A5：循环神经网络 (20 points)

任务：构建一个用于起名字的循环神经网络

数据：8000多个英文名字

（https://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/）

目录

**[A5：循环神经网络 (20 points) 1](#_Toc23760)**

**[1、模型原理 1](#_Toc18064)**

**[2、模型设计 1](#_Toc1244)**

**[3、训练过程 2](#_Toc32642)**

[（1）遇到的问题及解决方案： 2](#_Toc13918)

[（2）训练与优化 3](#_Toc1806)

**[4、名字生成 6](#_Toc14371)**

**[5、可视化 10](#_Toc29351)**

**[6、总结与反思 14](#_Toc29761)**

1、模型原理

我们知道，给定大量的文本数据，可训练一个基于循环神经网络的语言模型，该模型可用于计算一个句子的出现概率（P(w1, w2, ..., wT)），或者根据上文中的词推断某个词作为下一个词的出现概率（P(wt｜w1, w2, ..., wt-1)）。

基于RNN生成名字是对给出单词推断在某个类型中最可能出现的下个单词，直到发现最可能出现的为结束符时结束。要实现生成名字的多样性，又同时保证生成名字不过长，或者说像一个名字，要求我们模型预测具有可靠性，选取生成具有一定程度的随机性，因而在生成时考虑加入若生成为同一名字，则反复一定次数生成，直到获取到不同名字。

RNN的底层原理在于：通过一个循环单元保存之前时间步的信息，使得模型能在每个时间步都接收输入数据和来自上一时间步的隐藏状态，然后产生输出和新的隐藏状态，以此保持记忆。

2、模型设计

设计模型包括三个线性层i2h、i2o和o2o，用于将输入数据映射到隐藏状态、将输入数据映射到输出、将隐藏状态和输出合并到输出。dropout用于防止过拟合，softmax 对输出进行softmax操作。

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

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

self.hidden\_size = hidden\_size

# 三层核心结构

self.i2h = nn.Linear(n\_categories + input\_size + hidden\_size, hidden\_size)

self.i2o = nn.Linear(n\_categories + input\_size + hidden\_size, output\_size)

self.o2o = nn.Linear(hidden\_size + output\_size, output\_size)

#增加混乱和采样多样性防止过拟合，softmax对输出归一化

self.dropout = nn.Dropout(0.1)

self.softmax = nn.LogSoftmax(dim=1)

forward完成了循环单元的实现，定义了数据在模型中的前向传播过程，其中hidden变量起到了保存上一时间步信息的作用。

在每个时间步，模型接收一个类别信息（category）、一个输入（input）和上一时间步的隐藏状态（hidden），拼接成一个输入向量input\_combined，然后通过i2h层和i2o层分别映射到隐藏状态和输出合并成output\_combined，通过o2o层映射到最终的输出（如模型结构设计）。最后，对输出进行dropout，然后经过softmax归一化，最终返回输出和更新后的隐藏状态。

def forward(self, category, input, hidden):

# 合并所有输入

input\_combined = torch.cat((category, input, hidden), 1)

# 得到隐藏层

hidden = self.i2h(input\_combined)

# 得到当前隐藏的字母信息

output = self.i2o(input\_combined)

# 合并

output\_combined = torch.cat((hidden, output), 1)

output = self.o2o(output\_combined)

output = self.dropout(output)

output = self.softmax(output)

return output, hidden

初始化隐藏状态，返回一个全零的张量。

def initHidden(self):

return torch.zeros(1, self.hidden\_size)

rnn = RNN(n\_letters,128,n\_letters)#n\_letters = len(all\_letters)+1 # 添加EOS结束位

3、训练过程

（1）遇到的问题及解决方案：

防止数值溢出、梯度爆炸，增大学习率，解决loss居高不下的问题：

......

output,hidden = rnn(category\_tensor,input\_line\_tensor[i],hidden)

# 使用 torch.clamp 防止数值溢出

clamped\_output = torch.clamp(output, min=-1e6, max=1e6)

l = criterion(clamped\_output,target\_line\_tensor[i])

......

loss.backward()

# 梯度裁剪

torch.nn.utils.clip\_grad\_norm\_(rnn.parameters(), max\_norm=1.0)

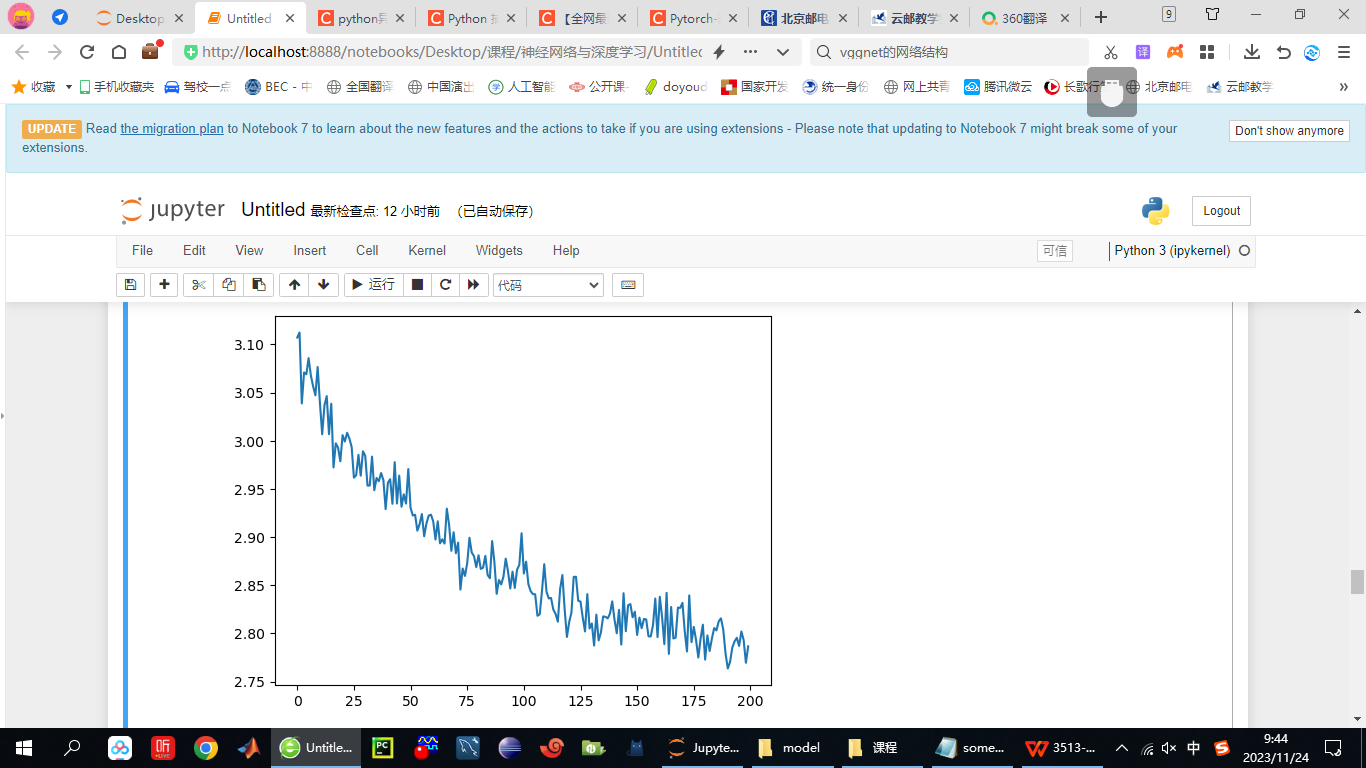
optimizer.step()

......

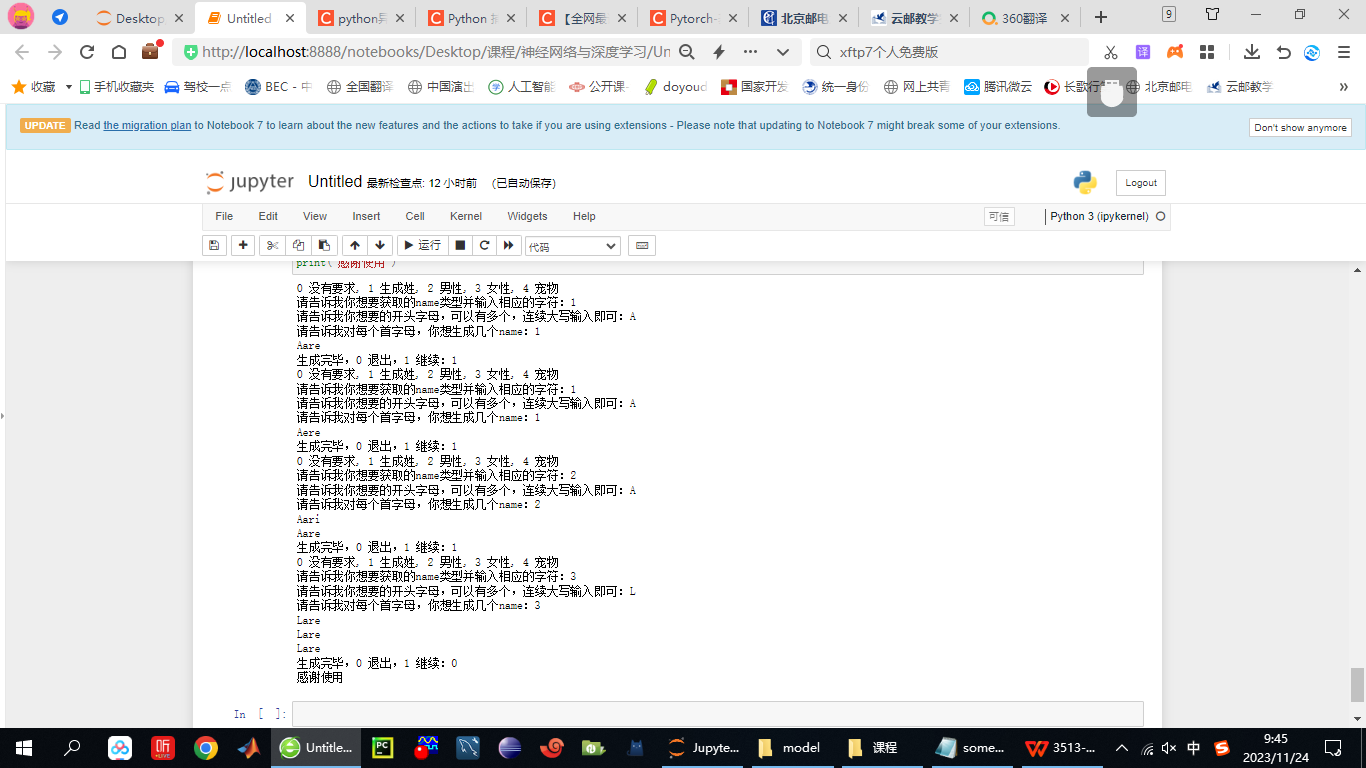
（2）训练与优化

Lr=0.001，不带动量的SGD

22m 32s

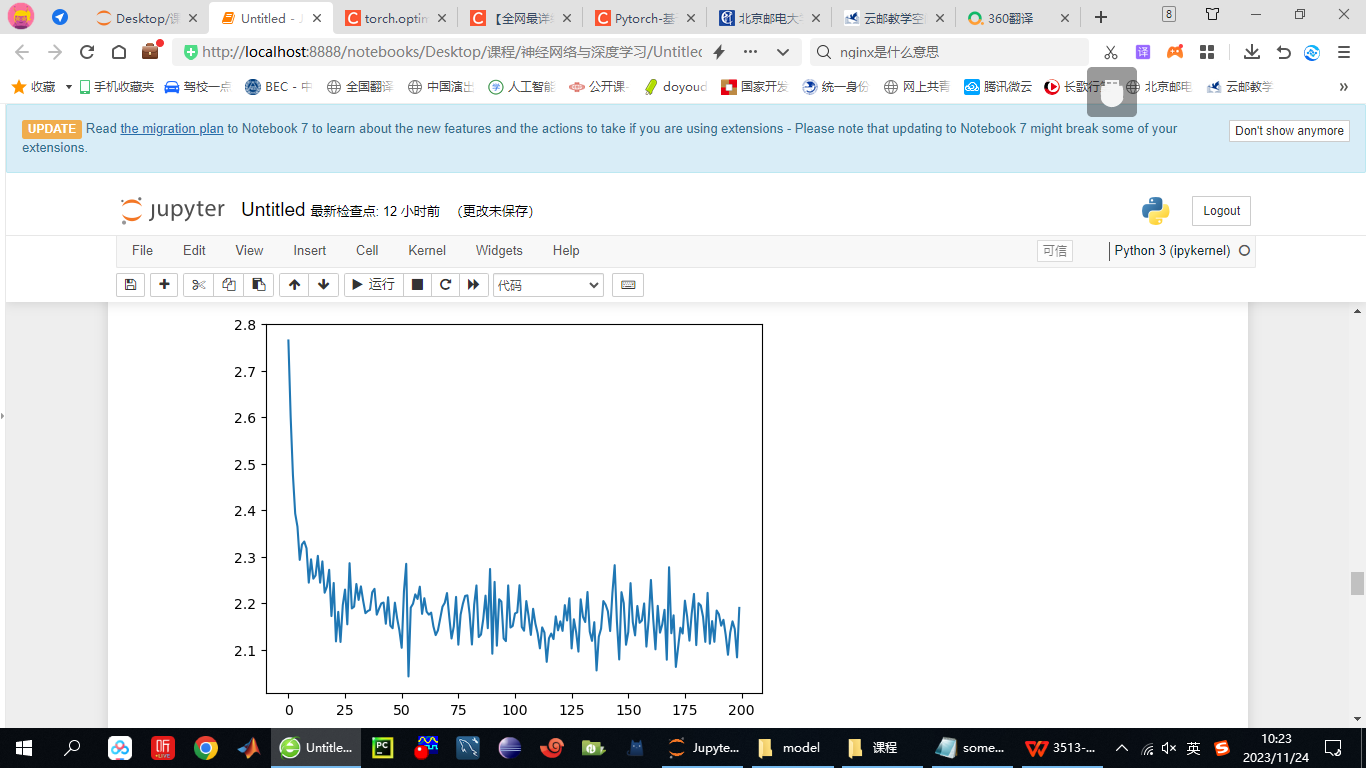


测试输入：1A111A112A213L30



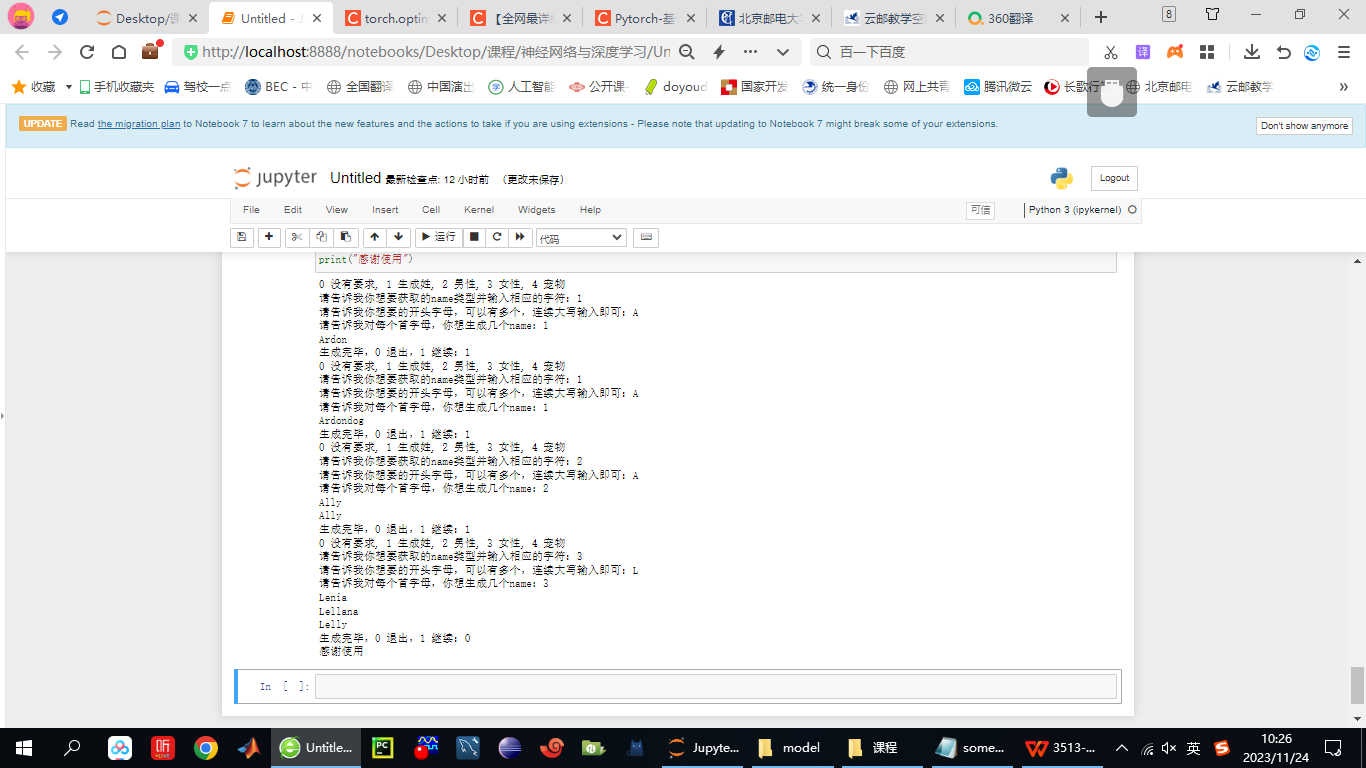
生成名字重复度很高，但能基本实现名字生成的任务。

将模型重命名为rnnmodel-11240941.pth，改为Adam优化器，训练新的模型：



可以看到，相较SGD，明显收敛更快，虽然训练结束总耗时29m 37s相较更长，但达到同等损失水平的时间更早。

同样测试输入，可以发现直观感知好很多，名字多样性增加了。

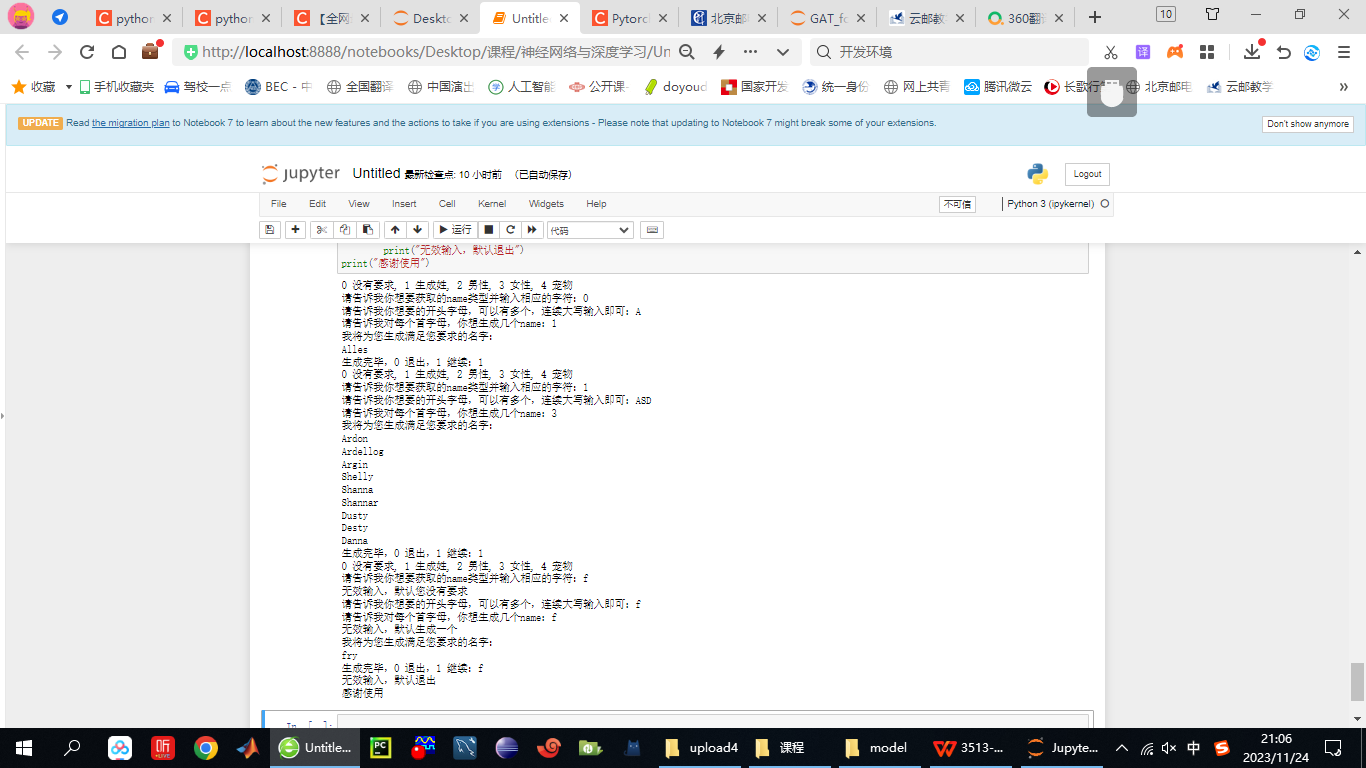


4、名字生成

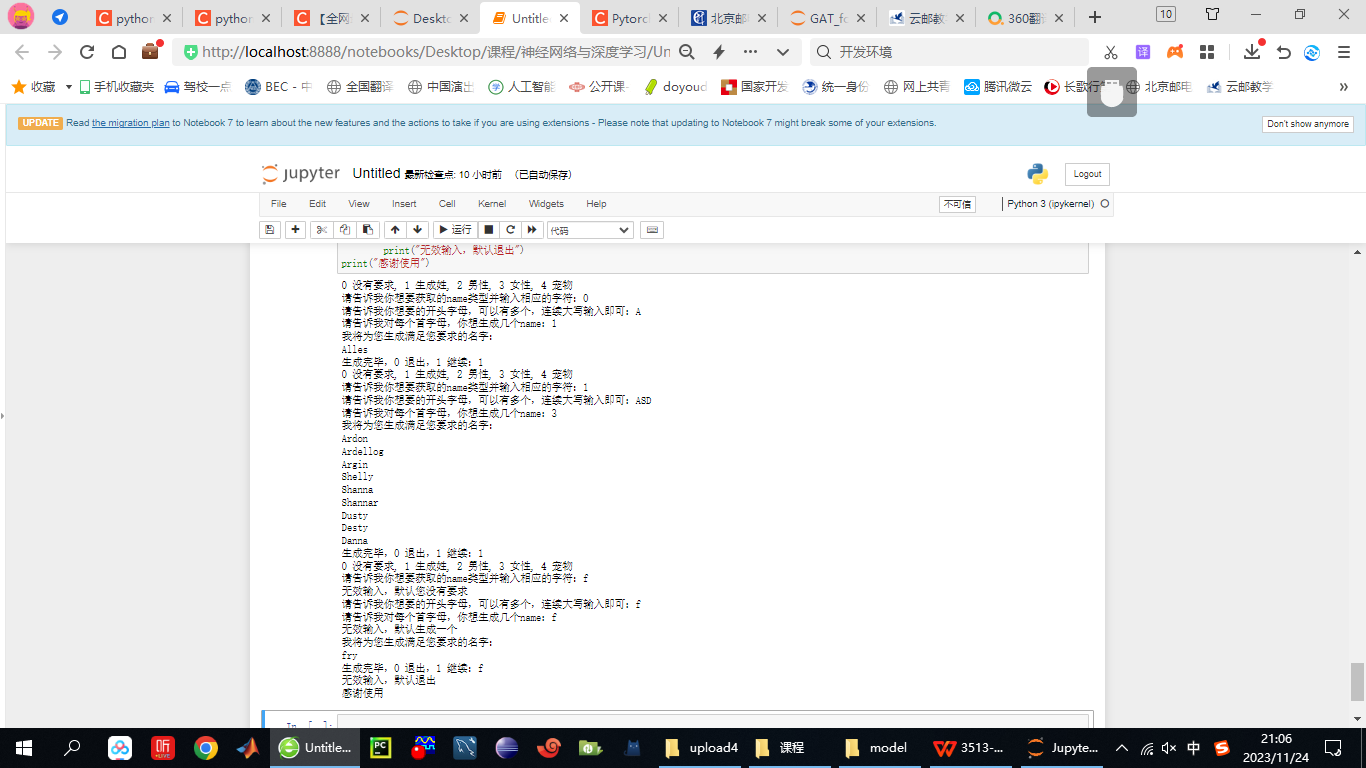
（基于首字母）

测试示例：

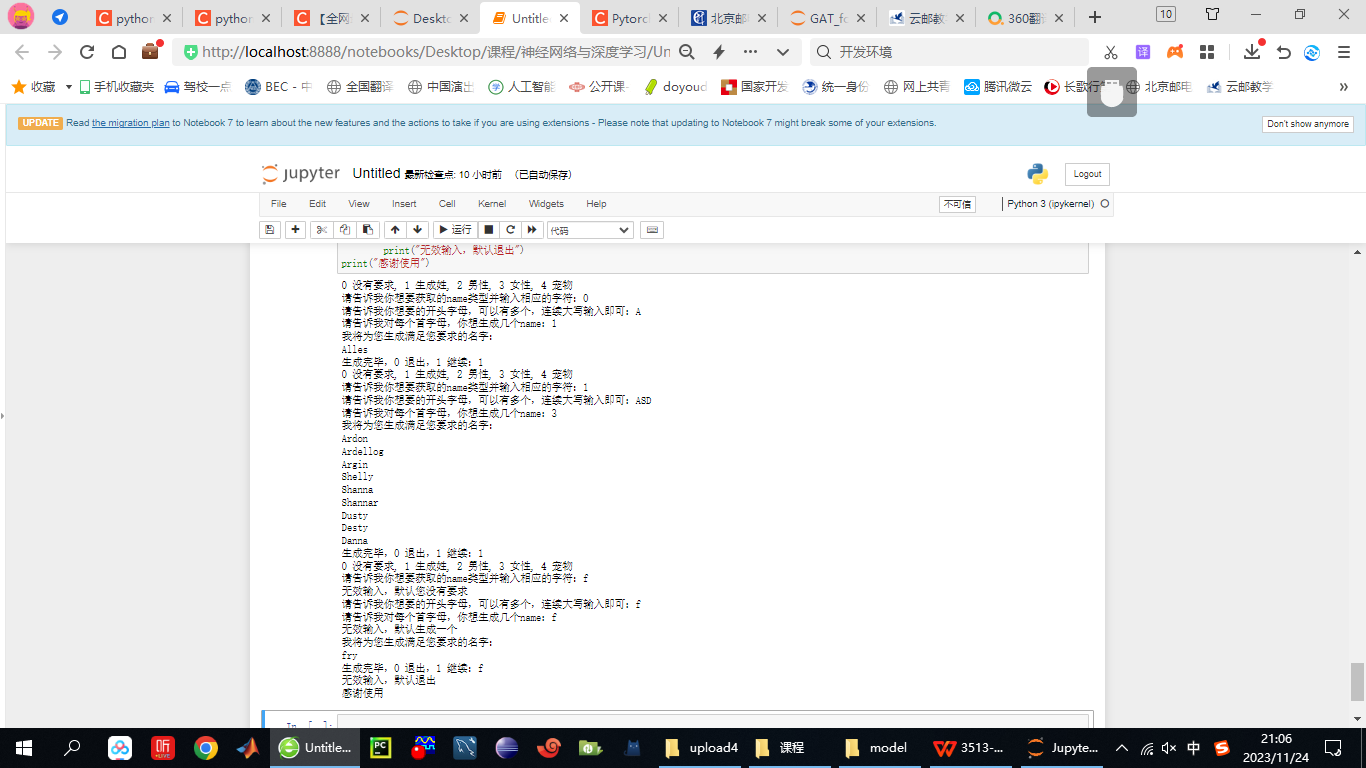
输入正常，且只对一个首字母生成一个名字：



输入正常，且对多个首字母生成多个名字：



输入异常：



（基于结尾）

若不对神经网络重新训练，利用训练好的rnn，实现基于结尾的生成思路：指定从零向量开始，正向推理直到生成指定字符且指定字符的下一个字符是结束符。

......

# 用零向量表示生成名字的开始

input = torch.zeros(1, 1, n\_letters)

hidden = rnn.initHidden()

output\_name = ''

# 将给定的结尾字母转换为对应的 one-hot 张量（后面用于推导下一个是否是结束符）

#end\_letter\_tensor = inputTensor(end\_letter)

for i in range(max\_length):

#这是为了防止第一次从零向量推导letter索引出错

letter = 0

while letter == 0:

output, hidden = rnn(category\_tensor, input[0], hidden)

\_, pred = torch.max(output.data, dim=1)

topi = pred.item()

try:

letter = all\_letters[topi]

pass

except:

pass

if i == 0:

#确保名字首字母是大写的

letter = letter.title()

#如果我们推导到给定的结尾字符了

elif letter == end\_letter:

#继续推导下一位

e\_output, e\_hidden = rnn(category\_tensor, end\_letter\_tensor[0], hidden)

\_, e\_pred = torch.max(e\_output.data, dim=1)

e\_topi = e\_pred.item()

#如果下一位是结束符

if e\_topi == n\_letters - 1:

output\_name = output\_name + letter

#生成完毕

Break

#如果我们生成了大写字母，马上跳出生成

elif letter.isupper():

break

output\_name = output\_name + letter

# 当前预测的字母作为下一个时刻的输入

input = inputTensor(letter)

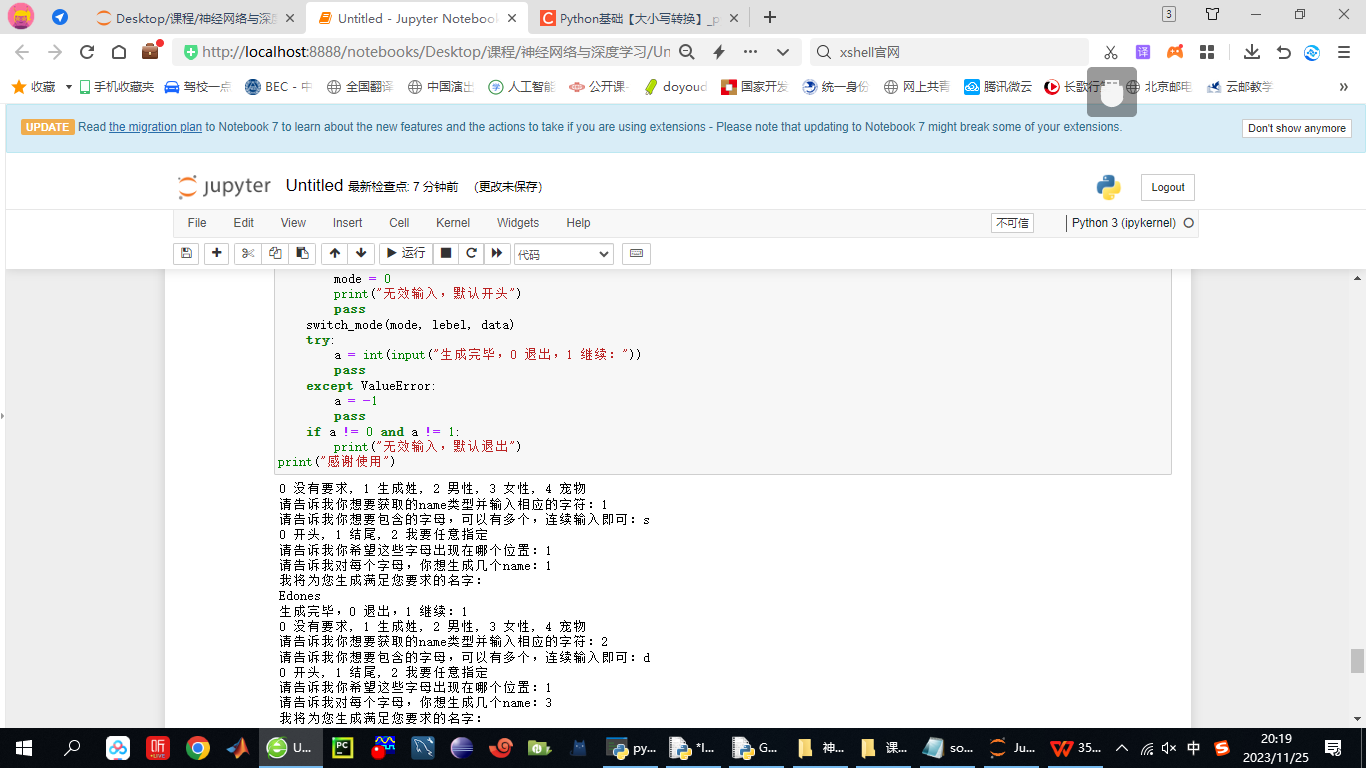
#print(type(output\_name))

#因为我们跳出了一些错误情况，这时候检测一下，不符合要求的我们就不要。

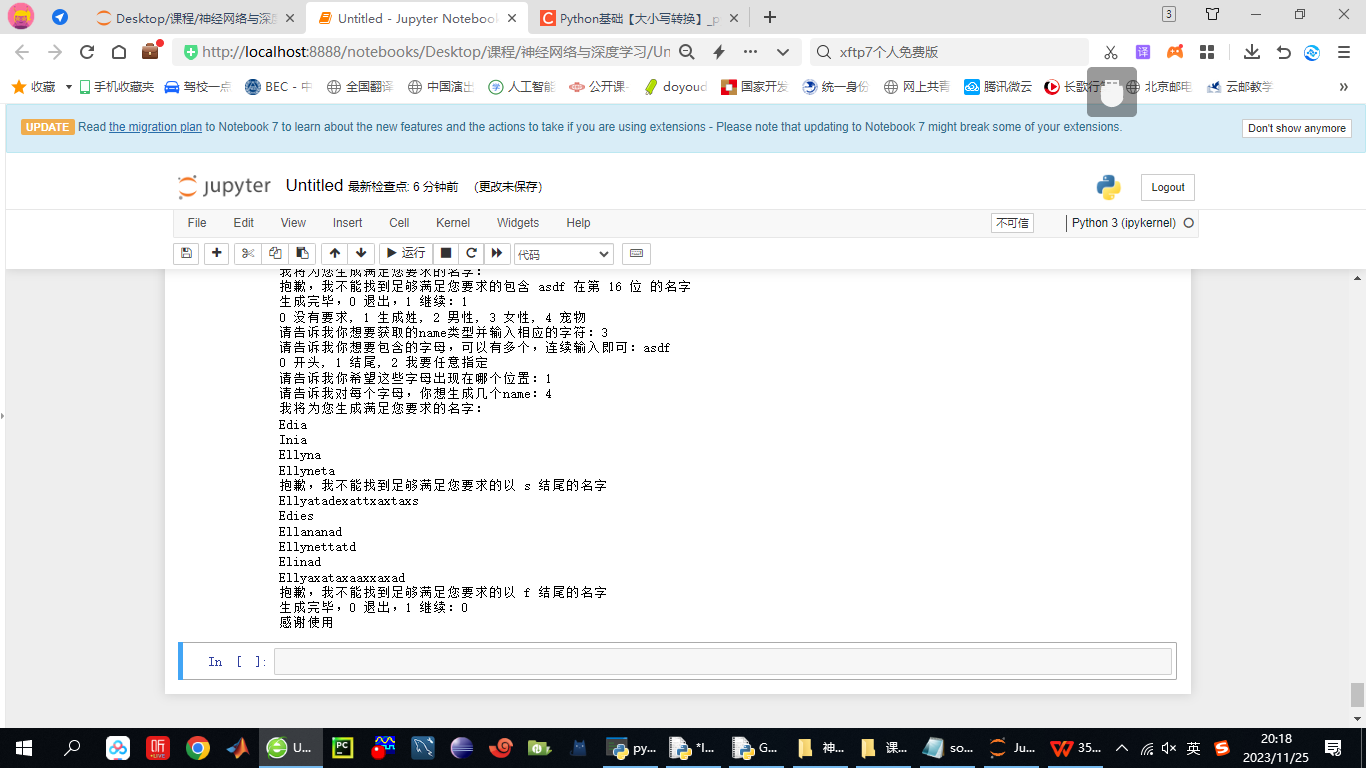
if output\_name[-1] != end\_letter:

output\_name = ''

#print("抱歉，我暂时不能生成以",end\_letter,"结尾的名字")



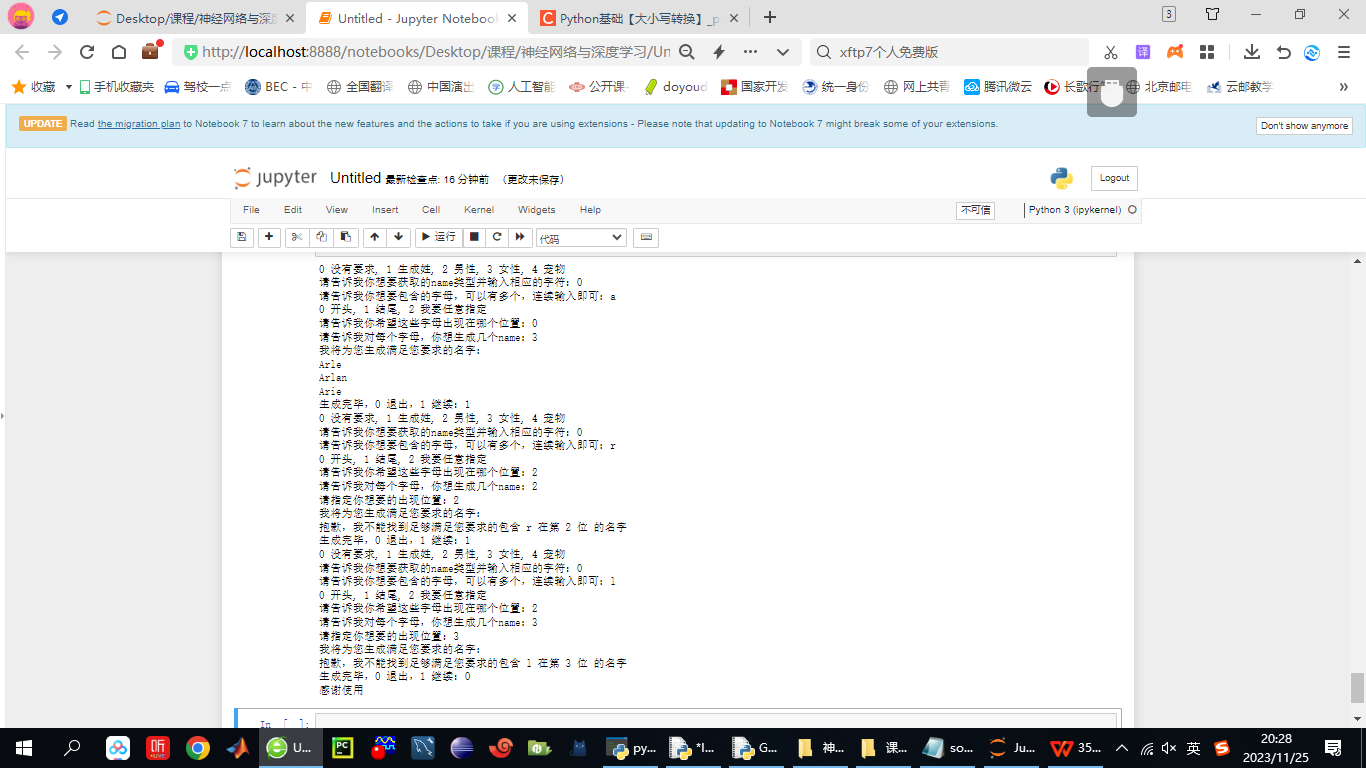
以上为输入一个指定结尾字符的输出。



以上为输入多个指定结尾字符的输出。

从代码可以看到，该思路下，模型多次做重复计算，是一个随机概率事件；从输出来看，该思路生成的名字丰富性不足，且并非每次都能形成生成结果。比如当我想得到4个以s结尾的名字，在第一个测试中我们发现它能够生成Edones，但当我第二次测试，告诉它我想要四个，它却只能生成两个，而且后面的名字也相对来说很不美观，有不太像一个名字的感觉。

基于此思路再对任意位置做生成，这种随机概率的体现就会放大，结果就会更明显：不能成功得到一个满足要求的名字。



因此，我们需要考虑在数据预处理或者模型上下功夫。

5、可视化

p = torch.exp(output)

p = p.numpy().flatten()

#p是概率矩阵，选出p中最大的五个值p[i]及其索引i（i = torch.nonzero(p==p[i]).squeeze()），可绘制all\_letters[j]、p[j]可视化图

我们可以定义获取all\_letters[j]、p[j]的函数如下：

import heapq

def get\_p(output):

p = torch.exp(output)

p = p.numpy().flatten()

p = list(p)

#p是概率矩阵，选出p中最大的五个值p[i]及其索引i（i = torch.nonzero(p==p[i]).squeeze()），可绘制all\_letters[j]、p[j]可视化图

#print(p.size)

#print(p)

p\_max = heapq.nlargest(5,p)#获取前五大的值并排序

index\_max = map(p.index,p\_max)#获取前五大的值下标

p\_max = list(p\_max)

index\_max = list(index\_max)

letter\_max = []

for i in index\_max:

try:

letter\_max.append(all\_letters[i])

pass

except:

#index为结束符的索引

letter\_max.append('end')

print(p\_max)

print(index\_max)#map生成的对象要转化成为list才能输出

print(letter\_max)

return letter\_max, p\_max

为了在生成名字之后让用户有选择性地选择某一名字进行打印，我们可以在每次成功生成后，将过程中获得的letter\_max, p\_max这两个一维列表存进新的列表中，得到一个二维列表，这个二维列表中就含有生成该名字的过程记录。当我们输出多个名字时，对每个加入name列表的名字，将属于它的二维列表记录加入新的空列表中，得到一个三维列表，用户选择输出系统为他生成的第see个名字的生成过程时，我们只需遍历三维列表的[see-1]索引下的数据（即对应名字的二维列表）。

为了画图，我们定义以下两个函数：

#用于对给定名字生成过程中每次推理得到的一维列表画柱状图输出

def plot\_letter(letter\_max, p\_max):

# 准备数据

x\_data = letter\_max

y\_data = p\_max

#设置字体

plt.rcParams["font.sans-serif"] = ["SimHei"]

#字符显示

plt.rcParams["axes.unicode\_minus"] = False

# 画图，plt.bar()可以画柱状图

for i in range(len(x\_data)):

plt.bar(x\_data[i], y\_data[i])

# 设置图片名称

plt.title("当前次预测概率最大的下一个字母及其概率图")

# 设置x轴标签名

plt.xlabel("字母")

# 设置y轴标签名

plt.ylabel("概率")

# 显示

plt.show()

#用于对选定名字的每一步调用plot\_letter函数画图，参数为二维列表

def plots\_letter(letters\_max, ps\_max):

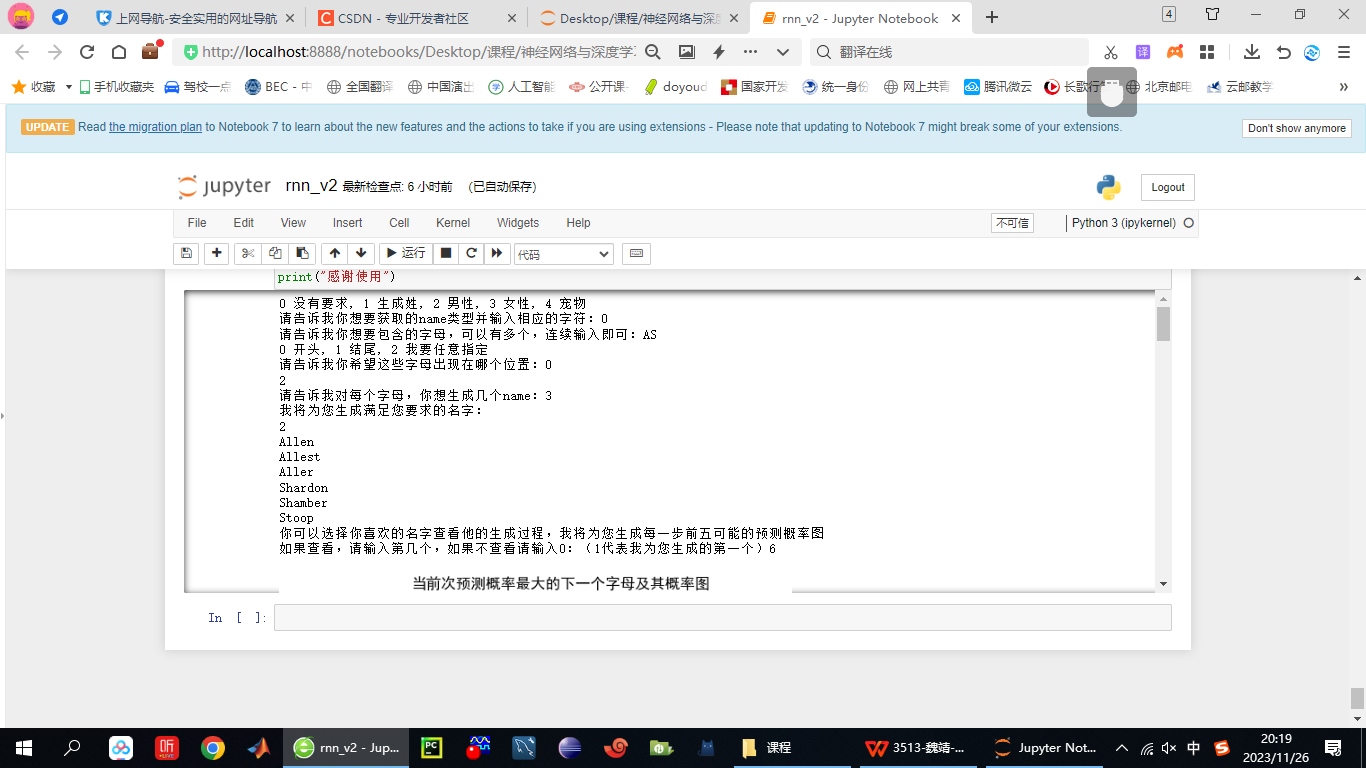
for i in range(len(letters\_max)):

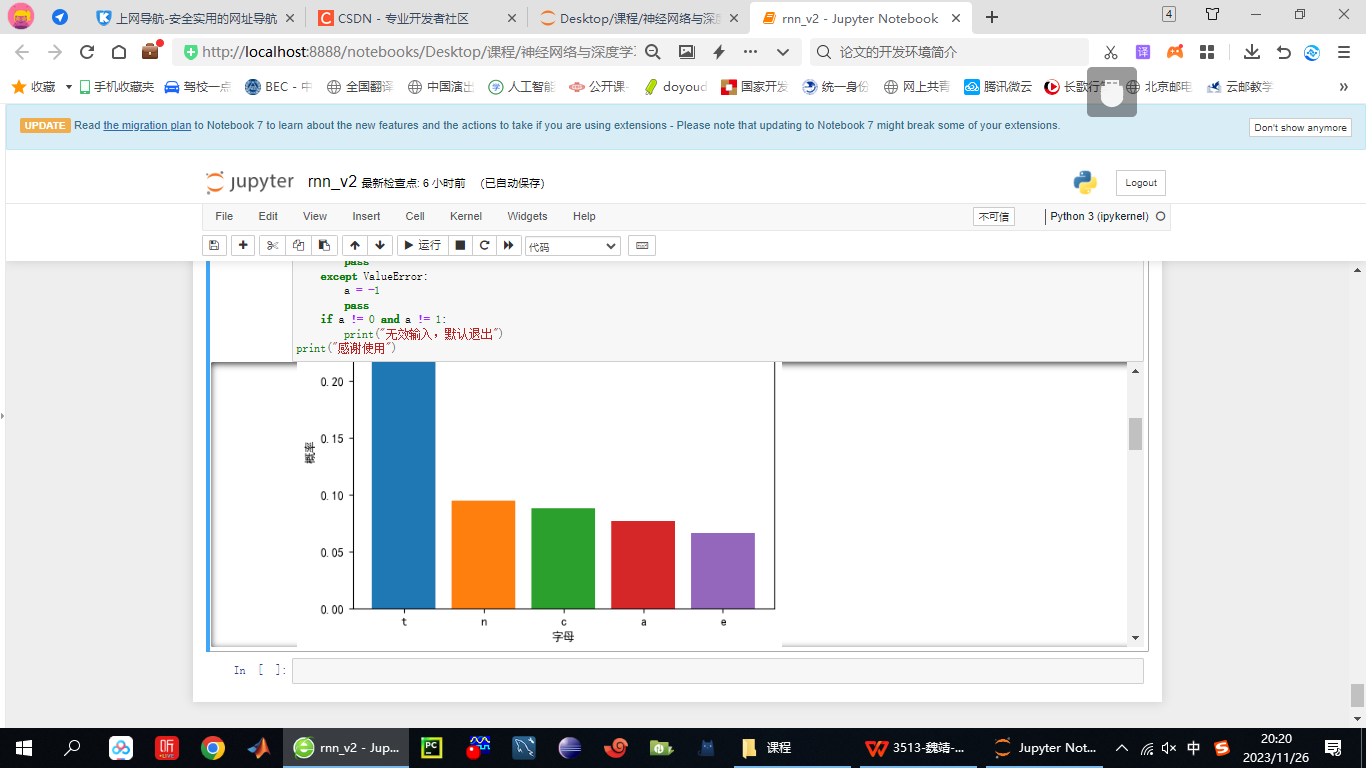
if letters\_max[i] ==[]:

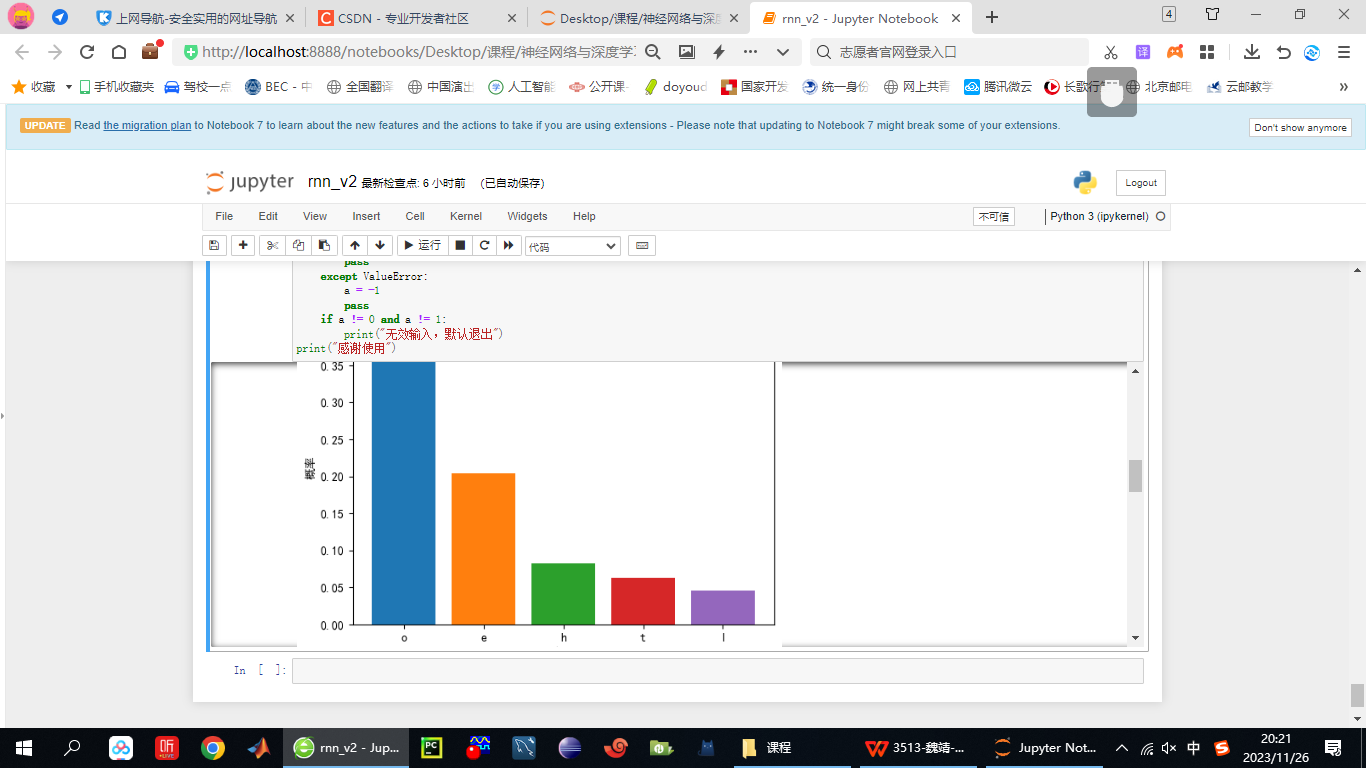
break

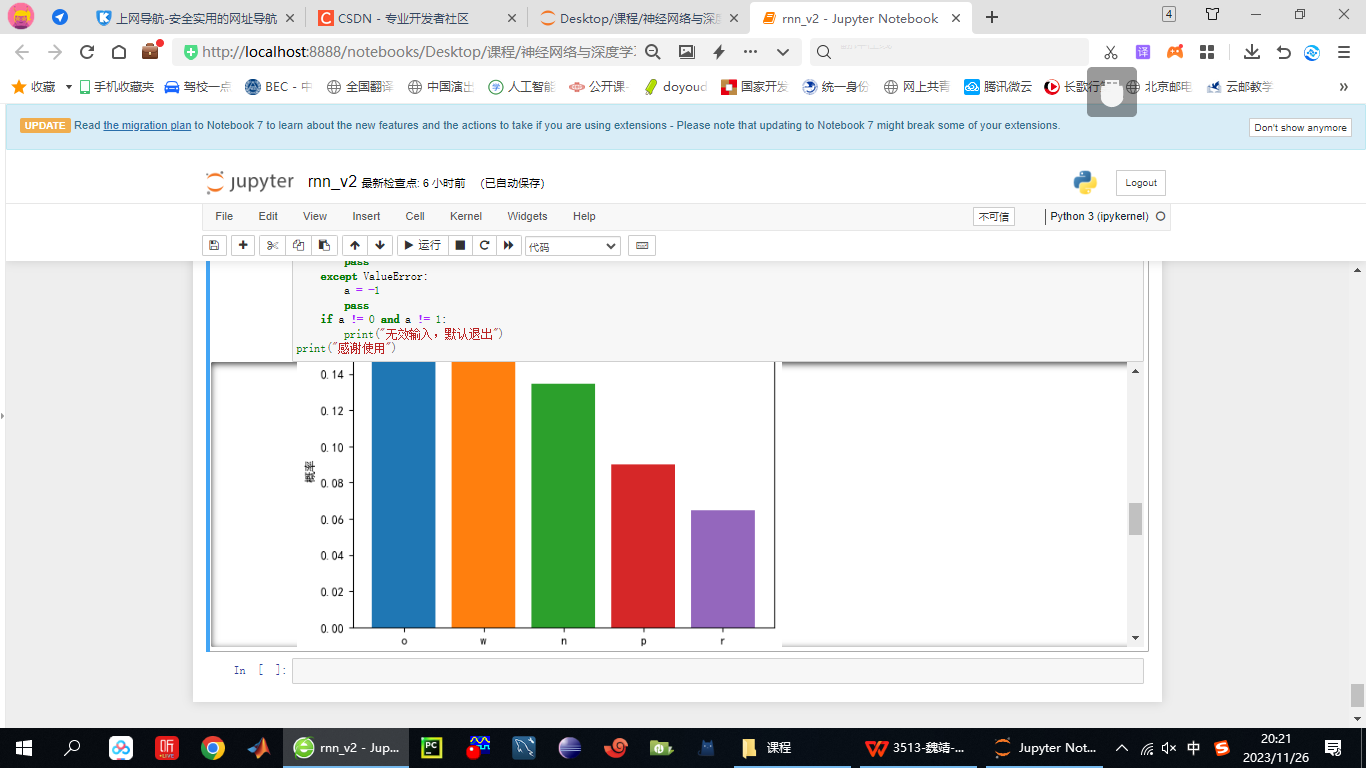
plot\_letter(letters\_max[i], ps\_max[i])

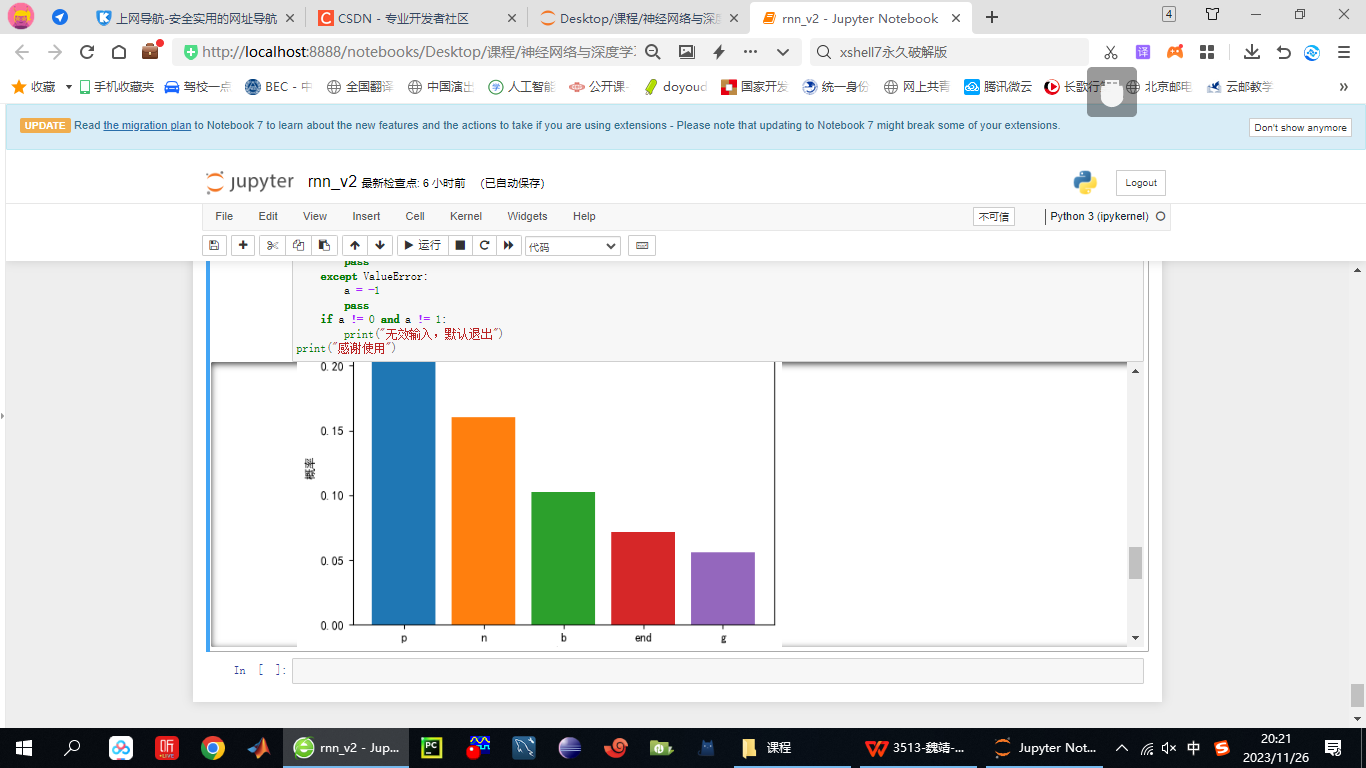
示例：

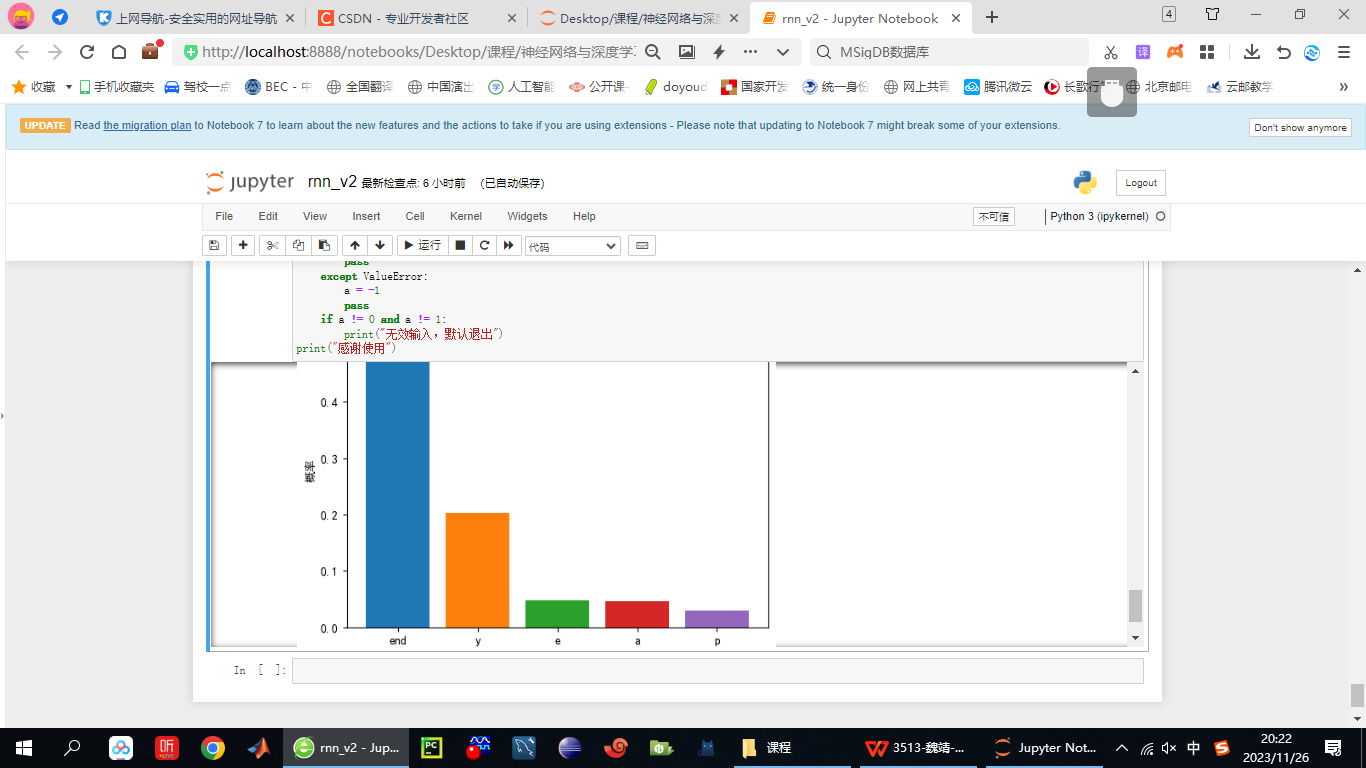












6、总结与反思

通过预处理文本数据、设计和训练rnn，掌握了rnn的底层原理，能够较好地实现用户友好的名字生成系统。