计算机科学与技术学院自然语言处理课程实验报告

实验题目：RNN 文本生成		学号：201600122057
日期：2019.4.20	班级： 智能 16	姓名： 贝仕成
Email：337263318@qq.com		

实验目的：基于 numpy 手动实现循环神经网络 RNN 并完成前向计算以及反向传播过程，并调整相应的参数，生成不同的文本。

实验软件和硬件环境：

Python 3.5.6

Jupyter notebook 5.0.0

神舟战神 Z7M-KP7S1 Windows10 16GRAM

NVIDIA GTX1050Ti

实验原理和方法：

根据 RNN 的流程，推演它的前向计算和反向传播，对 softmax 函数加入参数 τ 进行调整，可以观察其对结果的影响。

实验步骤：（不要求罗列完整源代码）

1. 补完 min-char-rnn.py，并按照实验要求的 part2 对 softmax 函数进行调整，加入了参数 τ （temp）

预处理部分：

```
# data I/O
data = open('shakespeare_train.txt', 'r').read() # should be simple plain text file
chars = list(set(data)) # 得到输入文件中所有字符种类
data_size, vocab_size = len(list(data)),len(chars) #your code##统计文件字符数和字符种类数
print ('data has %d characters, %d unique.' % (data_size, vocab_size))
char_to_ix = {ch:x for x,ch in enumerate(chars)}#your code# #构成从字母到数字的映射
ix_to_char = {x:ch for x,ch in enumerate(chars)}#your code# #构成数字到字母的映射
```

前向传播部分：

```
def lossFun(inputs, targets, hprev,temp=1):
    """
# forward pass
for t in range(len(inputs)):
   #encode inputs to 1-hot embedding,size(xs)=(len(input),vocab_size)
   xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation 1-hot-encoding
   xs[t][inputs[t]] = 1 # encode in 1-of-k representation 1-hot-encoding
   #forward

   #hs[t] 是t时刻的hidden state， active function = np.tanh(z), z = Wx*x_t+Wh*hs_(t-1) + bh,即本时刻输入层+一时刻个隐含层作为Z
    hs[t] = np.tanh(np.dot(Wxh,xs[t])+np.dot(Whh,hs[t-1])+bh)# hidden state
   #ys[t] = w*hs[t]+by
    ys[t] = np.dot(Why,hs[t])+by# unnormalized log probabilities for next chars
   #softmax(ys)
    ps[t] = np.exp(ys[t]/temp)/np.sum(np.exp(ys[t]/temp)) # probabilities for next chars
   #计算loss = cross_entropy ()
    loss += -np.log(ps[t][targets[t],0])# softmax (cross-entropy loss)
```

反向传播部分：

```
# backward pass: compute gradients going backwards
#初始化梯度
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(range(len(inputs))):
  #dy是softmax层求导，cross_entropy softmax 求导 aj-yi,yi为one-hot标签,aj为softm
      dy = np.copy(ps[t])
      dy[targets[t]] -=1#your code# # backprop into y.
  #反向传播，求Why与by的导数
      dy=dy/temp
      dWhy += np.dot(dy, hs[t].T)
      dby += dy
      #循环中每一步对应dby都是dy*1，再循环中不断累加。
  #反向传播到hidden state请参考https://blog.csdn.net/wjc1182511338/article/deta
      dh = np.dot(Why.T, dy)+dhnext # backprop into h
      dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
      dbh += dhraw
      dWxh += np.dot(dhraw,xs[t].T)
      dWhh += np.dot(dhraw,hs[t-1].T)
      dhnext =np.dot(Whh.T,dhraw)
```

尝试运行结果：

```
----
 yak tietburisd aH,
Th ymi tf beres tiarrnoy rathetiu
Tunoudee non

Ssey olle Lh itlaus iiyoraThenhe hetiiwl bye arls k
SQawsrs  hor, thu Fraheg woatood Clar'
A,
gree
Thast.
Thee

S: anird ?ir ilrysy
----
iter 1100, loss: 84.697856



----
  arsith ot thle.

HENIUCETER:
ASgert youre.

SuCOLTIUS:
Lfelanes I ceod; thy whed kpycill cethif I of, leititerls:
Heshit at fos. the to lestio ceptir and feaks un se seresen: on this tilasom dicher a
----
iter 13000, loss: 55.874368
```

```
----
 s and ofutiried.

MARP:
Wrvele.

CMIRIUS:
Cin giiths ne coner,
A nownef esuicienthi hes Botlonst hele me neratio:
Fre; he doow, y Mar fr a, aks ir thow, that Caglond! I reew and ther Hathy's ougy.

MA
----
iter 33300, loss: 52.940391
```

尝试修改 temp（τ）来查看实验结果：

τ=1

```
----
 ore-CLOUCEES:
Therer muat nent the grevee,
To fonsd bon'd sof dime ro shour touns, enas hapraf hit and Arul mor,
Ttanet ard.
By dush My lobl aaZt ound Init, hoth bersred I duxh to tracs fowh I aR:
An,
----
iter 0, loss: 103.146355


----
 is
sprese oplit sead mo.
Whopuls notlu bue sar jomot of thers.

CORIOL
USIINIUS:
We watl ath fepretore, fighn
Thit Tore piat hor, shour.

Therins garse
same lre tizee he bo nit hean:
Ant af mo y'd abl
----
iter 5000, loss: 55.044631
```

```
----
  ans bloerde souren,
Hood,
RBer ther heat 'dt,
Thit hat the thar'sd And kut hat then sole on bootA thethel?

ALCORINCAMNKIG:
We he se heald gher;
Mitincae thagts his ow asn burssde
Bold Gy go arer sea
----
iter 10000, loss: 53.714144


----
 rd ins, cond my foll out sancance fonpece shill my reveich and; have the,
Bot,
Will loour bodevoudd hithrer famy, of that thy to ale,
Wher hey fobl eors dings my
Where his oin have arck me this nive a
----
iter 30000, loss: 50.531827


----
 d brsorked ontwers lape
your.
Or,-

ARIALANDE:
Codntore nce mest dride I ackme try weakl ust.

Oxas, hame knamaboll; bies?
To in unell the bare rill yir emcadef; bloushire youlyold sold, fongernt whir
----
iter 50000, loss: 48.852493
```

τ =2

```
----
 re maRmpusy.
Of'ee:
Thlt;,
Ftus exitn;
Togtotl, Aifow': woo;n?
ccewhy, bu bill-ion min. mysh.k-entoneyghiblverie bryturs?
Voripc.

FItUTG:
Land morm; ahturnIs-?,
B'l'
MraTe
Amm'yex,
LfrWeried'obry'Hr
----
iter 0, loss: 103.128019

----
 is.

FRoviman and
Is the Bubthet Stir.

Ther:
end
Talls thenod ''t Camand he hape peat's weit.

SCAMENIUS:
Anent Riis iow, guse camexerane
Hime wiwhrenith in: youn, andme bod enim whim rore star raunt
----
iter 5000, loss: 54.012478


----
  the terser goDy'ch inod your hath orton?

LORVY
A:
inde dees theem
Got uhars 'fore tho of, haar
Whet Mhee menins, in to gorstity gat bled was tol:
The prince glur, ind jigses micill got towd ward I,
----
iter 10000, loss: 52.551759
```

```
----
 ve espriet kith versI brit and us, all to lows Paee the ingRoves,

Thirstule in bo,
And, hear and are dor ham he put my hayeg.

SICBSBRUTUTUM:
Bnatsingt you sreat foray
Bun, I ly not reary!

ARcou bo
----
iter 30000, loss: 50.161930



----
 der,
Than As bith thends,
Poct homeme one a benonther the fosensre
Be to fimertiem heres whes shate my, werviole briend?

AUCINOU:
Gary morse; you'dhis shobly bum daste! fotet I pifode at in that, ai
----
iter 50000, loss: 48.477584
```

2. 自己编写 RNN 来生成文本如下（载入了 char-rnn-snapshot.npz 中的参数）：

```python
def lossFun_d(inputs, targets, hprev,Wxh,Whh,Why,bh,by,temp=1):
    xs, hs, ys, ps = {}, {}, {}, {}
    hs[-1] = np.copy(hprev)
    loss = 0
  # forward pass
    for t in range(len(inputs)):
        xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation 1-hot-encoding
        xs[t][inputs[t]] = 1 # encode in 1-of-k representation 1-hot-encoding
    #forward
    #hs[t] 是t时刻的hidden state， active function = np.tanh(z), z = Wx*x_t+Wh*hs_(t-1) + bh,即本时刻输入层+一时刻个隐含层作为Z
        hs[t] = np.tanh(np.dot(Wxh,xs[t])+np.dot(Whh,hs[t-1])+bh)# hidden state
    #ys[t] = w*hs[t]+by
        ys[t] = np.dot(Why,hs[t])+by# unnormalized log probabilities for next chars
    #softmax(ys)
        ps[t] = np.exp(ys[t]/temp)/np.sum(np.exp(ys[t]/temp)) # probabilities for next chars
    #计算loss = cross_entropy ()
        loss += -np.log(ps[t][targets[t],0])# softmax (cross-entropy loss)
    print(hs[t])
    dWxh, dWhh, dWhy,dbh, dby = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why),np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])
    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -=1#your code# # backprop into y.
        dy/=temp
    #反向传播，求Why与by的导数
        dWhy += np.dot(dy, hs[t].T)
        dby += dy  #循环中每一步对应dby都是dy*1，再循环中不断累加。
    #反向传播到hidden state请参考https://blog.csdn.net/wjc1182511338/article/details/79191099完成，其中dh处反向传播的梯度外需加上dhnext
        dh = np.dot(Why.T, dy)+dhnext # backprop into h
        dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
        dbh += dhraw
        dWxh += np.dot(dhraw,xs[t].T)
        dWhh += np.dot(dhraw,hs[t-1].T)
        dhnext =np.dot(Whh.T,dhraw)
    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
        np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

```
def sample_modify(h, seed_ix, n, Wxh, Whh, Why, bh, by, temp=1):
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in range(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y/temp) / np.sum(np.exp(y/temp))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes
```

```
def RNN_generative(start,hidden_size,tao,n,learning_rate):
    a = np.load(open("char-rnn-snapshot.npz",'rb'))
    Wxh,Whh,Why,bh,by,mWxh, mWhh, mWhy = a["Wxh"],a["Whh"],a["Why"],a["bh"],a["by"],a["mWxh"], a["mWhh"], a["mWhy"]
    mbh, mby = a["mbh"], a["mby"]
    chars, data_size, vocab_size= a["chars"].tolist(), a["data_size"].tolist(), a["vocab_size"].tolist()
    char_to_ix, ix_to_char = a["char_to_ix"].tolist(), a["ix_to_char"].tolist()
    smooth_loss = -np.log(1.0/vocab_size)*(len(start)-1) # loss at iteration 0
    hprev = np.zeros((hidden_size,1))
    if len(start)==1:
        inputs=[char_to_ix[ch] for ch in start[0]]
    else:
        inputs=[char_to_ix[ch] for ch in start[0:-1]]


    '''
    for i in range(m):
        loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun_d(inputs, targets,hprev,Wxh,Whh,Why,bh,by)
        smooth_loss=smooth_loss*0.999+loss*0.001
        for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                      [dWxh, dWhh, dWhy, dbh, dby],
                                      [mWxh, mWhh, mWhy, mbh, mby]):
            mem += dparam * dparam
            param += -learning_rate * dparam / np.sqrt(mem + 1e-8)
     '''
    sample_ix=sample_modify(hprev,inputs[-1],n,Wxh,Whh,Why,bh,by)
    text=''.join(ix_to_char[ix] for ix in sample_ix)
    print ('----\n %s \n----' % (text, ))
    print(np.max(Wxh[:,inputs[0]]))
```

（这么编写是可以在既有参数的基础上继续更新参数）

直接生成给定 string 的后 400 个字符（只使用既有参数，且 τ 设为 1）

```
start='Please do not be late'
```

```
RNN_generative(start,250,1,1,400,0.01)
```

生成：

```
----
 orow in dees

MENININIZABETHBABUTAR:
Sent ail'' when befortep 'Tis!

SICINIUS:
Your dook: in ruwn brice on then, that wounf in that I hence forftle:
Why dear
We of a
DAR:

VIUTEIUS:
In dir his not death tell the thou be words, I the seear my they for fan gersedacherit suse Citrerts hat,
But, and hat same gothor!

CORIOLANUS:
O, mang!

VOLUMNIA:
I tamp theity,
And where than lord and bufs.

VOLUMN
----
```

3.对冒号后经常出现换行和空格的解释：

将“:”传入 model（载入的是 npz 文件中的参数），转换为其对应的 one-hot 向量 x，得到的 Wxh 中的最大响应值为：**Wxh[100][9]=4.829189868371359**。由于 Whh 和 h 的值不够大，比不过 Wxh • x 的值，所以忽略它们的影响。新计算出的 h，经发现其第一百维的数值最大，**h[100]=0.9998877562219607**，而 h 中元素的值的范围为[-1,1]，所以可以认为 h[100]被激活了。接着 Why 和 h 相乘，观察 Why 中的第 100 列，发现 **Why[0][100]**和 **Why[2][100]**的值最大，分别为 **2.67271236** 和 **2.26381243**，那么最后所得的 y 中 **y[0]**和 **y[2]**最大。显然经过 softmax 后再 sample，y[0]和 y[2]所对应的字符更容易被选出来。而这两种字符正是“\n”(换行符）以及“ ”(空格)，因此，“:”后面更容易出现这两种字符。

4.其他有趣现象：模型会周期性地在单词之间生成空格，即“ ”

Wxh • x 和 Whh • h 共同影响下一个 h，由于计算入的 h 中第 **73** 维通常是负数，并且 Whh 中 **Whh[73][73]**是第 73 列中最大的，所以这两者相乘有让 h[73]减少的影响，而 Wxh • x 也会影响 h 的计算，它对要计算的 h[73]有令其增大的影响。如果这个增大的影响远没有减小的影响大，那么 h[73]便会减小。
当 h[73]减小到一定值（比如**-0.9999466**），而 **Why[2][73]=-4.08772919** 是 **Why[:,73]**最大的值，那么这两者相乘将会得到一个很大的正值，它会使 y[2]的值变得非常大，也就是说模型更可能 sample 出一个空格，这就是模型会时不时生成空格的原因。

空格的生成不单单取决于空格前面的某个字符，而是前面一系列若干字符影响的叠加。它们对 sample 出空格的概率的影响会随着模型运行而逐渐积累（体现在 h 中），一旦和空格对应的这个值——也就是 y[2]变得足够大，空格就会生成出来。

结论分析与体会：

这是一次很有意思的实验，让我通过实际操作更进一步地了解了简单的 RNN 网络。就是由于是循环网络，反向求导过程花了点时间，不过理解后就觉得并不那么难了。实验中有许多有趣的现象都可以通过各个 W 矩阵以及 h 之间的相互作用来解释，我觉得以后可以用 RNN 在生成模型上面开展进一步工作。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1－3 道问答题：

1. 直接读取 char-rnn-snapshot.npz 会报错，该如何解决？

```
a = np.load(open("char-rnn-snapshot.npz",'rb'))
```

在后面加一个‘rb’便不会报错。