RNN应用-名字分类（按国家）

原创 JIALIN KANG 白鸽读书 2020-06-18

今天具体使用RNN模型做姓氏按国家的分类，学习教程来自PyTorch官网，网址为：
https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

NLP FROM SCRATCH: CLASSIFYING NAMES WITH A CHARACTER-LEVEL RNN
自然语言处理：使用字符级的RNN进行姓氏分类

应用内容：建立和训练一个字符级别的RNN模型。具体是我们先会使用从18种语言中的姓氏来训练RNN模型，然后给一个新的姓氏来预测姓氏属于哪个国家。

一、数据处理

数据来源：https://download.pytorch.org/tutorial/data.zip

以下是具体的十八个国家：

其中中国的部分姓氏如下：

```
Ang
Au-Yong
Bai
Ban
Bao
Bei
Bian
Bui
Cai
Cao
Cen
Chai
Chaim
Chan
Chang
Chao
Che
Chen
Cheng
Cheung
Chew
Chieu
Chin
```

文件格式：
在文件夹data/names下面是18个以[language.txt]命名的文件。每一个文件包含了一系列的姓氏，每一个姓氏占据一行。通过处理将文件用字典的形式进行包含，具体是{language:[names ...]}。下面是在jupyter中实现的具体代码：

1 导入模块

```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
```

```
!dir
```

```
 Volume in drive D is New Volume
 Volume Serial Number is 4AD5-03DA

 Directory of D:\Downloads\wechat

06/17/2020  03:28 PM    <DIR>          .
06/17/2020  03:28 PM    <DIR>          ..
06/17/2020  03:14 PM    <DIR>          .ipynb_checkpoints
06/17/2020  03:04 PM    <DIR>          name_classification
06/17/2020  03:28 PM             2,245 name_classification.ipynb
06/17/2020  12:58 AM            11,543 RNN.ipynb
               2 File(s)         13,788 bytes
               4 Dir(s)  396,096,294,912 bytes free
```

2 定义函数

```
def findFiles(path): return glob.glob(path)
print(findFiles('name_classification/data/names/*.txt'))
```

```
['name_classification/data/names\\Arabic.txt', 'name_classification/data/names\\Chinese.txt', 'name_classification/data/names\\Czech.txt', 'name_classification/data/names\\Dutch.txt', 'name_classification/data/names\\English.txt', 'name_classification/data/names\\French.txt', 'name_classification/data/names\\German.txt', 'name_classification/data/names\\Greek.txt', 'name_classification/data/names\\Irish.txt', 'name_classification/data/names\\Italian.txt', 'name_classification/data/names\\Japanese.txt', 'name_classification/data/names\\Korean.txt', 'name_classification/data/names\\Polish.txt', 'name_classification/data/names\\Portuguese.txt', 'name_classification/data/names\\Russian.txt', 'name_classification/data/names\\Scottish.txt', 'name_classification/data/names\\Spanish.txt', 'name_classification/data/names\\Vietnamese.txt']
```

定义函数介绍：
glob模块：这里使用的glob模块是python自带的一个文件操作相关模块，用它可以查找符合自己目的的文件，类似于Windows下的文件搜索，支持通配符操作（在计算机或软件技术中，通配符可以用于代替单个或多个字符。通常的，星号‘*’匹配0个或以上的字符，问号‘？‘匹配一个字符，[]匹配指定范围内的数字，如[0-9]匹配数字）。

glob方法：glob模块的主要方法就是glob，该方法返回所有匹配的文件路径列表；该方法需要用一个参数用来指定匹配的路径字符串（字符串可以为绝对路径也可以为相对路径），其返回的文件名只包括当前目录里的文件名，不包括子文件夹的文件。

3 将Unicode字符转化成纯ASCII码

```
import unicodedata
import string
```

```
all_letters = string.ascii_letters + " .,;'"
n_letters = len(all_letters)
```

```
# 定义将Unicode字符串转化成纯ASCII码的函数
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )
```

```
print(unicodeToAscii('Ślusàrski'))
```

```
Slusarski
```

【Unicode，ASCII和UTF-8的区别和联系？
这是一个字符编码的问题，总的来说就是在编码各国文字的时候，每个国家的编码方式不一样。最早是由美国制定的一套字符编码，对英语与二进制之间的关系，做了统一的规定。这被称为ASCII码，并且一直沿用至今。但问题是世界上存在着多种编码方式，同一个二进制数字可以被解释成不同的符号。为了给世界上所有的符号都给予一个独一无二的编码，研发出了Unicode。
互联网的普及，强烈要求出现一种统一的编码方式，UTF-8就是在互联网上使用最广的一种Unicode的实现方式。】

```
# 建立category_lines 字典，每一种语言的姓氏列表
category_lines = {}
all_categories = []

# 定义函数：读一个文件，并分成行
def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
    return [unicodeToAscii(line) for line in lines]

for filename in findFiles('name_classification/data/names/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines

n_categories = len(all_categories)

# 输出前五个意大利的姓氏
print(category_lines['Italian'][:5])
```

```
['Abandonato', 'Abatangelo', 'Abatantuono', 'Abate', 'Abategiovanni']
```

4 使用独热编码（one-hot vector）将姓氏转化成张量（Tensors）

我们有了姓氏的字典，我们需要将其转化成为张量Tensors来使用它。
我们使用大小为<1 * n_letters>的独热编码（one-hot vector）来
表示一个简单的字母。一个独热编码向量的只有在当前字母的下标
编码为1，其他的都为0。比如‘b’ = <0 1 0 0 0 . . .>。
为了表示一个单词，将这些向量合并起来形成一个二维的矩阵。
额外的1维是因为PyTorch假设所有内容都是批量的-我们在这里只
使用1的批量大小。

```
import torch

# 找到所有字母的下标，比如‘a’= 0
def letterToIndex(letter):
    return all_letters.find(letter)

# 仅仅作为样例，将字母转化成为一个<1 * n_letters>大小的张量
def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters)
    tensor[0][letterToIndex(letter)] = 1
    return tensor

# 将一行转化成为形状为<line_length * 1 * n_letters>或者是
# 一个独热编码的向量矩阵
def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor

print(letterToTensor('J'))

print(lineToTensor('Jones').size())
```

```
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0.]])
torch.Size([5, 1, 57])
```

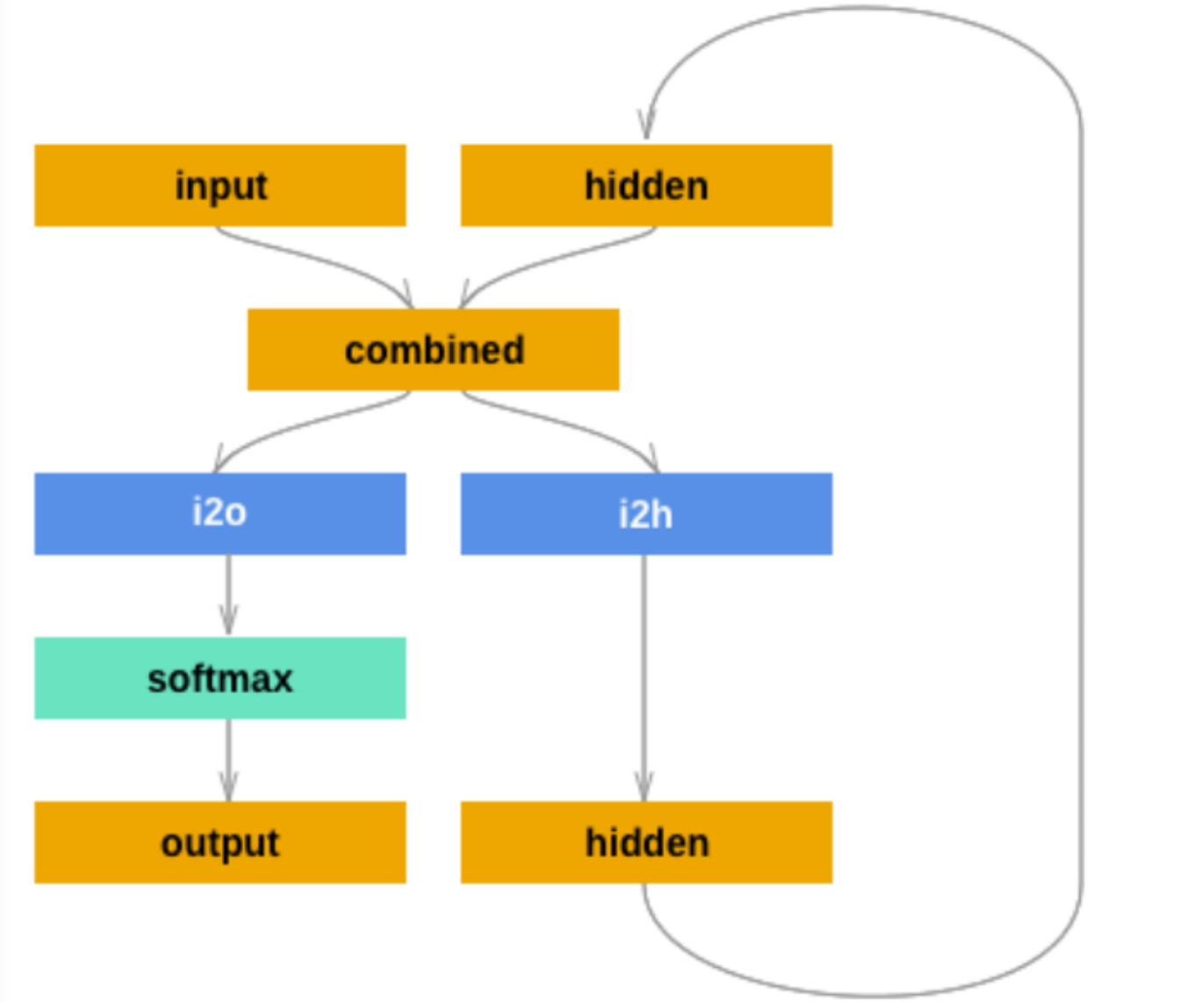
5 创建一个神经网络

pytorch中已经有写好的RNN方法，我们可以以一种很简洁的方法来创建一个RNN神经网络

]:
```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
print(n_letters, n_hidden, n_categories)
```

```
57 128 18
```

下面是具体的RNN神经网络流程图



6 输入测试用例

为了运行RNN网络中的某一步，我们需要输入当前字母的张量和初始的隐藏层（设定为0）.我们得到输出并的到输入下一个隐藏层的张量。

```
input = letterToTensor('A')
hidden = torch.zeros(1, n_hidden)

output, next_hidden = rnn(input, hidden)
print(output, next_hidden)
```

```
tensor([[-2.9956, -2.7920, -2.9005, -2.8569, -3.0309, -2.9180, -2.9116, -2.8723,
         -2.8813, -2.8591, -2.9003, -2.9879, -2.9541, -2.8248, -2.9344, -2.7850,
         -2.8697, -2.7946]], grad_fn=<LogSoftmaxBackward>) tensor([[-3.0044e-02,
1.4153e-02, -4.3966e-02,  2.2961e-02, -2.0384e-02,
          4.9052e-02,  3.1285e-02,  3.6606e-02, -1.6559e-03, -4.4501e-02,
          5.1951e-02,  8.7061e-02,  4.0251e-02, -8.3137e-02,  8.1600e-02,
          1.0364e-01,  3.1485e-02, -3.8218e-02,  3.5633e-02, -9.1784e-02,
         -9.6537e-05, -1.0995e-01,  2.4740e-02,  7.7045e-04, -3.9868e-02,
         -4.7721e-02,  4.3520e-02,  3.0371e-02, -8.7427e-02, -8.3737e-02,
          7.2497e-02, -1.3027e-01, -3.4511e-02, -9.1720e-03,  4.6176e-02,
         -1.6985e-02,  4.1497e-03, -1.1186e-01,  4.4511e-02,  6.1517e-02,
         -6.9737e-02,  7.4860e-02,  4.6895e-02,  4.7629e-02, -5.9640e-03,
         -6.4043e-02,  7.9512e-02,  5.5259e-02, -6.3575e-02,  3.2227e-02,
         -5.7525e-02, -4.1577e-02, -1.4798e-02, -4.5852e-02, -1.4949e-02,
         -4.8460e-02,  6.5745e-02, -5.9358e-02,  6.5038e-02,  4.4087e-02,
          1.0388e-02, -7.8601e-02,  6.7031e-02, -5.3549e-02, -3.6976e-02,
          2.5132e-02,  1.3124e-01,  2.8589e-02,  7.8419e-03, -5.5387e-02,
         -7.4571e-02,  1.7394e-02, -7.9464e-02, -4.9639e-02,  3.9489e-02,
          8.0986e-02, -3.0210e-02, -3.6567e-03,  3.8814e-02, -3.3395e-02,
          1.6117e-02, -3.0676e-02, -4.5131e-03, -1.3578e-02, -4.4870e-02,
         -7.3633e-03,  7.6237e-02,  7.4556e-02,  7.6024e-02, -2.8917e-02,
         -2.7140e-02, -5.4307e-02,  6.8266e-02, -1.9479e-02,  3.8792e-03,
          7.8649e-02,  1.0646e-01, -1.0039e-01, -7.2931e-02,  5.1495e-02,
          2.0397e-02, -1.2536e-01, -7.8824e-02, -2.0505e-02, -1.9297e-02,
         -2.9951e-02, -9.6388e-03,  4.2918e-02,  4.5247e-03, -4.2189e-02,
          1.2239e-01, -2.1621e-02,  1.2604e-01, -2.2288e-02, -1.8384e-02,
         -3.4321e-03,  1.1007e-01,  4.8485e-02, -4.2812e-03, -4.7291e-02,
          6.1552e-02,  1.3576e-01, -3.6203e-02,  3.4711e-03, -1.1385e-02,
         -2.4795e-02, -5.9927e-02,  8.8239e-02]], grad_fn=<AddmmBackward>)
```

7 准备训练RNN网络模型数据

在训练RNN模型之前，我们将使用一些有帮助的函数。
第一个函数就是网络的输出解释，我们将得知每一个类别的概率。
我们可以使用Tensor.topk来获得最大值的下标。

```
def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i

print(categoryFromOutput(output))
```

```
('Scottish', 15)
```

我们需要创建一个快速的方法来创建训练样例（姓氏和其语言）

```
import random

def randomChoice(l):
    return l[random.randint(0, len(l) - 1)]

def randomTrainingExample():
    category = randomChoice(all_categories)
    line = randomChoice(category_lines[category])
    category_tensor = torch.tensor([all_categories.index(category)],
                                   dtype=torch.long)
    line_tensor = lineToTensor(line)
    return category, line, category_tensor, line_tensor

for i in range(10):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    print('category=', category, '/ line =', line)
```

```
category= Portuguese / line = Ferro
category= Scottish / line = Jones
category= Italian / line = Breda
category= Japanese / line = Karamorita
category= French / line = Marchand
category= Irish / line = Fionn
category= English / line = Carrington
category= German / line = Trumbauer
category= Chinese / line = Cai
category= Irish / line = Peatain
```

8 训练RNN网络模型

我们要用很多的训练数据来训练RNN网络模型，并且让网络预测国家，根据预测的是否准确来调整模型参数。采用nn.NLLLoss作为损失函数，原因是RNN网络的最后一层是nn.LogSoftmax.

```
criterion = nn.NLLLoss()
```

每一次训练将会：
- 创建一个输入张量和目标张量
- 创造一个全0的初始隐藏状态
- 读取每个单词并且保持下一个字母的隐藏状态
- 将最后的预测输出和真是目标进行对照
- 反向传播
- 返回输出和损失值

```
learning_rate = 0.005

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    for p in rnn.parameters():
        p.data.add_(p.grad.data, alpha=-learning_rate)

    return output, loss.item()
```

我们现在只需要将数据放入模型中进行训练。因为train函数将会返回输出和损失，我们可以将其预测结果打印出来并且画出损失函数的曲线。我们将只打印损失函数的平均值。

```
import time
import math

n_iters = 100000
print_every = 5000
plot_every = 1000

# 记录loss的值用于画损失曲线
current_loss = 0
all_loss = []

def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

start = time.time()
```

```
for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output, loss = train(category_tensor, line_tensor)
    current_loss += loss

    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == category else '✗ (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100,
                                                timeSince(start),
                                                loss, line,guess, correct))
    if iter % plot_every == 0:
        all_loss.append(current_loss / plot_every)
        current_loss = 0
```

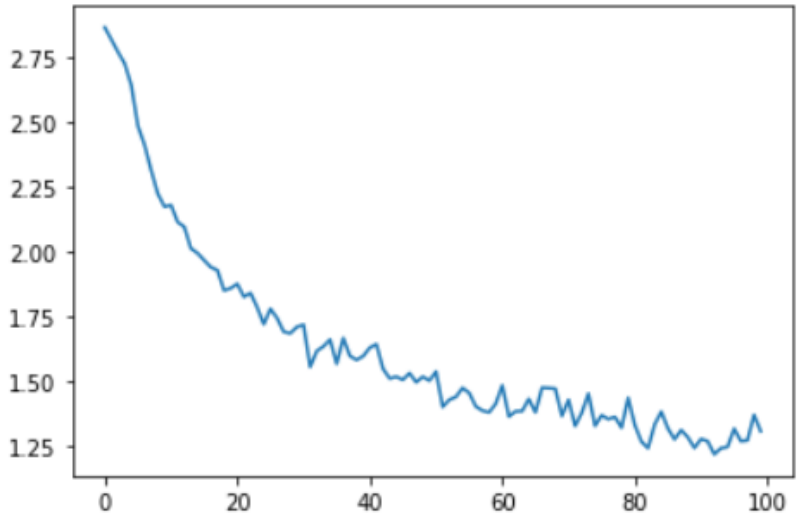
```
5000 5% (0m 9s) 2.4218 Nascimbene / Dutch ✗ (Italian)
10000 10% (0m 20s) 1.1312 Antopolsky / Russian ✓
15000 15% (0m 30s) 4.0684 Close / Scottish ✗ (Greek)
20000 20% (0m 40s) 0.5049 You / Korean ✓
25000 25% (0m 51s) 1.7926 Faure / Arabic ✗ (French)
30000 30% (1m 1s) 1.9056 Narita / Portuguese ✗ (Japanese)
35000 35% (1m 12s) 0.6025 Poniros / Greek ✓
40000 40% (1m 23s) 1.9508 Tsai  / Arabic ✗ (Korean)
45000 45% (1m 34s) 0.6777 Ribeiro / Portuguese ✓
50000 50% (1m 45s) 1.2969 Seelen / Dutch ✓
55000 55% (1m 57s) 2.0344 Russel / German ✗ (English)
60000 60% (2m 8s) 0.9086 Lambert / French ✓
65000 65% (2m 19s) 0.4696 Ta / Vietnamese ✓
70000 70% (2m 30s) 0.7405 an / Vietnamese ✓
75000 75% (2m 42s) 1.4717 Peeters / Portuguese ✗ (Dutch)
80000 80% (2m 55s) 0.0589 Janowski / Polish ✓
85000 85% (3m 6s) 2.9837 Blanc / Scottish ✗ (French)
90000 90% (3m 19s) 0.1685 Kruger / German ✓
95000 95% (3m 31s) 0.8876 San / Chinese ✗ (Korean)
100000 100% (3m 44s) 0.0061 Bukowski / Polish ✓
```

总共迭代了100000次，每5000打印一次当前RNN模型预测的结果与真实结果的情况。
每1000次收集一次损失函数的值。

9 结果可视化

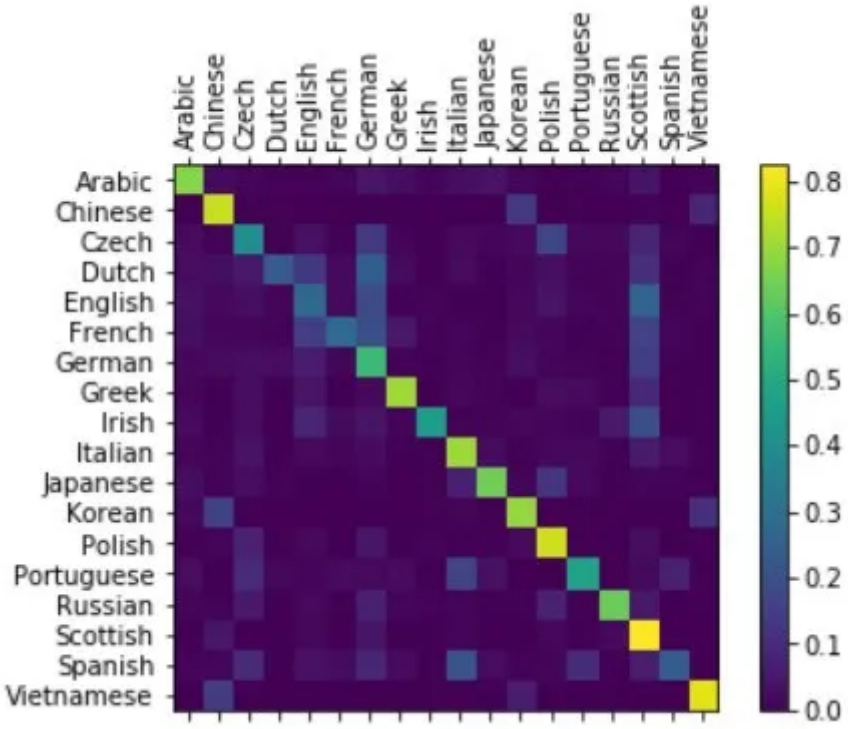
```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.figure()
plt.plot(all_loss)
```

52]: [<matplotlib.lines.Line2D at 0x21f1690c508>]



上面的曲线纵轴为损失值，横轴为迭代的次数（每1000次），可以看出，随着迭代次数的增加，损失值减少，也就是在训练模型中预测的准确度越高。

```
confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000

def evaluate(line_tensor):
    hidden = rnn.initHidden()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)
    return output

for i in range(n_confusion):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output = evaluate(line_tensor)
    guess, guess_i = categoryFromOutput(output)
    category_i = all_categories.index(category)
    confusion[category_i][guess_i] += 1

for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum()

fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(confusion.numpy())
fig.colorbar(cax)

ax.set_xticklabels(['']  + all_categories, rotation=90)
ax.set_yticklabels(['']  + all_categories)

ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

plt.show()
```



上面是一个热度图（heat-map）横轴代表RNN网络预测的结果，纵轴代表姓氏的真实国家。颜色越深代表概率越低，颜色越亮代表概率越大。可以看出亮色的部分集中在对角线上，也就是说RNN网络达到了很好的预测效果。但是除了对角线之外的其他地方也有一些比较明显的亮色。比如说英语（English）和苏格兰语（Scottish）就很容易判断错误。同样的情况出现在汉语（Chinese）和韩语（korean）也很容易判断错误。但是这样符合我们的认知。

10 自己输入姓氏进行判读

```
def predict(input_line, n_predictions = 3):
    print('\n> %s' % input_line)
    with torch.no_grad():
        output = evaluate(lineToTensor(input_line))

        topv, topi = output.topk(n_predictions, 1, True)
        predictions = []

        for i in range(n_predictions):
            value = topv[0][i].item()
            category_index = topi[0][i].item()
            print('(%.2f) %s' % (value, all_categories[category_index]))
            predictions.append([value, all_categories[category_index]])
```

```
predict('kang')
predict('zhao')
predict('liang')
```

```
> kang
(-1.18) German
(-1.52) Chinese
(-1.93) Dutch

> zhao
(-1.75) Chinese
(-1.98) Japanese
(-2.18) Russian

> liang
(-1.40) German
(-1.98) English
(-2.32) Chinese
```

我在这里输入了三个中国姓氏的拼音kang（康），zhao（赵），liang（梁），RNN网络预测kang的结果是：最高概率是German（德文）第二是Chinese（中文）第三是dutch（荷兰文）

以上就是关于姓氏国别预测RNN模型学习的全部内容
全程代码在jupyter中完成，进入阅读原文获得我上传在github上的源代码和训练数据