NLP Text summarization part 2

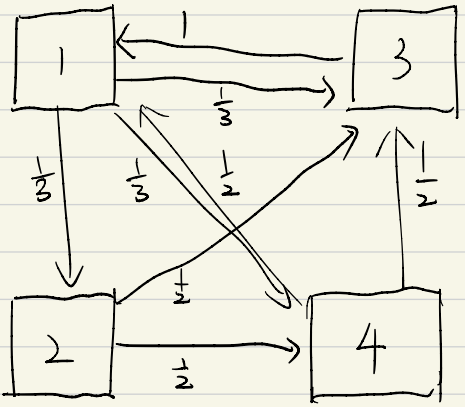
一般文本摘要有两种方式：

① 记号笔：所有的内容都来自原文 ⇒ extractive Summarization

② 钢笔：写出来，就是看完之后做总结做消化
⇒ Abstractive Summarization 抽象式的摘要

- 文本摘要
  - 抽取式
    - 无监督抽取
      - TextRank
      - Two-level sparse Representation
      - RA-MDS
      - DAE
      - P4CSUM
    - 有监督抽取
      - R2N2
      - NeuralSum
      - SummaRuNNer
      - BeroSum
      - STRASS
  - 生成式

Google: PageRank 对网页排序



	1	2	3	4
1	0	0	1	1/2
2	1/3	0	0	0
3	1/3	1/2	0	1/2
4	1/3	1/2	0	0

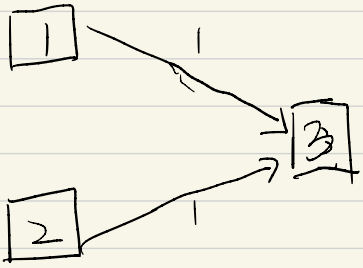
$$V=\begin{pmatrix}0.25\\0.25\\0.25\\0.25\end{pmatrix} \quad AV=\begin{pmatrix}0.37\\0.08\\0.33\\0.20\end{pmatrix} \quad A^2V=A(AV)=A\begin{pmatrix}0.37\\0.08\\0.33\\0.20\end{pmatrix}=\begin{pmatrix}0.43\\0.12\\0.27\\0.16\end{pmatrix}$$

$$A^3V=\begin{pmatrix}0.35\\0.14\\0.29\\0.22\end{pmatrix} \cdots\cdots A^7V=\begin{pmatrix}0.38\\0.12\\0.29\\0.19\end{pmatrix} \quad A^8V=\begin{pmatrix}0.38\\0.12\\0.29\\0.19\end{pmatrix}$$

converge

Two problems in PageRank

① Nodes with no outgoing edges (dangling nodes)



3 没有出边

$$V_0=\begin{bmatrix}\frac{1}{3}\\\frac{1}{3}\\\frac{1}{3}\end{bmatrix} \cdots V_\infty=\begin{bmatrix}0\\0\\0\end{bmatrix}$$

$$V_1=\begin{bmatrix}0&0&0\\0&0&0\\1&1&0\end{bmatrix}\cdot\begin{bmatrix}\frac{1}{3}\\\frac{1}{3}\\\frac{1}{3}\end{bmatrix}=\begin{bmatrix}0\\0\\\frac{2}{3}\end{bmatrix}$$

damping factor → solution

Text Rank 也是用了 Page Rank 的思想

TextRank Process

1. 把所有的文章中的句子，连接起来
2. 把文本分割成一个个的句子
3. 对每个句子做向量化
4. 计算句子之间的相似度，并存储到矩阵中
5. 把相似度矩阵转化为图，其中句子是节点，相似度是节点间的权重
6. 取到top-k权重的句子，作为最终的摘要

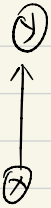
networkx => PageRank算法

生成式算法：

Sequence Components：

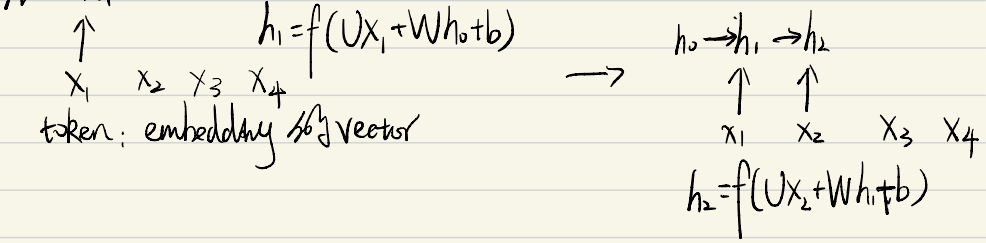
Recurrent Neural Network：

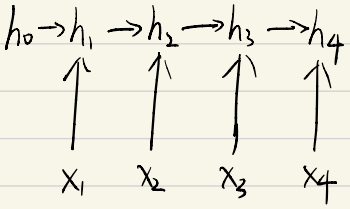
Start from a single layer



$y=f(Wx+b)$

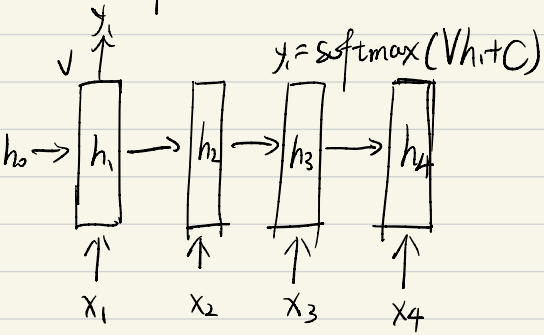
classical RNN：



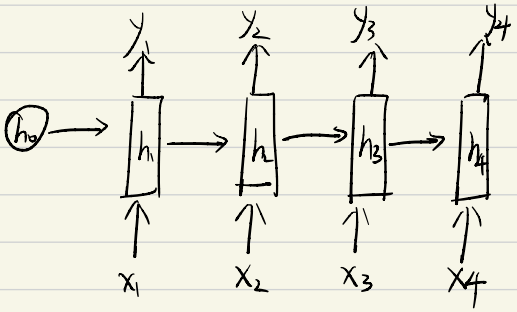


每一步使用的参数 U、W、b都是一样的，也就是说每个步骤的参数都是共享的，这是RNN的重要特点，要牢记

RNN Output



不同的RNN结构：N vs N



常用于命名实体识别

计算视频中每一帧的分类标签

因要对每一帧进行计算，因此输入和输出序列等长

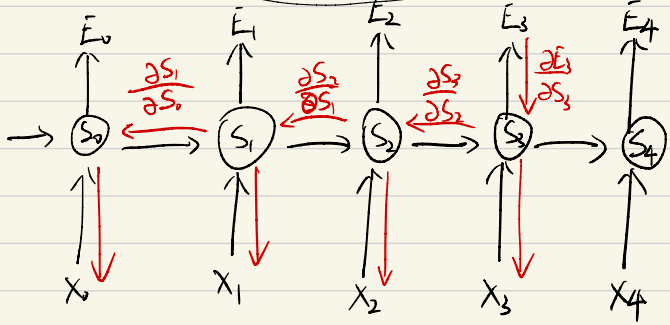
Backpropagate Through Time (BPTT) 反向传播

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W} \qquad \frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial s_3} \frac{\partial s_3}{\partial W} \qquad \text{But } s_3 = \tanh(Ux_t + Ws_2)$$

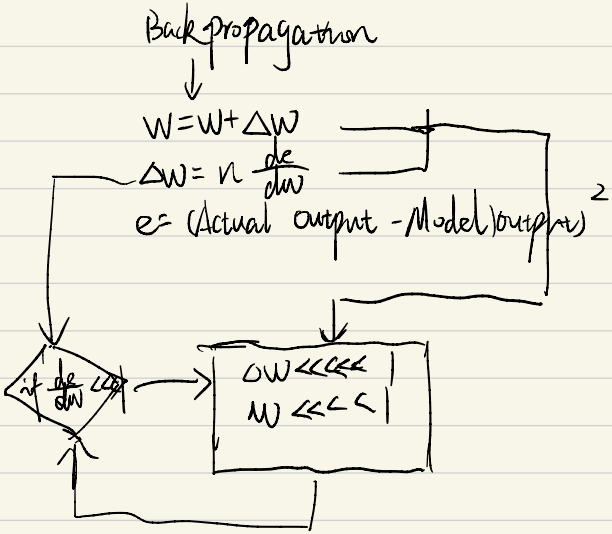
But $S_3 = \tanh(WX_4 + WS_2)$ S_3 depends on S_2, which depends on W and S_1, and so on.

0.01

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial S_3} \frac{\partial S_3}{\partial S_k} \frac{\partial S_k}{\partial W}$$

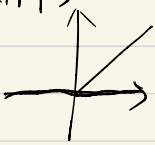


Vanishing Gradient 梯度消失



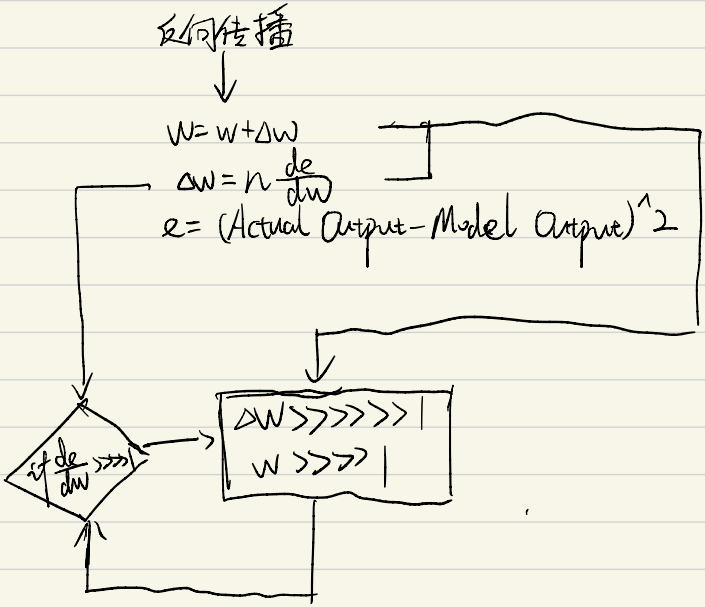
Solve Vanishing Gradient (梯度消失)

- ReLU activation function
- LSTM, GRU



ReLU

Exploding Gradient 梯度爆炸



Solve Exploding Gradient

① 梯度裁剪 Common solution: clipping gradient

if $\|g\| > v$ · g: gradient

· v: threshold

If norm of gradient exceeds some threshold, clip it!

②

Truncated BPTT

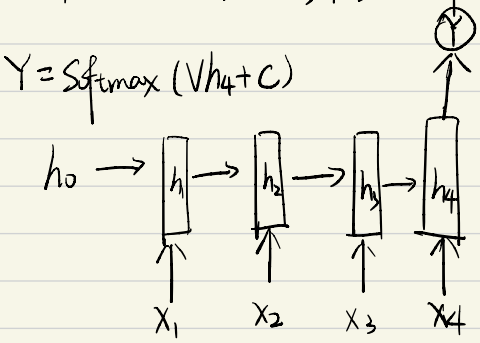
k1: 前向传播的时间步。一般来说，这个参数影响模型训练的快慢，即权重更新的频率

k2: 使用BPTT反向传播的时间步，一般来说，这个参数需要大一点，这样网络能更好的学习序列信息，但这个参数太大的话会导致梯度消失

③ RMSprop to adjust learning rate

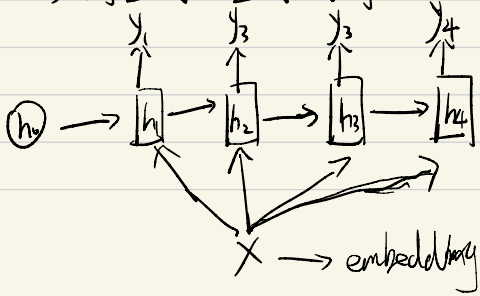
Bidirectional RNN

NVS 1 ⇒ 文本分类，情感分析



1 VSN ⇒ 从图像生成文字(image caption)，此时输入的X就是图像的特征，而输出的y序列就是一段句子

从类别生成语音或音乐等



其实都是RNN，只是输入和输出不一样了

图片通过CNN映射到向量空间中变成了feature map

普通RNN：tf.keras.layers.SimpleRNN(4, input_shape=(3,2))

4：h_0 hidden state

(3,2)：整个例的长度，embedding的大小

输入的部分加一个embedding layer：onehot ⇒ 向量

两个trick，①使用已经有的embedding作为参数

②embedding_lookup

embedding_initializer = tf.keras.initializers.Constant( )

embedding_lookup(..., feature_batch)

多输出的RNN：

tf.keras.layers.SimpleRNN(4, input_shape=( ), return_sequences=True)

每个时间步都加层：

tf.keras.layers.TimeDistributed(tf.keras.layers.Dense( ))

多层叠加：

```
model = tf.keras.sequential( )
model.add(tf.keras.layers.SimpleRNN(..., return_sequences=True)
model.add( ... ( ..., return_sequences=True)
model.add( ... )
```

这一步一定要有，否则就没有向上输出了

双向的RNN：

model.add(tf.keras.layers.Bidirectional( ))

RNN的优点与缺点：

Advantage: RNN can model sequence of data(i.e. time series) so that each sample can be assumed to be dependent on previous ones

· share parameters 参数共享

Disadvantage: ① 梯度消失和梯度爆炸

② 训练RNN很困难

③ 不能捕获很长的序列

LSTM

The Gates

Forget gate: Decides how much of past you should remember

$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ 因为用了sigmoid 所以$0 < f_t < 1$

↓

bias

Update Gate: decides how much of this unit is add to current state

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ → range(0, 1)

$t' = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$ → range(-1, 1)

Memory pipe:

$$C_t = f_t C_{t-1} + i_t t$$

Output Gate: Decides which part of the current cell makes it to the output

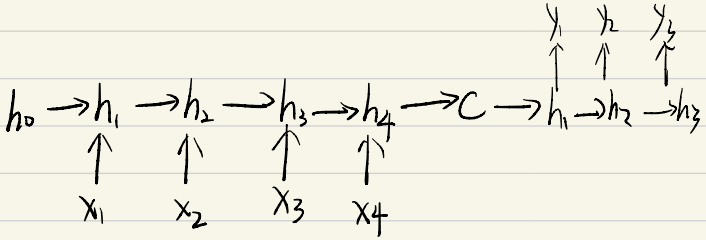
$$O_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = O_t * \tanh(C_t)$$

GRU: LSTM的变种　减少LSTM的参数　减少了门用个门控制

LSTM & GRU的对比:
GRU的参数更少，收敛速度更快，因此其实际花费时间步很多，这大大加速迭代过程
从表现上讲，两者之间没有绝对定论，两者performance差距并不大，这次调参带来的效果明显，不如在LSTM or GRU的激活函数(ex tanh改为tanh变体)和权重初始化上下功夫
一般来说，选GRU作为基本单元，因为它收敛速度快，可加快试验迭代，快速迭代

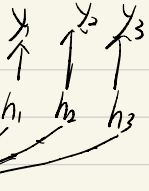
Seq2Seq 基本是N vs M

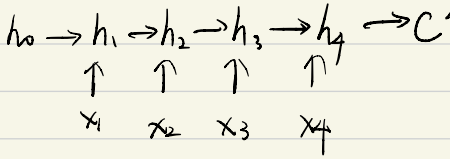
N vs M
(1) $c = h_4$
(2) $c = q(h_4)$
(3) $c = q(h_1, h_2, h_3, h_4)$



Encoder-Decoder Architecture

translate，语音，文本摘要





变种

paper: Sequence to Sequence learning with Neural Networks

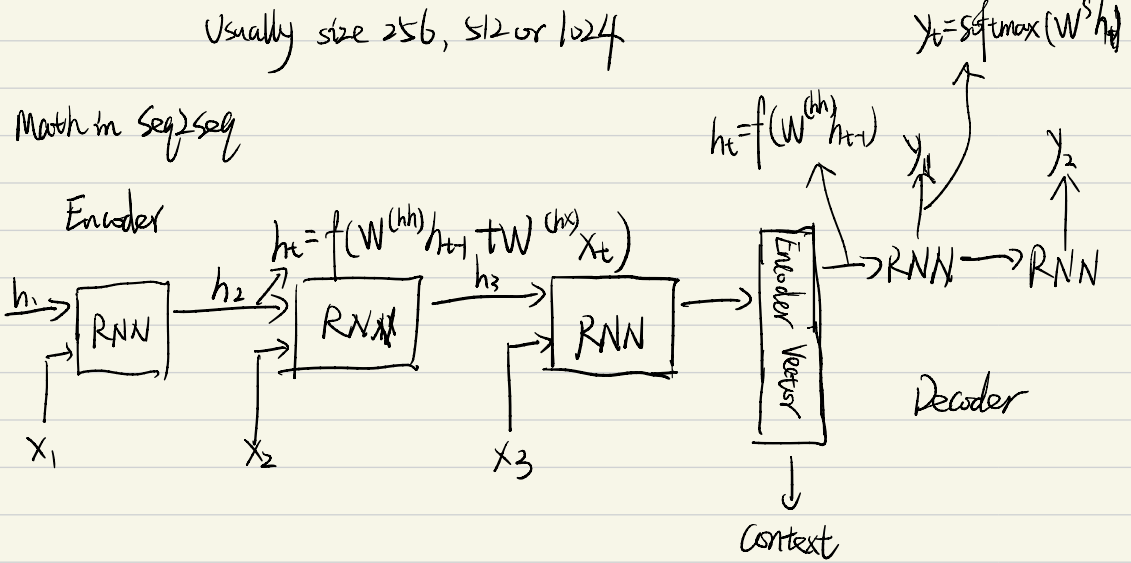
Input: use pre-trained embeddings or train our own embedding on our dataset

Context Vector

encoder → Context → decoder

Usually size 256, 512 or 1024

Math in Seq2Seq



Attention mechanism

encoder 部分实际不能保存那么多信息，注意力机制会有选择地关注input

Additive attention:

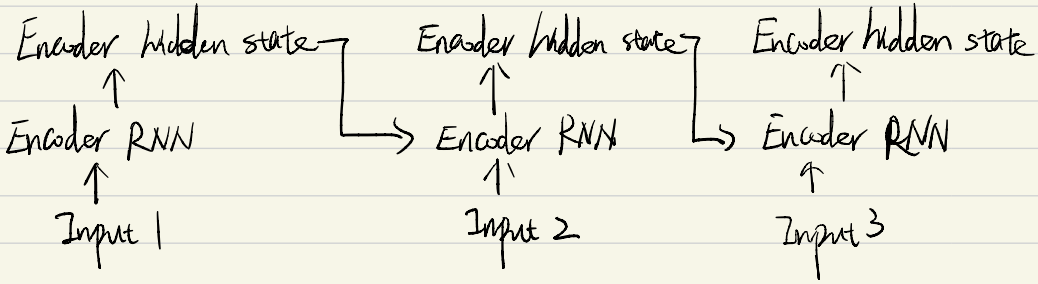
seq2seq with attention:

用当前的输入信息跟之前的每个hidden state作一个打分，来看看多相关，打完分之后做一个softmax，作出概率分布，加和算context vector

怎么加和：$\sum$ weight * encoder hidden state

加性attention how to compute:

①Producing the encoder hidden states



②③④ Calculating Alignment Scores

$$Score_{alignment} = W_{combined} \cdot \tanh(W_{decoder} \cdot H_{decoder} + W_{encoder} \cdot H_{encoder})$$

addition

$W_{combined}$：学习来的

$W_{decoder}$：学习来的

$W_{encoder}$：学习来的

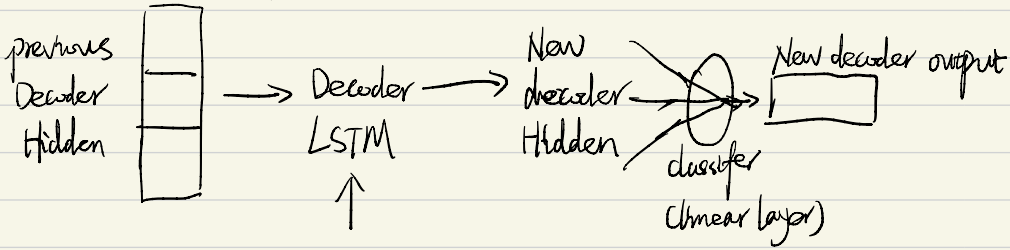
⑤ Softmax the alignment scores

Softmax (alignment score) = attention weight

⑥ Calculating the Context Vector

Encoder outputs * Attention weights = Context Vector

⑦ Decoding the Output



Context Vector + Embedding of previous decoder output

multi attention 和 additive attention 的不同在于：

add: Concat: $score_{align} = W \cdot \tanh(W_{combined}(H_{encoder} + H_{decoder}))$

Multi: General: $score_{align} = W(H_{encoder} \cdot H_{decoder})$

LSTM:

return_state：返回 hidden & context state

最终保存的是 embedding

attention 机制：

tensorflow-addons　　seq2seq.LuongAttention

loss function：cross entropy

Teacher forcing：第一个输出错了会导致下一个也出错，后面会连续出错

Ground Truth：强制改输入

cons：

训练时有 Ground Truth，但是 predict 时是没有的 => exposure bias

解决方法：① Beam Search

② Scheduled Sampling：抛硬币的方法来决定是否使用 Ground Truth

一开始高概率 Ground Truth，后面随着学习越来越好，然后一点点下降 Ground Truth 概率