层是用来输出的。

在forward方法里，input_seq参数是我们输入的长度为12的序列，lstm的输出结果是隐层和记忆细胞的在当前时间点的状态。lstm层的输出结果经过线性层计算出最后的结果。

下一个阶段就是创建一个LSTM()类的示例，定义损失函数和优化函数。这里我们用的是交叉熵损失函数和adam优化方法。

```
model = LSTM()
loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

print出我们的模型！

```
print(model)
```

```
LSTM(
  (lstm): LSTM(1, 100)
  (linear): Linear(in_features=100, out_features=1, bias=True)
)
```

训练模型

我们训练150轮。你可以尝试更多的epochs。每25轮输出一次loss

```
epochs = 150

for i in range(epochs):
    for seq, labels in train_inout_seq:
        optimizer.zero_grad()
        model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                        torch.zeros(1, 1, model.hidden_layer_size))
```

```
        y_pred = model(seq)

        single_loss = loss_function(y_pred, labels)
        single_loss.backward()
        optimizer.step()

    if i%25 == 1:
        print(f'epoch: {i:3} loss: {single_loss.item():10.8f}')

print(f'epoch: {i:3} loss: {single_loss.item():10.10f}')
```

```
epoch:   1 loss: 0.01516276
epoch:  26 loss: 0.00965644
epoch:  51 loss: 0.00252884
epoch:  76 loss: 0.01772214
epoch: 101 loss: 0.00010952
epoch: 126 loss: 0.00532575
epoch: 149 loss: 0.0000004581
```

你可能得到不同的值因为PyTorch神经网络的默认权重是随机的。

PS：此处的loss并不是本轮的平均loss，而是本轮的最后一个序列的loss，函数里的平均意思是指y_pred和labels里面所有元素的平均值！！！所有loss曲线波动很大、

做出预测

现在我们的模型已经训练完了，我们可以开始做预测。由于我们的测试集包含最后十二个月的乘客数，并且我们的模型每次训练的序列长度为12，所以我们首先筛选出训练集的最后12个值：

```
fut_pred = 12

test_inputs = train_data_normalized[-train_window:].tolist()
print(test_inputs)
```

```
[0.12527473270893097, 0.04615384712815285, 0.3274725377559662,
0.2835164964199066, 0.3890109956264496, 0.6175824403762817, 0.9516483545303345,
1.0, 0.5780220031738281, 0.33186814188957214, 0.13406594097614288,
0.32307693362236023]
```

初始的测试集包含12项。这12项将会预测测试集的第一项，测试集第一项的数字是133.预测出的值接下来会加入到test_inputs列表的末尾。在第二次循环中，最后的12项又作为输入，一个新的预测值又会加入到test_inputs列表的末尾。这个循环将会执行12次。在循环的最后，test_inputs将会包含24项。最后的12项就是测试集的预测结果。

下面的代码是用来预测的：

```
model.eval()

for i in range(fut_pred):
    seq = torch.FloatTensor(test_inputs[-train_window:])
    with torch.no_grad():
        model.hidden = (torch.zeros(1, 1, model.hidden_layer_size),
                        torch.zeros(1, 1, model.hidden_layer_size))
        test_inputs.append(model(seq).item())
```

如果你print test_inputs 列表，你会发现它包含24个元素。最后12个预测结果如下：

```
test_inputs[fut_pred:]
```

```
[0.8146875500679016,
 0.7292792201042175,
 0.9062390923500061,
 1.2367297410964966,
 1.3315496444702148,
 1.2219644784927368,
 1.519028663635254,
 1.2008862495422363,
 1.5351794958114624,
 1.496110200881958,
 1.5902581214904785,
 1.5625801086425781]
```

由于我们一开始将数据集标准化了，所以预测值也是标准化后的数据。我们需要将标准化后的数据转换成实际预测值。我们可以使用min/max scaler实例的inverse_transform 方法来将预测值转化为实际值：

```
actual_predictions =
scaler.inverse_transform(np.array(test_inputs[train_window:] ).reshape(-1, 1))
print(actual_predictions)
```

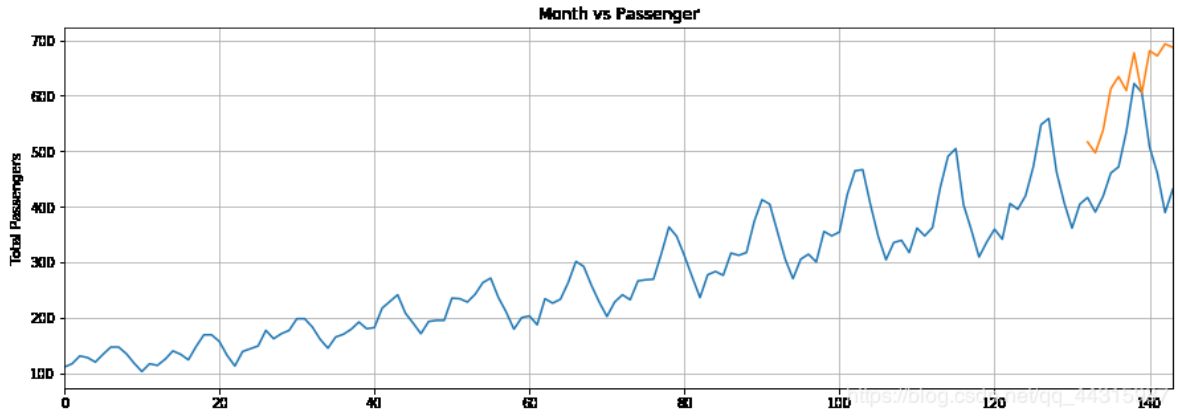
```
[[516.84141764]
 [497.41102257]
 [537.66939351]
 [612.8560161 ]
 [634.42754412]
 [609.49691886]
 [677.07902098]
 [604.70162177]
 [680.7533353 ]
 [671.8650707 ]
 [693.28372264]
 [686.98697472]]
```

现在我们画出图像：

```
x = np.arange(132, 144, 1)
print(x)
```

```
[132 133 134 135 136 137 138 139 140 141 142 143]
```

```
plt.title('Month vs Passenger')
plt.ylabel('Total Passengers')
plt.grid(True)
plt.autoscale(axis='x', tight=True)
plt.plot(flight_data['passengers'])
plt.plot(x,actual_predictions)
plt.show()
```

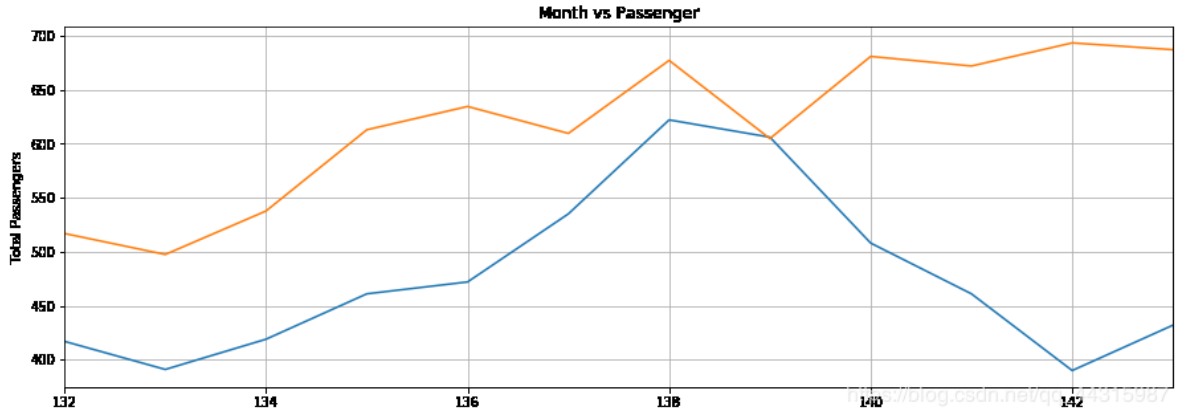


LSTM预测结果是橘色的曲线。你可以发现我们的算法不是很准确，很大的原因是数据集太小，但是它仍然可以捕捉到全部乘客数据最后十二个月的趋势。你可以尝试更多的epochs使结果更好。

为了更好的观察输出结果，你可以用下面代码画出最后十二个月的图像：

```
plt.title('Month vs Passenger')
plt.ylabel('Total Passengers')
plt.grid(True)
plt.autoscale(axis='x', tight=True)

plt.plot(flight_data['passengers'][-train_window:])
plt.plot(x,actual_predictions)
plt.show()
```



原作者总结

LSTM使解决序列问题用处最广的算法之一，在这篇文章中，我们可以知道怎么使用LSTM来预测时间序列。当然，这个算法还有很多提升空间，由于循环神经网络容易出现梯度爆炸的情况，我们可以加入梯度裁剪；为了减少泛化误差，我们可以使用k折交叉验证等等方法来优化模型。

原文链接：https://blog.csdn.net/qq_44315987/article/details/104621632

个人总结：

作者的整个实验流程除了少一个用训练集预测模型进行调参、动态调整学习率，其他的各个方面都很值得借鉴，一是数据的切分和处理，二是整个模型的设计，都很值得借鉴，当然这两部分也都有改进的地方；三是整个流程特别适合初学者进行实际操作，同时便于明白原理。另外他的loss处理有点问题，为了查找和排查问题，个人建议输出loss曲线，直观地观察！

Write by sheen