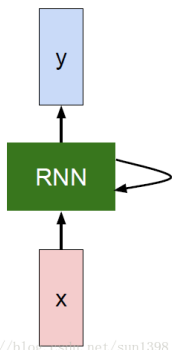
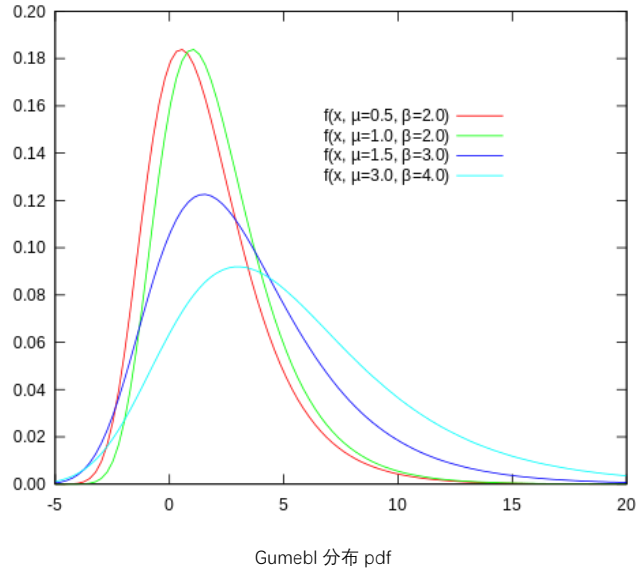


计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：风格迁移		学号：201600301148
日期：4.25	班级：人工智能	姓名：周阳
Email: 862077860@qq.com		
<p>实验目的：</p> <ol style="list-style-type: none">1. 实现 RNN 前项计算2. 实现 RNN 反向传播 (BPTT)3. 实现 gumble softmax + temprature4. 参数解释		
<p>实验软件和硬件环境：</p> <p>CPU: 英特尔至强 E5 GPU: NVIDIA GeForce 1060 6G 内存: 16G Pycharm Python 3.6</p>		
<p>实验原理和方法：</p> <p>1, RNN 前向计算原理</p> <p>We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:</p> <div><div>$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$<p>new state some function with parameters W old state input vector at some time step</p></div><div><p>https://blog.csdn.net/sun1398</p></div></div> <p>定义输入是一个 one-hot 向量代表一个词的表示 $\mathbf{x} \in \mathbb{R}^{\text{vocabsize} \times 1}$</p> $\mathbf{z} = \mathbf{W}_{xh} * \mathbf{x}_t + \mathbf{W}_{hh} * \mathbf{h}_{t-1} + \mathbf{b}_h \quad (1)$ $\mathbf{h}_t = \tanh(\mathbf{z}) \quad (2)$ $\mathbf{y}_t = \mathbf{W}_{hy} * \mathbf{h}_t + \mathbf{b}_y \quad (3)$ $p_t = \frac{e^{\mathbf{y}_t - \max(\mathbf{y}_t)}}{\sum_i^{\text{inputsize}} e^{\mathbf{y}_t^i - \max(\mathbf{y}_t)}} \quad (4)$ <p>上面就是整个前项计算的过程，其中最终使用的 softmax 进行输出 target。</p>		

$$noise \sim gumbel(\mu, \beta) \quad (5)$$



$$y_t^{new} = noise + y_t \quad (6)$$

如果使用 gumbelsoftmax 和 temperature 则使用上述公式 (5) 和 (6) 代替原公式 (3) 即可。

Gumbelsoftmax 对 gumbel 分布进行采样

Loss 使用交叉熵 loss, 定义如下:

$$Loss = y_t^{real} * \log(P_t) \quad (7)$$

2, RNN 反向传播原理

RNN 的反向传播要考虑到上一次的隐状态输出 h_{t-1} .

重要公式推导如下:

$$\frac{dLoss}{dy_t^i} = \begin{cases} p_t^i, & \text{if } i \neq target \\ p_t^i - 1, & \text{if } i = target \end{cases} \quad (8)$$

$$\frac{dy_t}{dW_{hy}} = h_t \quad (9)$$

$$\frac{dy_t}{db_y} = 1 \quad (10)$$

$$\frac{dLoss}{dh_t} = \frac{dLoss}{dy_t} \frac{dy_t}{dh_t} + \frac{dh_{t+1}}{dh_t} = W_{hy}^T * dy + dh_{t+1} \quad (11)$$

实验步骤：（不要求罗列完整源代码）

1, 前项传播与反向计算

```
#encode inputs to 1-hot embedding,size(xs)=(len(input),vocab_size)
xs[t] = np.zeros((vocab_size,1))#your code# # encode in 1-of-k representation 1-hot-encoding
xs[t][inputs[t]] = 1 #your code# # encode in 1-of-k representation 1-hot-encoding
#forward

#hs[t] 是 t 时刻的 hidden state, active function = np.tanh(z),  $z = Wx \cdot x_t + Wh \cdot hs_{(t-1)} + bh$ ,即本时刻输入层+一时刻个隐含层作为 Z
z = Wxh.dot(xs[t]) + Whh.dot(hs[t-1]) + bh
hs[t] = np.tanh(z) #your code# # hidden state
#ys[t] = w*hs[t]+by
ys[t] = Why.dot(hs[t]) + by #your code# # unnormalized log probabilities for next chars

## gumbel
noise = np.random.gumbel()
ys[t] = (noise + ys[t])/tempreature

#softmax(ys)
ps[t] = np.exp(ys[t] - np.max(ys[t]))/np.sum(np.exp(ys[t] - np.max(ys[t]))) #your code# #
probabilities for next chars
#计算 loss = cross_entropy ()
loss += - np.log(ps[t][targets[t]]) #your code# # softmax (cross-entropy loss)

#dy 是 softmax 层求导, cross_entropy softmax 求导  $a_j - y_i, y_i$  为 one-hot 标签, $a_j$  为 softmax 之后第 j 个神经元输出, 详情请见
https://blog.csdn.net/u014313009/article/details/51045303
dy = ps[t]#your code#
dy[targets[t]] -= 1 #your code# # backprop into y.
#反向传播, 求 Why 与 by 的导数
dWhy += dy.dot(hs[t].T) #your code#
dby += dy #your code#
# 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 = Why.T.dot(dy) + dhnext #your code# # backprop into h

dhraw = dh * (1 - hs[t]**2) #your code# # backprop through tanh nonlinearity
dbh += dhraw #your code#
dWxh += dhraw .dot(xs[t].T)#your code#
dWhh += dhraw .dot(hs[t-1].T) #your code#
dhnext = Whh.dot(dhraw) #your code#
```

2, 生成

```

def sample_softmax(ys):
    #softmax(ys)
    pt = np.exp(ys - np.max(ys))/np.sum(np.exp(ys - np.max(ys))) #your code# #
probabilities for next chars
    ix = np.random.choice(range(vocab_size), p= pt.ravel())
    return ix_to_char[ix]

def sample_gumble_softmax(ys,tempreature):
    noise = np.random.gumbel()
    ys = (noise + ys)/tempreature
    pt = np.exp(ys - np.max(ys))/np.sum(np.exp(ys - np.max(ys))) #your code# #
probabilities for next chars
    ix = np.random.choice(range(vocab_size), p= pt.ravel())
    return ix_to_char[ix]

def sample_single_word(last_word = "a",last_state = np.zeros((hidden_size,1))):
    xs = np.zeros((vocab_size,1))
    xs[char_to_ix[last_word]] = 1

    #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
    z = Wxh.dot(xs) + Whh.dot(last_state) + bh
    hs = np.tanh(z)
    ys = Why.dot(hs) + by
    return ys,hs

```

3,续写生成结果讨论

使用随机种子,使结果可复现

Softmax 结果:

```

-----softmax sample-----
Press'
Stribtoch
And the and mell ott,
Tim faloyseed bead
We, your sunsger noo'
Whes have the the not

```

Gumbel tempreature = 1

```

-----gumbel sample temp:1-----
ace
Whisious, we insule wod-'niethices too't, but oork'd we dear having did love mas
e home;
In scoma

```

Gumbel tempreature = 2

```
-----gumbel sample temp:2-----
Greiv! lo torem. elizef: fuxaysoban. Mands:
revpate gach't an itie
Weste
Telf noplay
Oies; Wear.?
Yo
```

Gumbel tempreature = 5

```
-----gumbel sample temp:5-----
fJw.patggsbuzadn-I?waZuZhVnasnamx-Rnmep;, Hukeqams
Ofw,stnvukn;h::-OLliHdiicts 'im.qw!
hakJ,
Cglo:v
```

Gumbel tempreature = 10

```
-----gumbel sample temp:10-----
hl:?dr fEe :scejcvvrwrk'pGSCatFa,tKyvm
wCsLloCtjqw:Bqove&C??HSVOC:,! L
jsoJSvpuysuvvaRR UQ;ySb uaf-r
```

讨论：虽然说，gumbel softmax 可以模拟随机采样，tempreature 参数决定了，这种采样分布的平滑性，tempreature 越小最终越离散，可以看出结果越好。

4，生成使用的参数讨论

输入一个“冒号”所使用的计算参数有：

首先冒号在 vocab 中的 index 是 9，

所以使用的 onehot 向量就是只有 9 为 1 的向量

所以 $W_{xh}(\text{hidden_size}, \text{vocab_size})$ 使用第九维的向量作为“冒号”的表示加上 W_{hh} 与 h_{t-1} 的乘积映射到同一个空间，加上 bias b_h

经过 tanh 当前状态后只有一部分被激活进行下面的运算

后面的所有参数都会使用到: W_{hy}, b_h, b_y

值得注意的是：最终 softmax 模拟采样得到最终的结果不一定是得分最高的，而是很有可能得分很高。

结论分析与体会：

1，RNN 是自然语言处理中的基本神经网络框架，现在的很多问题都是基于这样一个框架完成的。

2，不同于计算机视觉中 CNN 框架的是：RNN 中拥有非常多的变种神经元，并且需要考虑时序按照 BPTT 展开。

3，RNN 中不像 CNN，RNN 会丢失很多输入中的信息。

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，Gumbel softmax 到底做了一件什么样的事情？

答：Gumbel softmax 要完成两个目标①可导，可反向传播 ②随机扰动，与 softmax 一致，并且可以达到模拟随机采样的效果。