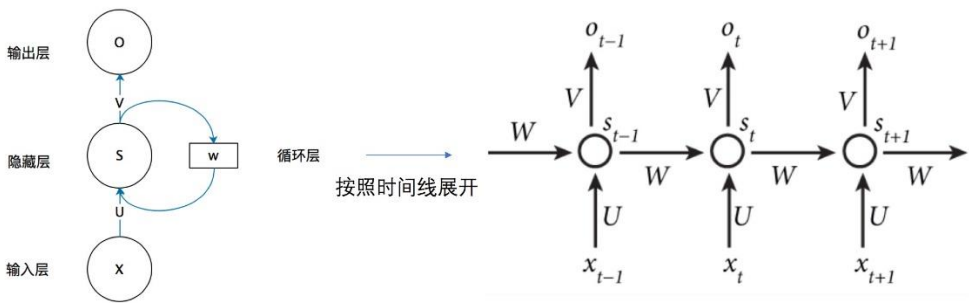
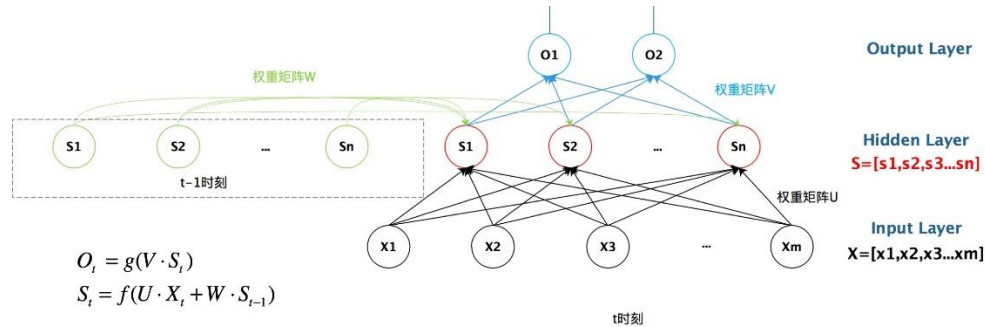


计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目： Fun with RNNs		学号： 201600181073
日期： 2019. 4. 23	班级： 智能 16	姓名： 唐超
<p>实验目的：</p> <p>用 RNN 实现给定初始字符或字符串生成合理的连续文本。</p> <ul style="list-style-type: none">● Part 1: 补全 min-char-code.py, 训练生成文本的 RNN;● Part 2: 对 softmax 函数的 temperature 参数取不同值, 并比较不同 temperature 值对初始字符生成文本的影响;● Part 3: 给定 starter string, 生成后续文本;● Part 4: 用 RNN 生成的文本中, 换行符或空格通常跟随冒号 (即 “:”) 字符。在训练好的权重数据中, 确定对此行为负责的特定权重并解释。		
<p>实验软件和硬件环境：</p> <p>Jupyter notebook&python3.6</p>		
<p>实验原理和方法：</p> <p>RNN 的主要结构：</p>  <p>某一时刻的网络结构：</p>  <p>(循环神经网络的隐藏层的值 s 不仅仅取决于当前这次的输入 x, 还取决于上一次隐藏层的值 s)</p>		

实验步骤：（不要求罗列完整源代码）

Part 1

主要过程见上一栏，具体代码见 min-char-code.ipynb.

Part 2

在生成文本时，改变网络的 softmax 函数的计算公式，引入 temperature 参数 τ ：

$$\text{softmax}(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^K e^{\frac{Q(k)}{\tau}}}$$

改变 temperature 生成文本的 sample 函数如下：

```
def sampleWithTemperature(h, seed_ix, n, tpt):
    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/tpt) / np.sum(np.exp(y/tpt))
        ix = np.random.choice(range(vocab_size), p=p.ravel())
        x = np.zeros((vocab_size, 1))
        x[ix] = 1
        ixes.append(ix)
    return ixes
```

采用已训练好的 RNN 权重，在不同的 temperature 下用初始字符生成的文本如下。

```
starter = s    n = 50    temperature = 1
-----
st sow me lates poess fed not
That thun wagh king t

starter = s    n = 50    temperature = 3
-----
seenonk;, lawsy?
C
TEtrIp
bloeme shonably
Oy udn,
I
```

```

starter = s      n = 50      temperature = 10
-----
srs-sIV.WHwZ'm
BagAveI!w
L,QUfw jYou.h e:
d!knce&Ni

```

可以看出，当 temperature 取值越大时，生成的文本越莫名其妙，这一点可以从 softmax 函数的公式中得到解释， τ 越大， $e^{\frac{Q(k)}{\tau}}$ 越趋近于 1，生成每个字符的可能性越趋于相等，即与训练的 RNN 无关，趋近于等可能性地从所有字符中随机选择。

Part 3

在给定初始字符串的情况下，生成后续文本，需要计算初始字符串的最后一个字符的隐藏层状态，具体函数如下：

```

def sampleFromString(starter, n, tpt):
    starterIx = [char_to_ix[ch] for ch in starter]
    # compute the hidden activity h at the end of the starter
    h = np.zeros((hidden_size, 1))
    for t in range(len(starterIx)):
        x = np.zeros((vocab_size, 1))
        x[starterIx[t]] = 1
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    # generate text
    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/tpt) / np.sum(np.exp(y/tpt))
        ix = np.random.choice(range(vocab_size), p=p.ravel()) #
        x = np.zeros((vocab_size, 1))
        x[ix] = 1
        ixes.append(ix)

    continuation = "".join([ix_to_char[ix] for ix in ixes])
    fullText = starter+continuation
    return fullText

```

Examples:

```

starter = she      n = 50      temperature = 1
-----
she and Juke we lust afsietie, jeen attay'
Core hath

```

```

starter = she is    n = 50      temperature = 1
-----
she is'd dead, the with net cortued sharchaing, be hing

```

```

starter = she is a      n = 50      temperature = 1
-----
she is am my huss myeuthuse balt:
I, for her.

FIDUCHIUS:

starter = she is a little      n = 50      temperature = 1
-----
she is a littled coidn solt,
I thon clanice, here have now, shy h

starter = everyone      n = 50      temperature = 1
-----
everyones son of the my he's our whelct'd a tolader oauch

```

Part 4

用 RNN 生成的文本中，换行符或空格通常跟随冒号（即“:”）字符。为了找到权重的哪一部分对这一现象的影响最大，即找到权重的哪一部分对生成“\n”或“ ”的概率值贡献最大，采用以下步骤：

1. 找到“:”作为输入的 RNN 隐藏层平均状态，具体地：

用随机字符生成一定长度的文本，记录此文本序列的最后一个字符的隐藏层状态 h_i ，把 h_i 作为“:”的前一时刻的隐藏层，计算得到“:”的隐藏层 h_i' 。重复这一过程，得到 h_1', h_2', \dots ，最后求其均值，得到 \bar{h} 。用 \bar{h} 可以求得“:”后生成“\n”或“ ”的概率，虽然两者的概率不稳定，但两者的和一般都超过了 90%，如

```

'\n' with probability (60.71%) after ':'
' ' with probability (36.62%) after ':'

```

2. 将 Why 对应于计算“\n”或“ ”的行向量与 \bar{h} 做 elementwise product（如果做向量内积即为得到“\n”或“ ”的概率比重），得到同样大小的向量，找出其最大的 10 个元素的索引，就得到了 Why 中的 most relevant weights 的位置。
3. 对 Wxh，找到与“:”有关的列向量 $Wxh[:, \text{init_ix}]$ ，求其均值 avg_wxh ， $Wxh[:, \text{init_ix}]$ 中与得到 Why 中的 most relevant weights 有关的且大于 avg_wxh 的元素，即为 Wxh 中的 most relevant weights。

最终实验结果如下：

- “\n”

Weights Involved:

Wxh at[:, 9]

Why at [0, :]

Most relevant weighs:

Wxh at [[100, 187, 166, 108, 114], 9]

Why at [0, [100, 187, 166, 108, 114]]

- “ ”

Weights Involved:

Wxh at[:, 9]

Why at [2, :]

Most relevant weighs:

Wxh at [[100, 108, 125, 187, 114], 9]

Why at [2, [100, 108, 125, 187, 114]]

结论分析与体会：

体会：通过这次实验，学习了用 RNN 生成文本和寻找权重中对某一现象负责的部分的方法，加深了对 RNN 的认识。

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. 刚开始做时，对 RNN 的网络结构没有完全理解，导致无从下手，后来又重新从网上学习 RNN 的讲解和代码；
2. 采用作业文档中的方法读取权重数据时 import 部分出错，查阅资料发现 python3 中直接 import pickle 即可；
3. Part4 中将步骤 2 中的 Why 对应于计算 “\n” 或 “ ” 的行向量与 \bar{h} 的运算错误视为向量内积，导致后续过程不能理解。