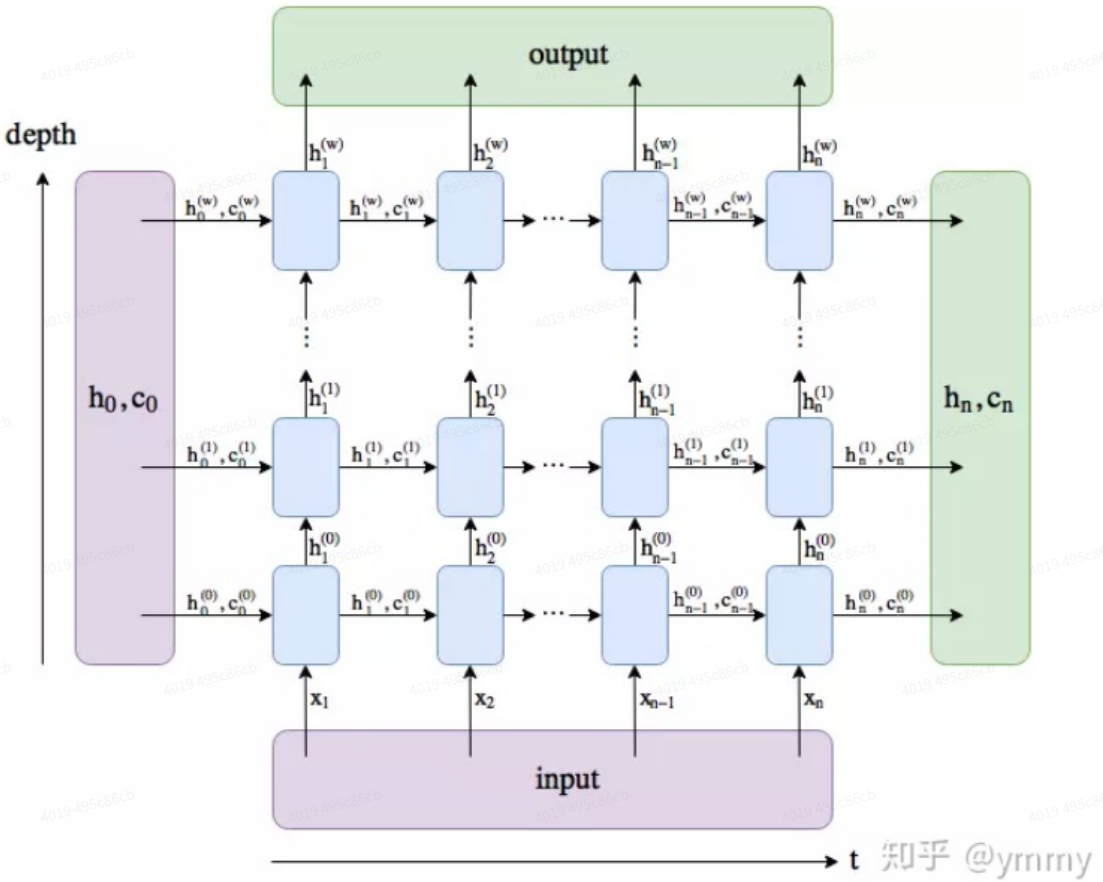
4.LSTM实操

1.LSTM网络结构与pytorch

LSTM中将整个网络看成这样：



蓝色的模块是指前面的单位LSTM结构。横向连接的各个LSTM结构就是序列化。不同状态（时间）下的LSTM结构。纵向连接的是同一状态下LSTM结构。或者我们可以简单看作是多个LSTM细胞结构串联，其个数为num_layers, 将其视为一个新的LSTM结构，并且按照时序连接起来。图中每个参数都是自带维度的。

Pytorch中的库：

nn.LSTM(input_size, hidden_size, num_layers)

input_size：输入的X的维度。

hidden_size: 输出和输入的hi 的维度。

num_layers: depth，即串联的LSTM个数。

2.时间序列实战

2.1. 数据

数据集，是从1949-1960,12年，每个月的乘客数量。即一共有144条数据，表示了144个月的乘客数量。

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121
139	1960	August	606
140	1960	September	508
141	1960	October	461
142	1960	November	390
143	1960	December	432

2.2 思路

采用的肯定是利用前几期的数据来预测当前期的数据。(具体的方法是：利用前N（N= 3）期的数据为输入，当前期的数据为标签计算误差)。个人觉得可以N是一个超参，可以慢慢调.

由于时间序列数据每个时期的数据样本只有一个，那么X为(time_step, 1, input_size), Y为(time_step, 1, output_size)。

构建一个LSTM网络，输入的数据就是X，hidden_size可任意取，num_layers也可视情况取，即用多少层LSTM串联(同一时期内）。

最后用一层线性层Linear(hidden_size, output_size)进行输出。比较输出与Y的误差。不断迭代对参数进行优化。

2.3代码

Python

```python
#step 1. 加载飞行数据
flight_data = pd.read_csv('flights.csv')
# 数据归一化
maxPassenger = flight_data['passengers'].max()
minPassenger = flight_data['passengers'].min()
flight_data['passengers'] = (flight_data['passengers'] - minPassenger) \
     / (maxPassenger - minPassenger)

dataset = flight_data['passengers'].values.tolist()

#step 2. 划分数据集
# 数据集合目标函数值赋值，其中dateset为数据，look_back为以几行数据为特征数目
# look_back表示3期回头，即使用前三期的数据预测下一期
# 用前3期数据预测下1期
def createDataset(dataset, look_back):
    dataX = []
    dataY = []
    for i in range(len(dataset)-look_back):
        dataX.append(dataset[i:i+look_back])
        dataY.append(dataset[i+look_back])

    dataX = torch.tensor(dataX)
    dataX = dataX.reshape(-1, 1, look_back)
    dataY = torch.tensor(dataY)
    dataY = dataY.reshape(-1, 1, 1)
    return dataX, dataY

data = createDataset(dataset=dataset, look_back=3)    # 划分数据集,3个月为一组

# step3. 划分训练集和测试集
# 由于是时间序列数据，不适合这样随机打乱
def splitData(data,rate=0.7): #默认是0.7的训练集，0.2的测试集
    # 默认训练集比例为0.7
    dataX, dataY = data
    nSample = dataX.shape[0]
    nTrain = int(nSample*rate)
    trainData = (dataX[:nTrain], dataY[:nTrain])
    testData = (dataX[nTrain:], dataY[nTrain:])
    return trainData, testData


# 获取训练集和测试集，用80%的数据来训练拟合，20%的数据来预测
rate = 0.8
trainData, testData = splitData(data, rate=rate)
```

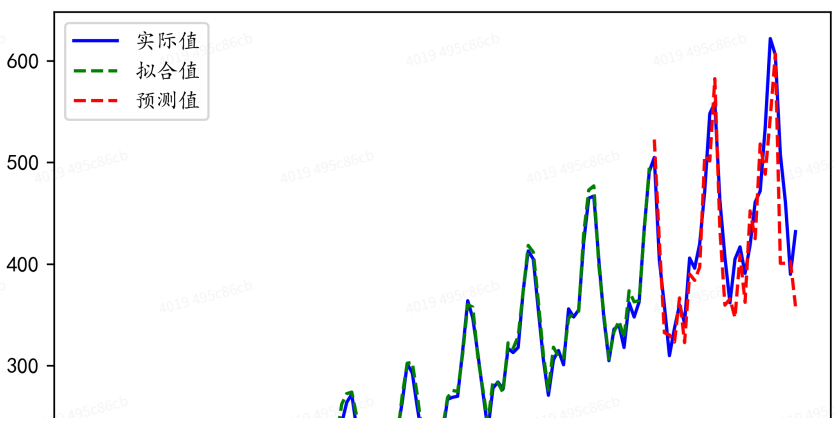
Python

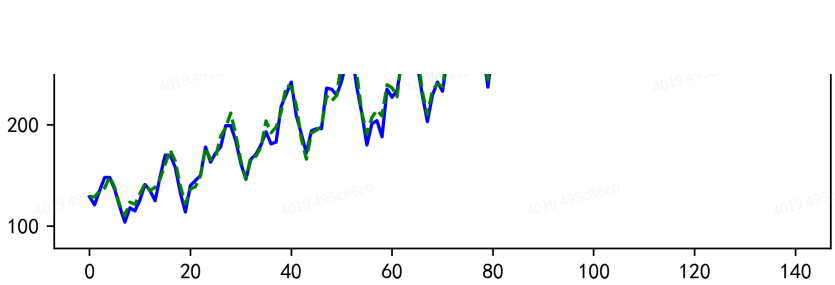
```python
# step4: 定义模型
class LstmModel(nn.Module):
    def __init__(self, inputSize=5, hiddenSize=6):
        super().__init__()
        # LSTM层-> 两个LSTM单元叠加
        self.lstm = nn.LSTM(input_size = inputSize,
                            hidden_size = hiddenSize,
                            num_layers = 2)
        self.output = nn.Linear(6,1)  # 线性输出


    def forward(self,x):
        # x: input->(time_step, batch, input_size)
        # x的维度是【数量量：整批数量量：输入特征维度】
        # X是【112 ,1, 3】
        # lstm两层，目标是从 3->6

        x1, (h ,c)= self.lstm(x)
        # x1: output->(time_step, batch, output_size)

        a, b, c = x1.shape
        out = self.output(x1.view(-1,c))  # 只有三维数据转化为二维才能作为输入
        # 重新将结果转化为三维
        out = out.view(a,b,-1)
        return out

# 定义模型
lstm = LstmModel(inputSize=3) # inputSize与look_back保持一致

# step5.模型训练
def training_loop(nEpochs, model, optimizer, lossFn, trainData,
testData=None):
    trainX, trainY = trainData
    if testData is not None:
        testX, testY = testData
    for epoch in range(1, nEpochs+1):
        optimizer.zero_grad()  # 梯度清0
        trainP = model(trainX)
        loss = lossFn(trainP, trainY)
        loss.backward()  # 反向传播
        optimizer.step()
        if epoch % 100 == 0:
            print(f"Epoch: {epoch}, Loss: {loss.item()}")
    return model

```

```
45
46  # 使用优化器Adam比SGD更好
47  optimizer = optim.Adam(lstm.parameters(), lr=0.1)
48  loss_func = nn.MSELoss()
49
50  # 训练模型
51  lstm = training_loop(nEpochs=1000, model= lstm,
52                       optimizer=optimizer, lossFn=loss_func,
    trainData=trainData)
```

Dockerfile

```
1   #Step6: 可视化
2   dataX, dataY = data  # 原始数据 -> (time_step, batch, input_size)
3   dataY = dataY.view(-1).data.numpy()  # 展开为1维
4   dataY = dataY * (maxPassenger - minPassenger) + minPassenger
5   dataP = lstm(dataX)  # 进行拟合
6   dataP = dataP.view(-1).data.numpy()  # 展开为1维
7   dataP = dataP * (maxPassenger - minPassenger) + minPassenger
8
9   nTrain = int(dataX.shape[0] * rate)  # 拟合的数量
10  nData = dataX.shape[0]  # 预测的数量
11
12  # 绘制对比图
13  plt.rcParams['font.sans-serif'] = 'KaiTi'  # 正常显示中文
14  fig = plt.figure(dpi=400)
15  ax = fig.add_subplot(111)
16  ax.plot(dataY, color='blue', label="实际值")
17  ax.plot(np.arange(nTrain), dataP[:nTrain], color='green',\
18      linestyle='--', label = '拟合值')
19  ax.plot(np.arange(nTrain, nData), dataP[nTrain:], \
20       linestyle='--', color = 'red', label='预测值')
21  ax.legend()
22  fig.savefig('test.png', dpi=400)
23
```




200
100
0
20
40
60
80
100
120
140