2021.11

黄元通

20195063

自然语言处理

实验报告

目录

[基础内容 搭建LSTM网络 2](#_Toc87801761)

[一、题目 2](#_Toc87801762)

[二、网络结构设计 2](#_Toc87801763)

[1、计算公式 2](#_Toc87801764)

[2、程序实现 3](#_Toc87801765)

[三、传入、传出及其他细节 4](#_Toc87801766)

[1、输入参数及维度 4](#_Toc87801767)

[2、运行设备统一 5](#_Toc87801768)

[3、输出参数及维度 5](#_Toc87801769)

[4、未传state时的默认初始化 7](#_Toc87801770)

[四、结果 8](#_Toc87801771)

[提高内容 搭建双层LSTM网络 10](#_Toc87801772)

[一、题目 10](#_Toc87801773)

[二、网络结构设计 10](#_Toc87801774)

[1、流程图 10](#_Toc87801775)

[2、程序实现 10](#_Toc87801776)

[三、传入、传出及其他细节 11](#_Toc87801777)

[1、传入 11](#_Toc87801778)

[2、传出 12](#_Toc87801779)

[四、结果 12](#_Toc87801780)

[附录 14](#_Toc87801781)

[一、主程序修改（使用说明） 14](#_Toc87801782)

[二、LSTM.py源码 14](#_Toc87801783)

# 基础内容 搭建LSTM网络

## 一、题目

尝试自己搭建LSTM网络

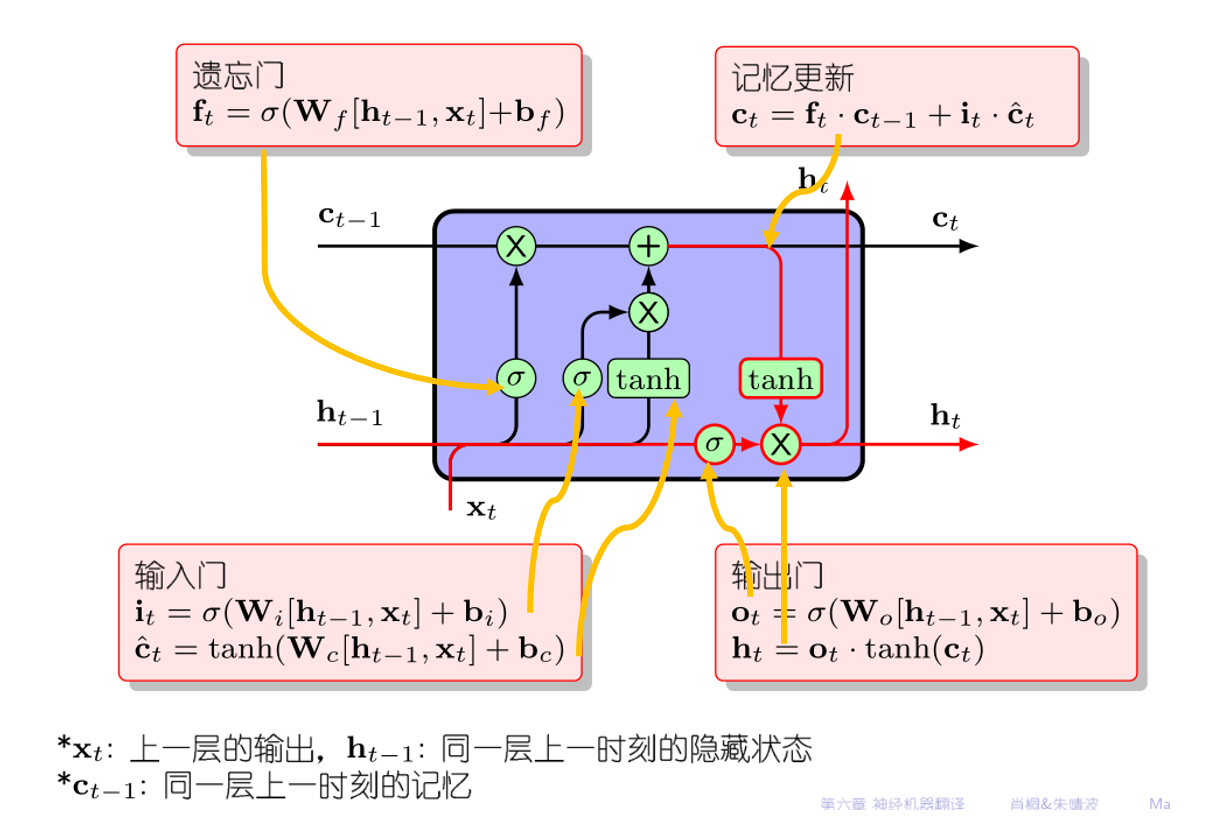
* 1. 不能调用nn.LSTM、nn.LSTMCell，可以使用nn.Linear、nn.Parameter等等搭建网络
  2. 可以参考torch.nn.LSTM的计算公式、可以仿照其输入输出，官方文档： https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html#torch.nn.LSTM
  3. 数据加载、模型训练的代码都是现成的，只需要完成模型搭建

## 二、网络结构设计

### 1、计算公式

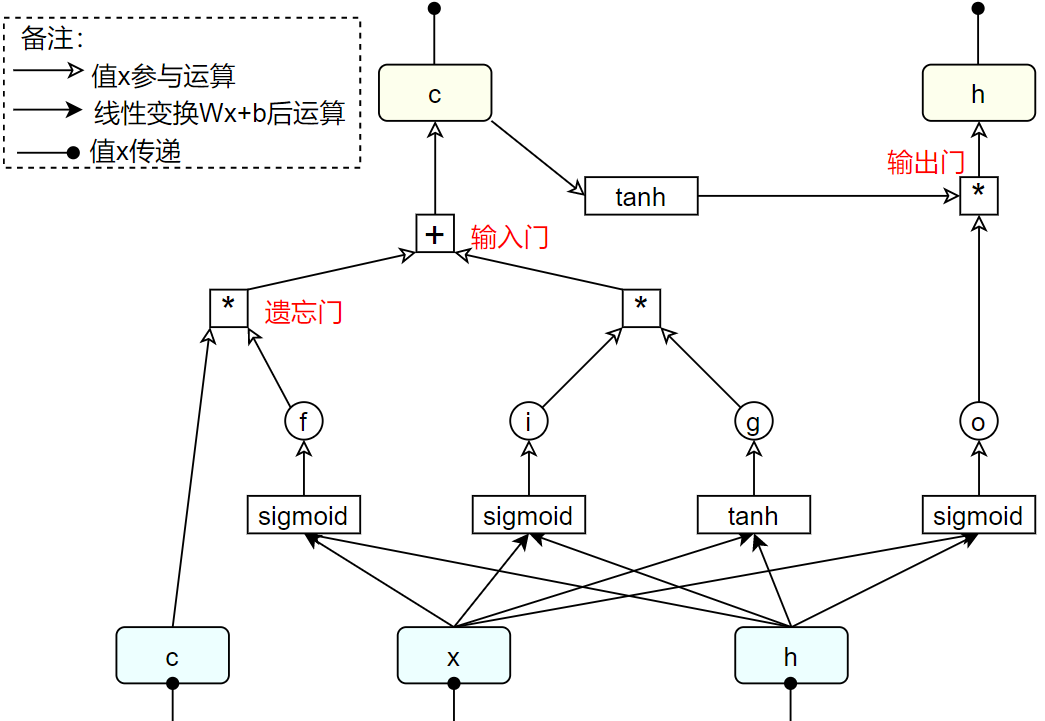
参考torch.nn.LSTM，其计算公式如下：

其中，为遗忘门，（新记忆占比）和（新记忆）共同组成输入门，为输出门，为输入（或上一层输出），为记忆。其方程与LSTM结构图对应如下：



### 2、程序实现

通过torch.nn中的Linear、Sigmoid、Tanh函数实现上述功能，则其网络结构图如下：

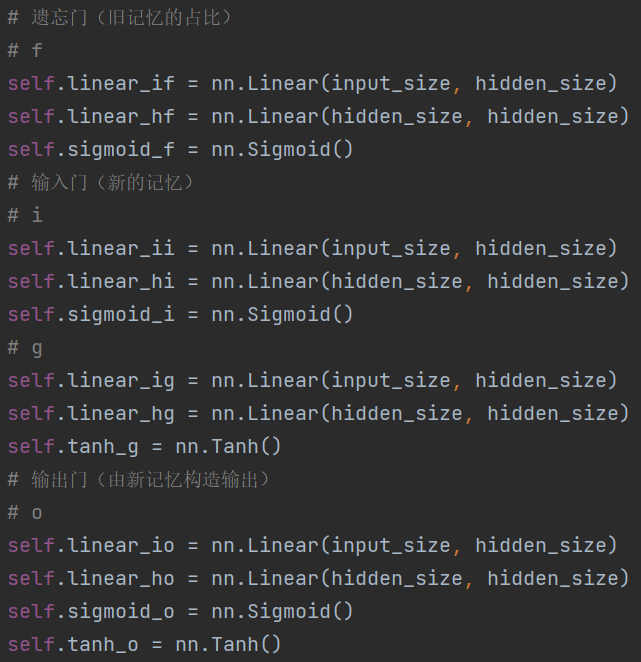


因此网络非线性，无法使用nn.Sequential( )进行封装，因此选择在自实现类LSTM时，通过在forward( )函数中手动指定计算流程。

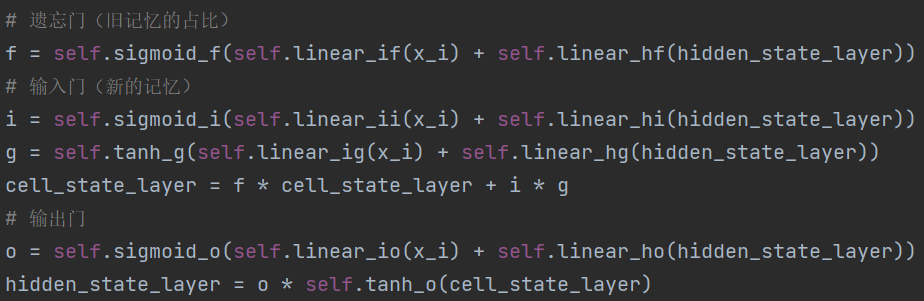
所有运算的线性层：输入输出维度分别为：input\_size、hidden\_size；

所有运算的线性层：输入输出维度都别为：hidden\_size。类的代码实现如下：

1. 在\_\_init\_\_函数中添加如下图所示的函数组件：



1. 在forward( )中，计算过程如下图所示：

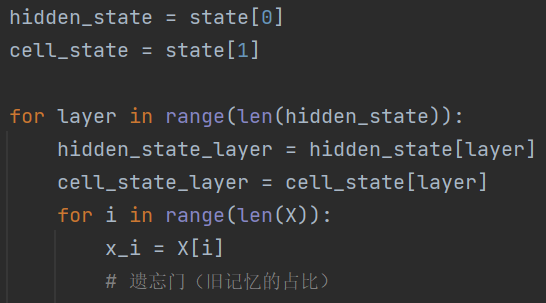


## 三、传入、传出及其他细节

### 1、输入参数及维度

在主程序（LSTMLM.py）中，通过以下语句进行LSTM调用：

outputs, (\_, \_) = self.LSTM(X, (hidden\_state.to(device), cell\_state.to(device)))

1. 参数含义：
   1. X为输入数据
   2. hidden\_state为外界定义的LSTM胞体初始化h值
   3. cell\_state为外界定义的LSTM胞体初始化c值
2. 各参数的shape，及在本实验中的值：
   1. X：[n\_step=5, batch\_size=128, embeding size=256]
   2. hidden\_state：[num\_layers=1 \* num\_directions=1, batch\_size=128, n\_hidden=256]
   3. cell\_state：[num\_layers=1 \* num\_directions=1, batch\_size=128, n\_hidden=256]
3. 各维度含义：
   1. n\_step一层LSTM的循环个数，即每次使用的句子长度（单词个数为5）
   2. embeding size为外界定义的LSTM胞体中间参数的维度，即LSTM的hidden\_size
   3. batch\_size为批训练时，一次所用数据个数
   4. num\_layers为LSTM层数
   5. num\_directions标识LSTM方向，等于1时为单向，等于2时为双向
4. 数据类型分别为(tensor, tuple(tensor, tensor))，因此，在自实现的LSTM类中，其forward( )函数进行如下处理：

### 2、运行设备统一

因为在主程序中可能会将网络及各参数运行设备设置为cpu（即CPU）或cuda:0（即GPU），因此自实现的该LSTM网络也应该运行在相同的设备上。

1. pytorch的选取原则：

首先使用nn.LSTM进行测试。

因为在主程序中hidden\_state、cell\_state的运行设备为CPU，而X的运行设备为GPU（实验所用电脑为支持cuda运行的环境，代表更广泛的情景），通过测试发现，若使用如下语句进行运行会发出设备不统一的报错。

outputs, (\_, \_) = self.LSTM(X, (hidden\_state, cell\_state))

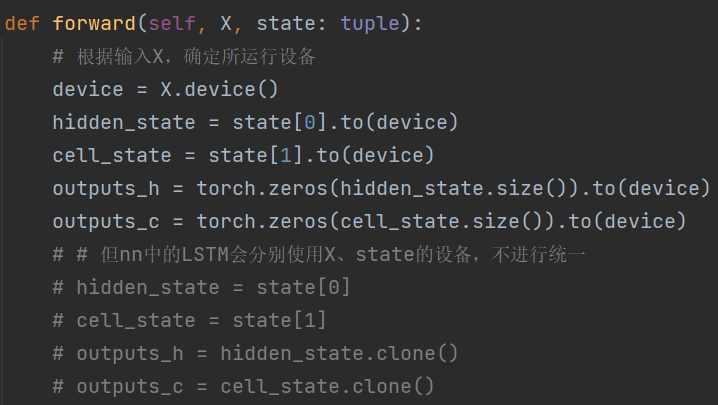
也即pytorch官方默认使用X、hidden\_state、cell\_state他们各自传入时的设备。

1. 自实现的选取原则：

但考虑到X、hidden\_state、cell\_state若有运行设备不统一的情况无法运行，这样需要编程人员手动显式地在调用前将设备统一。

因此为了降低编程人员的负担，在自实现LSTM时决定在胞体内将所有变量都设置到X所在的设备上。

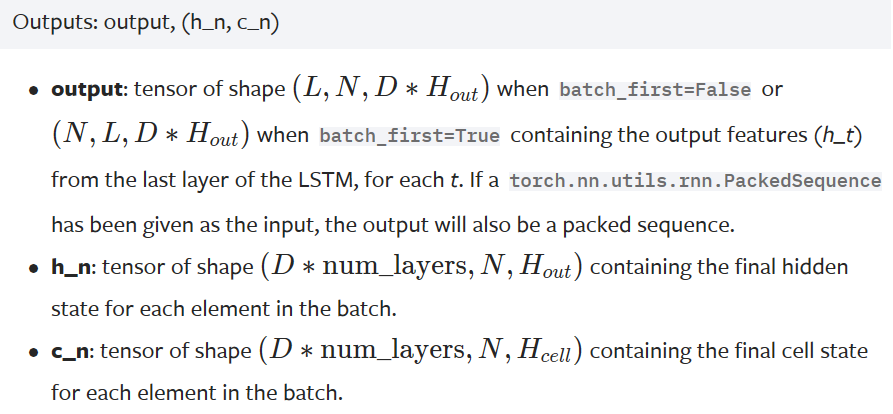
forward( )函数更新为下：



### 3、输出参数及维度

1. 参数含义：

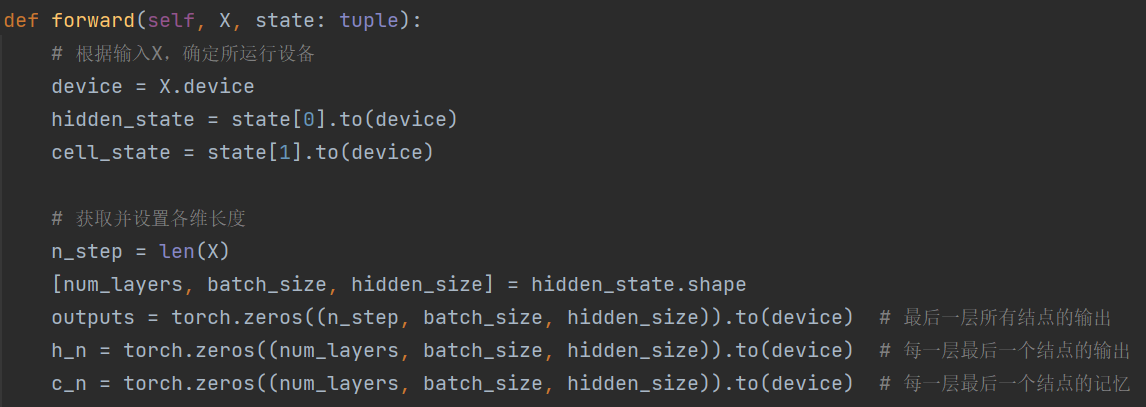
参考pytorch官网给出的LSTM网络的输出：



即：

* 1. output保存了最后一层，每个time step的输出h，如果是双向LSTM，每个time step的输出h = [h正向, h逆向] (同一个time step的正向和逆向的h连接起来)。第一维表示序列长度，第二维表示一批的样本数(batch)，第三维是 hidden\_size(隐藏层大小) \* num\_directions。
  2. h\_n保存了每一层，最后一个time step的输出h，如果是双向LSTM，单独保存前向和后向的最后一个time step的输出h。
  3. c\_n与h\_n一致，只是它保存的是c的值。

1. 在自定义的LSTM结点实现时，使用outputs、h\_n、c\_n分别存储上述参数output、h\_n、c\_n。n\_step、num\_layers、 batch\_size、hidden\_size值获取过程，以及outputs、h\_n、c\_n定义如下更新后代码所示：



1. outputs、h\_n、c\_n值的获取，如下更新后代码所示：



### 4、未传state时的默认初始化

在主程序（LSTMLM.py）中，还可通过以下语句进行LSTM调用：

outputs, (\_, \_) = self.LSTM(X)

1. 因此需要添加在未传入LSTM胞体默认状态state时，实现默认初始化，因此将forward( )函数state的默认值设为None
2. 此时num\_layers、hidden\_size因为未传入state，其值的获取途径也需要做相应改变。可放入初始化函数\_\_init\_\_当中获取
3. num\_layers的值也因此需要添加在\_\_init\_\_的参数列表，默认值为1

修改后的代码如下：



## 四、结果

1. 使用自定义的LSTM层时，使用以下命令均可成功运行，如图所示：

outputs, (\_, \_) = self.LSTM(X) （图1.4.1）

outputs, (\_, \_) = self.LSTM(X, (hidden\_state, cell\_state)) （图1.4.2）

图1.4.1 使用self.LSTM(X)，使用默认初始化方法

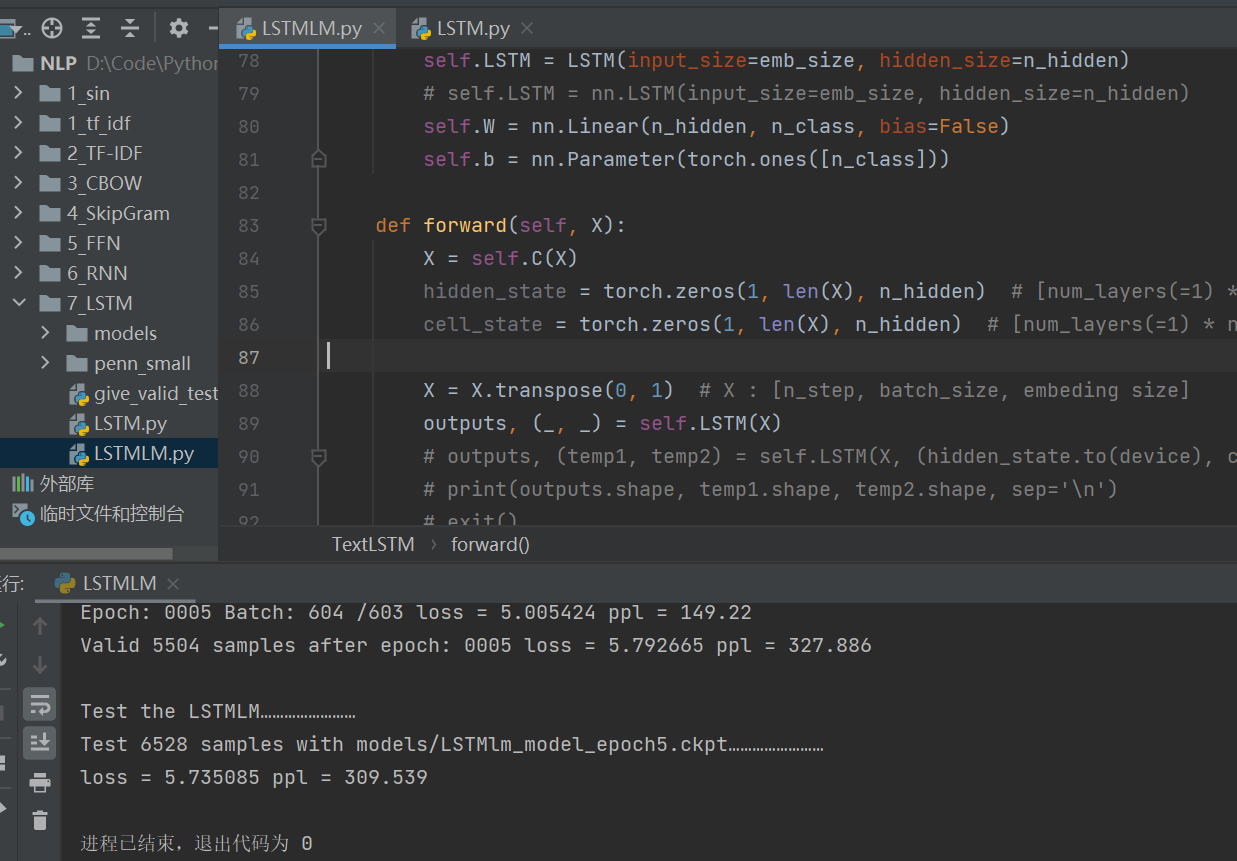
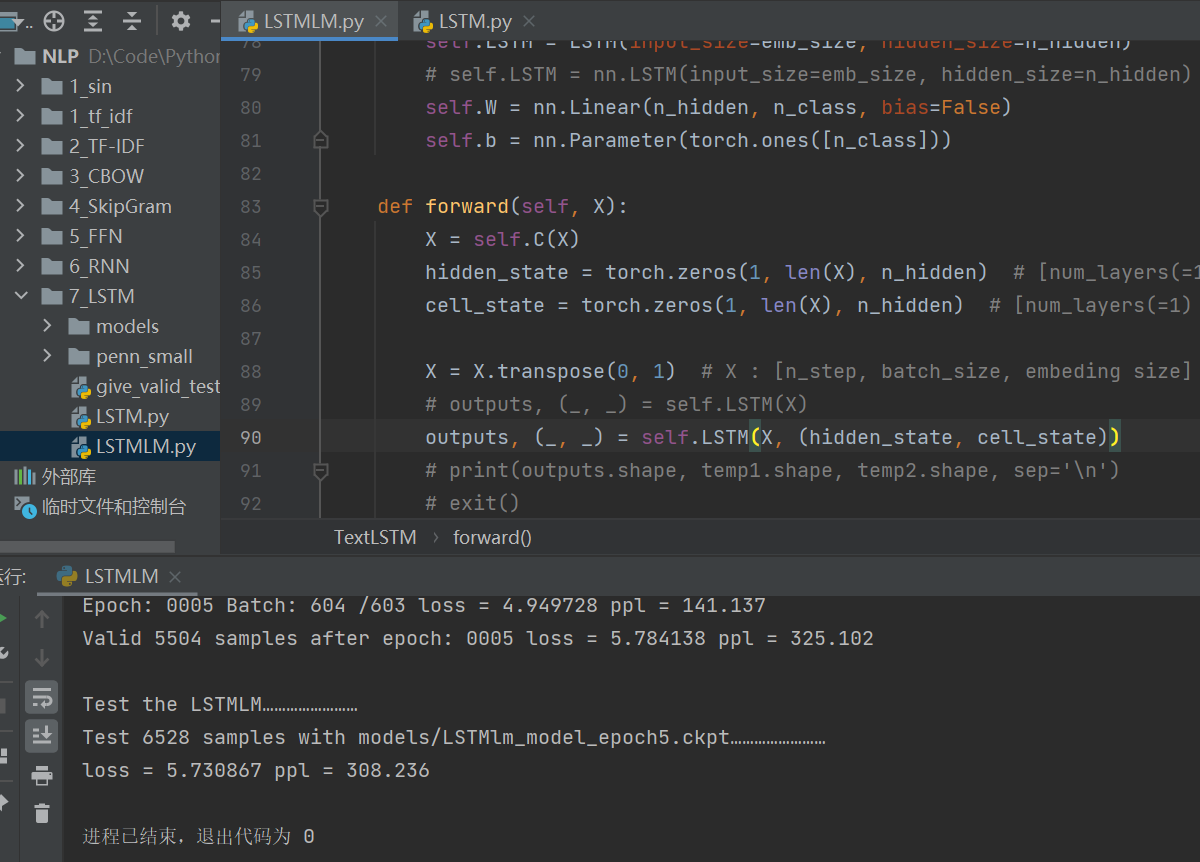


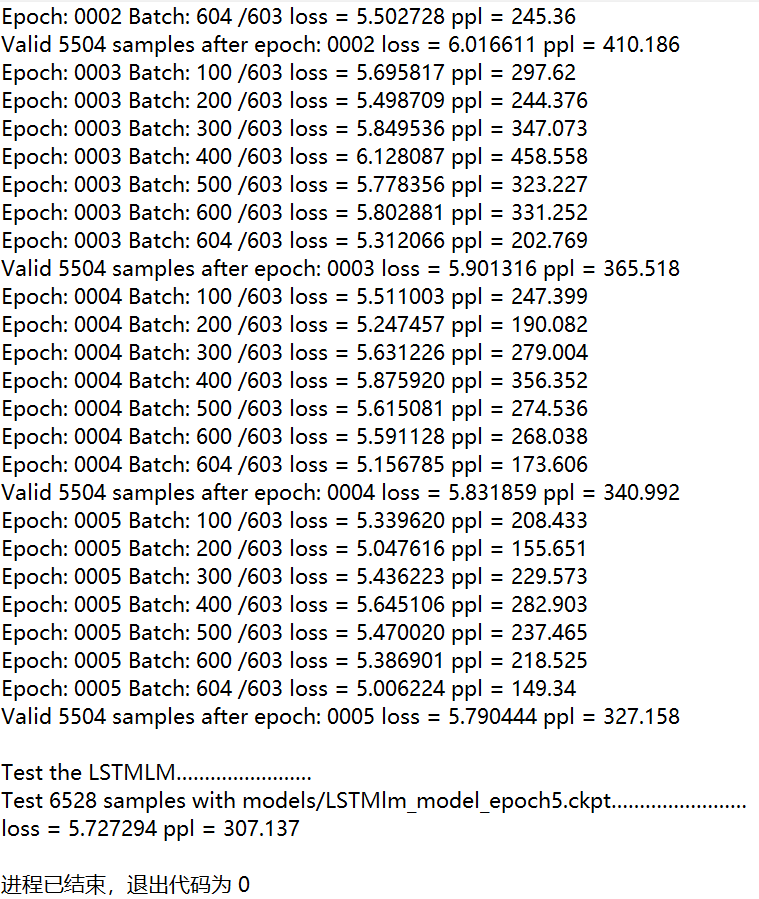
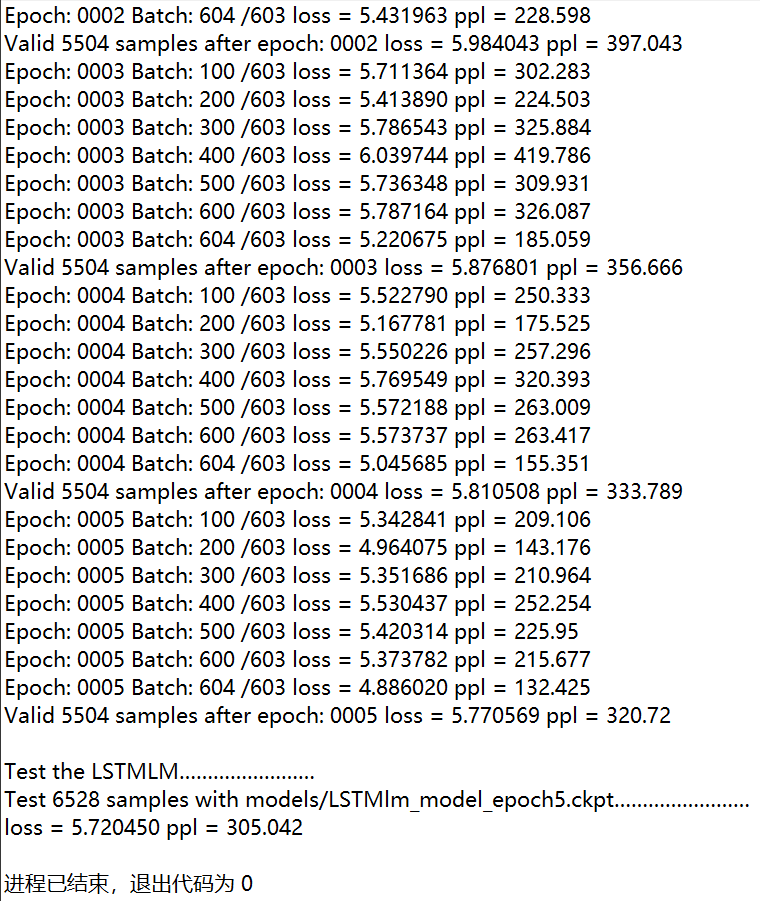
图1.4.1 使用self.LSTM(X, (hidden\_state, cell\_state))，使用外部参数



1. 通过多次运行测试，将使用自实现的LSTM和nn.LSTM的两种情况进行比较，发现自实现的LSTM和nn.LSTM在最终（epoch=5的训练情况下）运行后，考虑到波动性将多次测试所得值进行综合计量，其ppl值基本一致。在此训练情况下均在大致区间内。

如下图所示，左侧图1.4.3为使用nn.LSTM进行测试最终所得结果，右侧图1.4.4为使用自定义的LSTM进行测试最终所得结果。

图1.4.3 使用nn.LSTM所得结果 图1.4.4 使用自定义的LSTM所得结果



# 提高内容 搭建双层LSTM网络

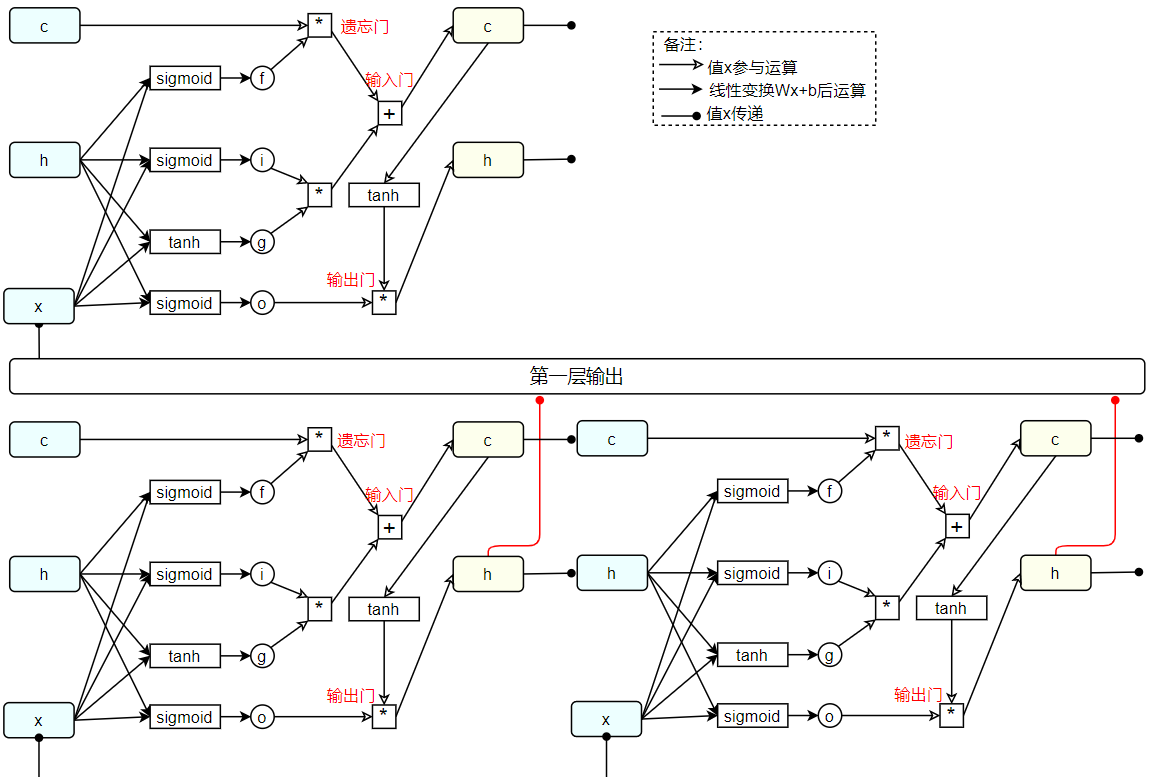
## 一、题目

在自己搭建出来的LSTM网络基础上，实现双层LSTM网络。

## 二、网络结构设计

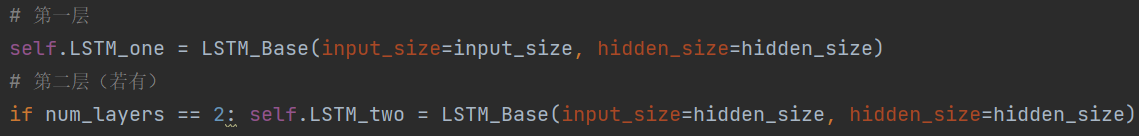
### 1、流程图

使用“基础内容 搭建LSTM网络”部分设计单层LSTM网络，进行如下串联叠加。其中第一层的输入即为传入的X，第二层的输入为第一层的输出（h\_n部分）。



### 2、程序实现

1. 将“基础内容 搭建LSTM网络”重原有的LSTM类改为LSTM\_Base类
2. 添加LSTM类，在\_\_init\_\_中定义包含如下组件：



1. 其中第一层的输入即为在主程序中调用时传入的X，第二层的输入为第一层的输出（h\_n部分），即在forward( )函数中通过如下过程实现：

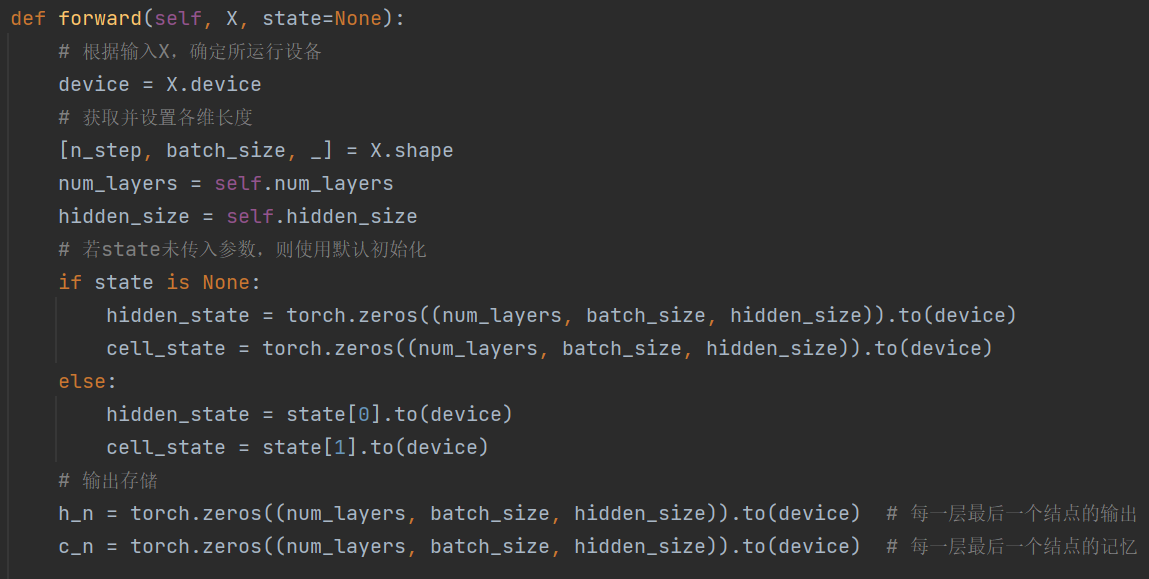
outputs1, (h\_n1, c\_n1) = self.LSTM\_one(X, (hidden\_state[0], cell\_state[0]))

outputs2, (h\_n2, c\_n2) = self.LSTM\_two(outputs1, (hidden\_state[1], cell\_state[1]))

## 三、传入、传出及其他细节

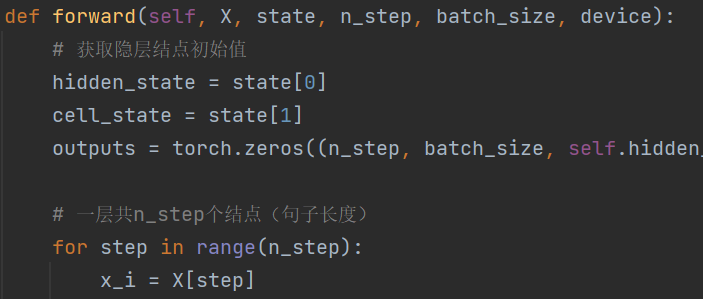
### 1、传入

1. 将运行设备统一、未传state时的默认初始化、输入参数维度处理，等工作都从原有的单层网络类中（LSTM\_Base）移出至新的LSTM类中，如下：



1. 对于传入的state参数，其最外层的num\_layers维在LSTM类中完成拆分。在本实验中，以第一层为例，在调用LSTM内部的LSTM\_Base的forward( )时，传入参数的shape变为：
   * 1. X：[n\_step=5, batch\_size=128, embeding size=256]
     2. hidden\_state：[batch\_size=128, n\_hidden=256]
     3. cell\_state：[batch\_size=128, n\_hidden=256]

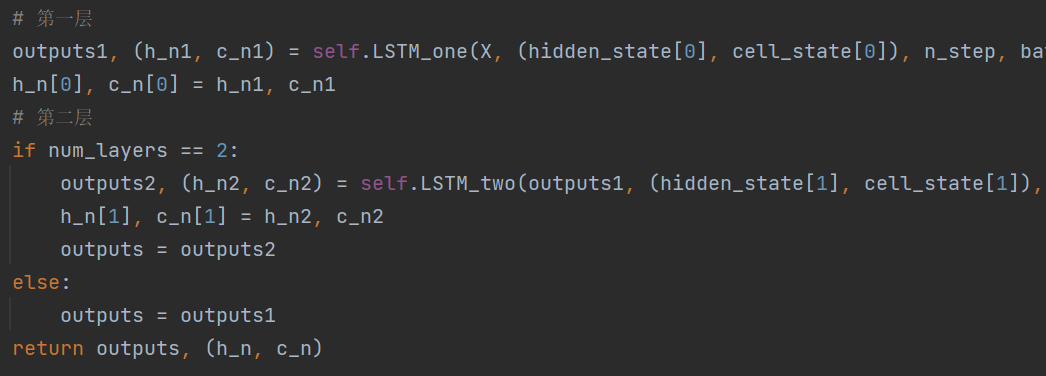
在LSTM\_Base中，其forward( )函数变为如下：



### 2、传出

1. 在LSTM类中，两层（num\_layers=2时）网络以串联方式连接，先获取第一层输出outputs1, (h\_n1, c\_n1)，再将output1作为第二层的输入传入进行计算。
2. 当网络层数为2时，对于LSTM网络总体，其输出中的outputs即为最后一层网络的outputs2，其中的h\_n、c\_n分别将相应位置取值为各层网络输出中的h\_n1、c\_n1，h\_n2、c\_n2。

如下图所示：



## 四、结果

1. 使用自定义的LSTM层时，使用以下命令均可成功运行，如图所示：

outputs, (\_, \_) = self.LSTM(X) （图1.4.1）

outputs, (\_, \_) = self.LSTM(X, (hidden\_state, cell\_state)) （图1.4.2）

图1.4.1 使用self.LSTM(X)，使用默认初始化方法

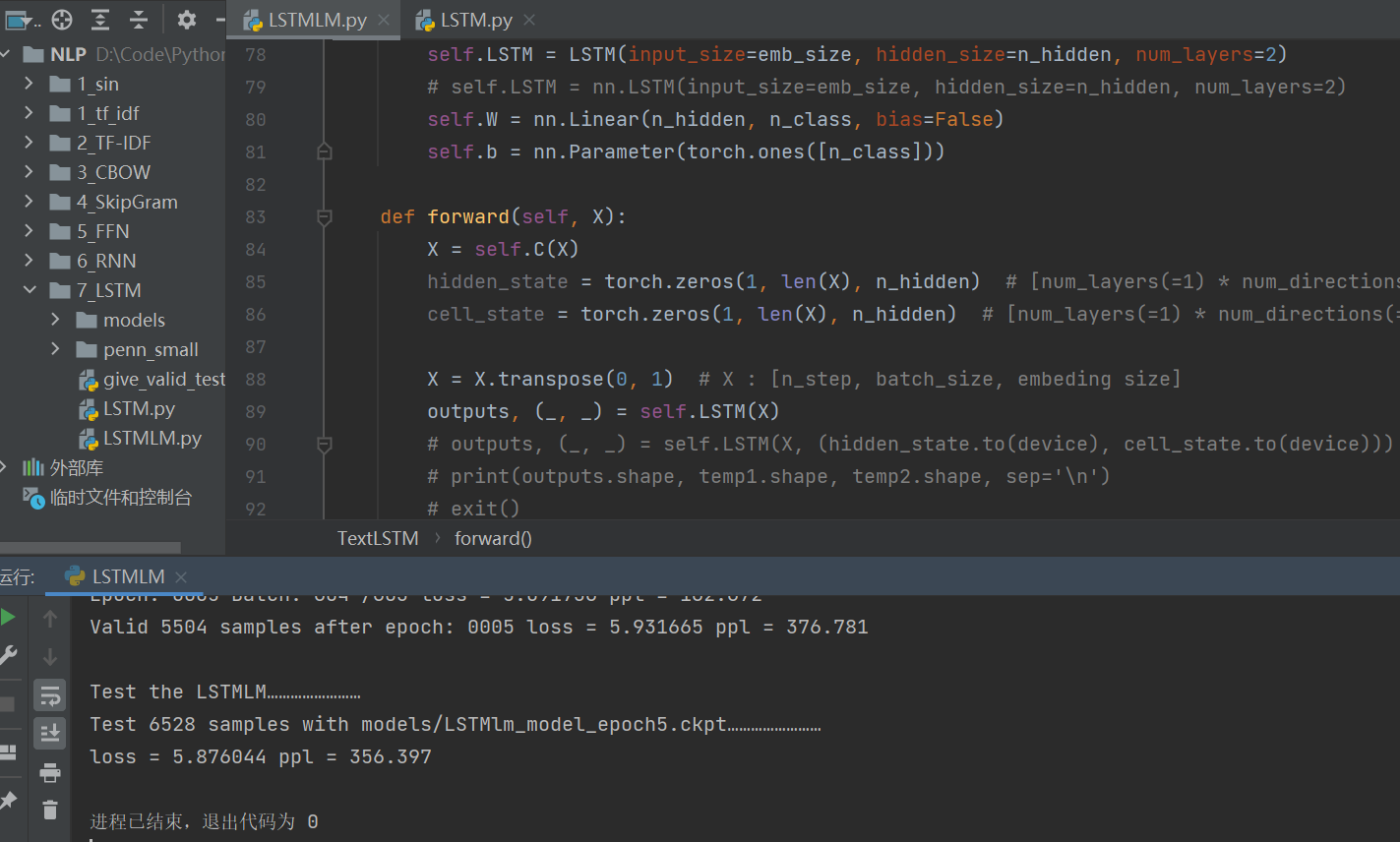
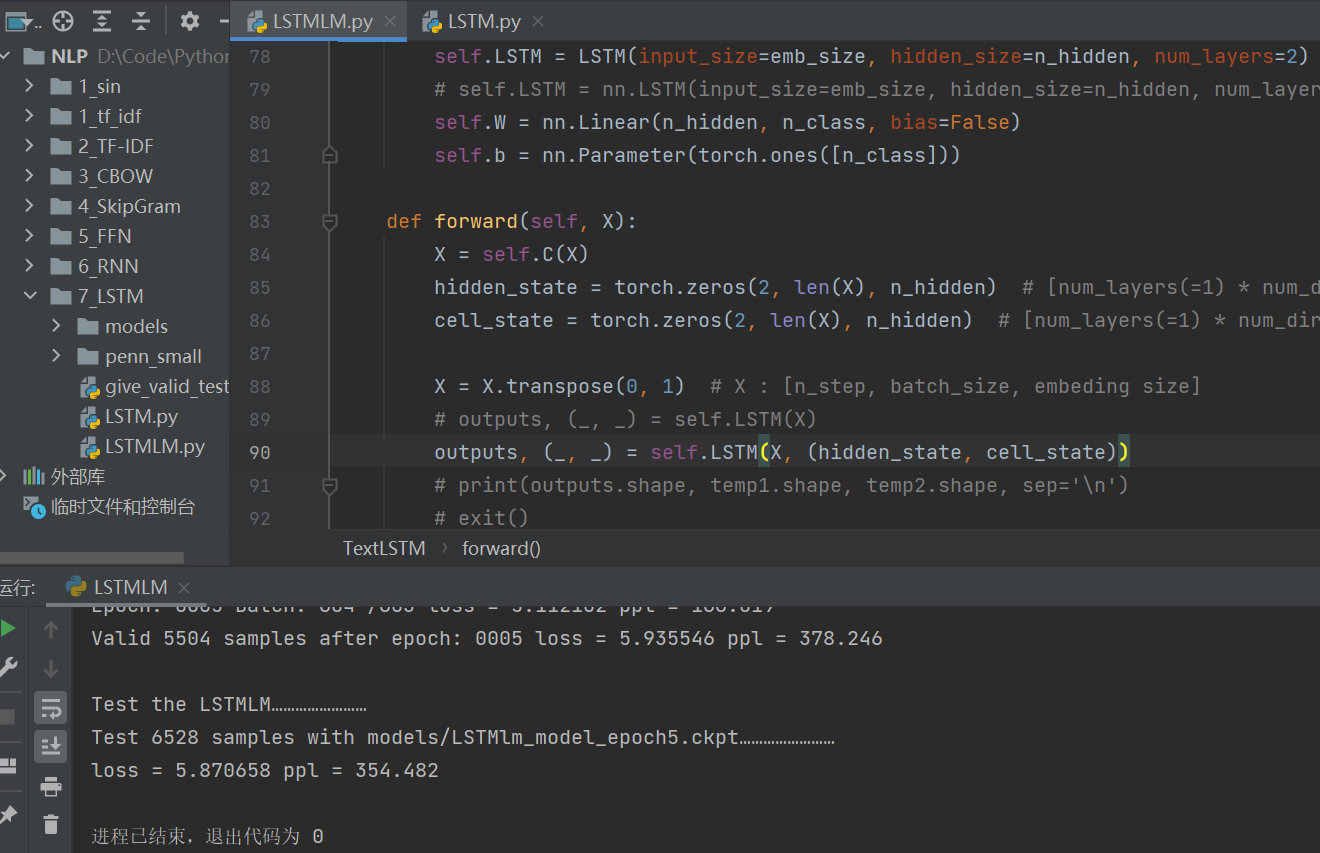


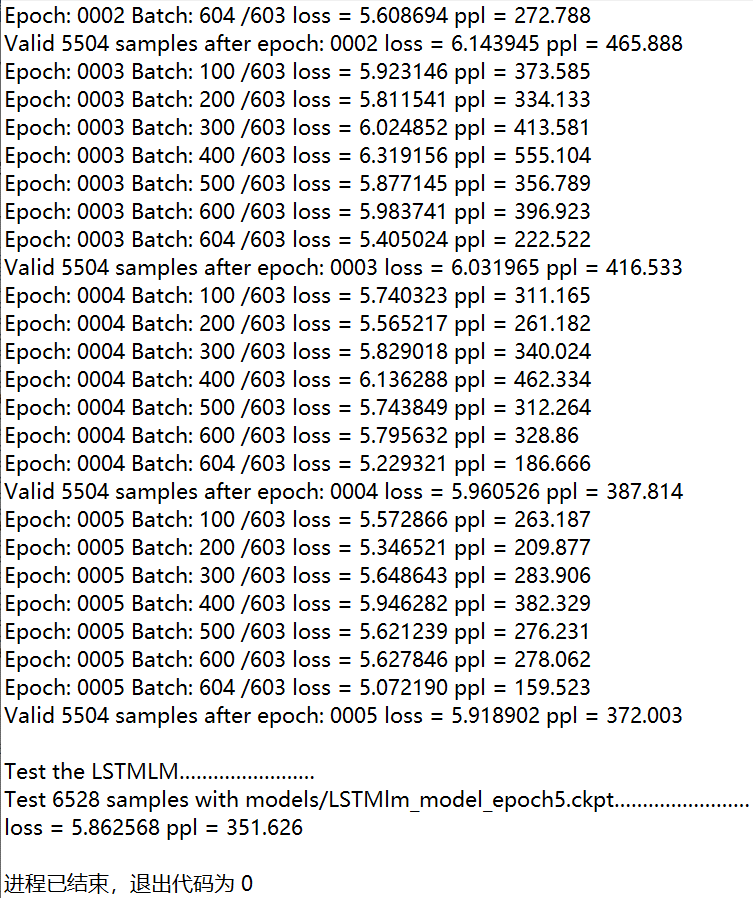
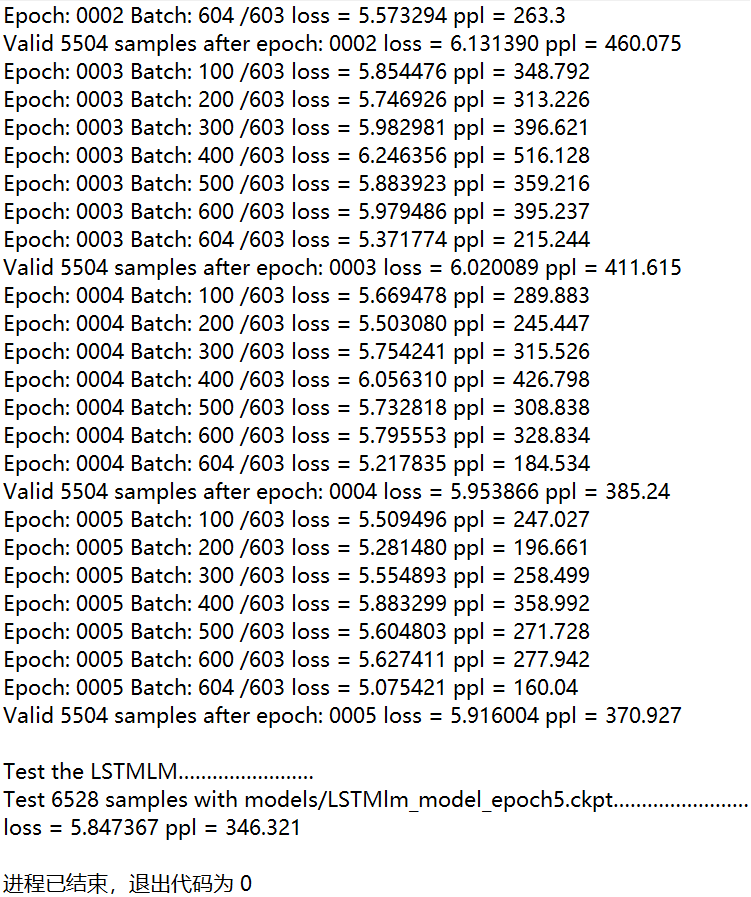
图1.4.1 使用self.LSTM(X, (hidden\_state, cell\_state))，使用外部参数



1. 将使用自实现的LSTM和nn.LSTM的两种情况进行10次运行比较，发现自实现的LSTM和nn.LSTM在最终（epoch=5的训练情况下）运行后，考虑到波动性将多次测试所得值进行综合计量，自实现LSTM与nn.LSTM的ppl值大致相同。在此训练情况下自实现LSTM在大致区间内；自实现LSTM在大致区间内。

如下图所示，左侧图1.4.3为使用nn.LSTM进行测试最终所得结果，右侧图1.4.4为使用自定义的LSTM进行测试最终所得结果。

图1.4.3 使用nn.LSTM所得结果 图1.4.4 使用自定义的LSTM所得结果



# 附录

## 一、主程序修改（使用说明）

* 1. 需要添加语句“from LSTM import LSTM”以使用自定义LSTM
  2. 使用如下语句进行使用，其中input\_size、hidden\_size为必须传入的形参，num\_layers可不传入，默认值为1。

LSTM(input\_size=emb\_size, hidden\_size=n\_hidden, num\_layers=2)

* 1. 使用如下语句进行调用，其中从外界指定state初始值的调用其X与(hidden\_state, cell\_state)可在不同运行设备上，会自动转为在X所在设备。

使用默认state：outputs, (h\_n, c\_n) = self.LSTM(X)

指定初始state：outputs, (h\_n, c\_n) = self.LSTM(X, (hidden\_state, cell\_state))

## 二、LSTM.py源码

import torch

from torch import nn

# 自实现LSTM结构单层基础

class LSTM\_Base(nn.Module):

# 输入维度、输出维度

def \_\_init\_\_(self, input\_size, hidden\_size):

super(LSTM\_Base, self).\_\_init\_\_()

# 遗忘门（旧记忆的占比）

# f

self.linear\_if = nn.Linear(input\_size, hidden\_size)

self.linear\_hf = nn.Linear(hidden\_size, hidden\_size)

self.sigmoid\_f = nn.Sigmoid()

# 输入门（新的记忆）

# i

self.linear\_ii = nn.Linear(input\_size, hidden\_size)

self.linear\_hi = nn.Linear(hidden\_size, hidden\_size)

self.sigmoid\_i = nn.Sigmoid()

# g

self.linear\_ig = nn.Linear(input\_size, hidden\_size)

self.linear\_hg = nn.Linear(hidden\_size, hidden\_size)

self.tanh\_g = nn.Tanh()

# 输出门（由新记忆构造输出）

# o

self.linear\_io = nn.Linear(input\_size, hidden\_size)

self.linear\_ho = nn.Linear(hidden\_size, hidden\_size)

self.sigmoid\_o = nn.Sigmoid()

self.tanh\_o = nn.Tanh()

# 需要存储一些网络规模有关数据

self.hidden\_size = hidden\_size # 中间变量维度（输出维度）

def forward(self, X, state, n\_step, batch\_size, device):

# 获取隐层结点初始值

hidden\_state = state[0]

cell\_state = state[1]

outputs = torch.zeros((n\_step, batch\_size, self.hidden\_size)).to(device) # 最后一层所有结点的输出

# 一层共n\_step个结点（句子长度）

for step in range(n\_step):

x\_i = X[step]

# 遗忘门（旧记忆的占比）

f = self.sigmoid\_f(self.linear\_if(x\_i) + self.linear\_hf(hidden\_state))

# 输入门（新的记忆）

i = self.sigmoid\_i(self.linear\_ii(x\_i) + self.linear\_hi(hidden\_state))

g = self.tanh\_g(self.linear\_ig(x\_i) + self.linear\_hg(hidden\_state))

cell\_state = f \* cell\_state + i \* g

# 输出门

o = self.sigmoid\_o(self.linear\_io(x\_i) + self.linear\_ho(hidden\_state))

hidden\_state = o \* self.tanh\_o(cell\_state)

# 将所有n\_step个结点，值保存在outputs中

outputs[step] = hidden\_state

# 在每一层的最后一个结点，值保存在outputs\_h、outputs\_c中

h\_n = hidden\_state

c\_n = cell\_state

return outputs, (h\_n, c\_n)

# 自实现双层LSTM结构（串联）

class LSTM(nn.Module):

# 输入维度、输出维度

def \_\_init\_\_(self, input\_size, hidden\_size, \*, num\_layers=1):

super(LSTM, self).\_\_init\_\_()

# 第一层

self.LSTM\_one = LSTM\_Base(input\_size=input\_size, hidden\_size=hidden\_size)

# 第二层（若有）

if num\_layers == 2:

self.LSTM\_two = LSTM\_Base(input\_size=hidden\_size, hidden\_size=hidden\_size)

# 需要存储一些网络规模有关数据

self.num\_layers = num\_layers # 网络层数

self.hidden\_size = hidden\_size # 中间变量维度（输出维度）

def forward(self, X, state=None):

# 根据输入X，确定所运行设备

device = X.device

# 获取并设置各维长度

[n\_step, batch\_size, \_] = X.shape

num\_layers = self.num\_layers

hidden\_size = self.hidden\_size

# 若state未传入参数，则使用默认初始化

if state is None:

hidden\_state = torch.zeros((num\_layers, batch\_size, hidden\_size)).to(device)

cell\_state = torch.zeros((num\_layers, batch\_size, hidden\_size)).to(device)

else:

hidden\_state = state[0].to(device)

cell\_state = state[1].to(device)

# 输出存储

h\_n = torch.zeros((num\_layers, batch\_size, hidden\_size)).to(device) # 每一层最后一个结点的输出

c\_n = torch.zeros((num\_layers, batch\_size, hidden\_size)).to(device) # 每一层最后一个结点的记忆

# 第一层

outputs1, (h\_n1, c\_n1) = self.LSTM\_one(X, (hidden\_state[0], cell\_state[0]), n\_step, batch\_size, device)

h\_n[0], c\_n[0] = h\_n1, c\_n1

# 第二层

if num\_layers == 2:

outputs2, (h\_n2, c\_n2) = self.LSTM\_two(outputs1, (hidden\_state[1], cell\_state[1]), n\_step, batch\_size, device)

h\_n[1], c\_n[1] = h\_n2, c\_n2

outputs = outputs2

else:

outputs = outputs1

return outputs, (h\_n, c\_n)