

## Text summarization part 4

OOV & word-repetition 问题的改进

实际 baseline：先找 simple model 做出来，然后再去慢慢优化

OOV: out of vocabulary

什么是 OOV？

ex: tensor → word2vec →   

tensorflow → word2vec →   

tensorflow → word2vec → X

Solution:

① ignore

② UNK work; reserve a dimension in feature space, it may eliminate some impact of OOV word, but very limited. Especially the OOV word plays an important role in the NLP task.

The ~~ecotax~~ portico in Pont-de-Buis  
Le portique ~~ecotaxe~~ de Pont-de-Buis  
↓

The <unk> portico in <unk>  
Le <unk> <unk> de <unk>

### ③ Enlarge Vocabulary

## ④ Spell check

If you have observed a lot of OOV words are caused by typos (e.g.

chatbot), it is pretty natural to use spell check, but one caveat is that, the spell check may over-correct some correct words, you have to use the corrected words as additional feature and combine this with other solutions together.

## Subword generation

For a word, we generate character n-grams of length 3 to 6 present in it.

eatingly → <eatingly>

Then, we generate character n-grams of length n

<eat'my>

Thus, we get a list of character n-grams for a word.

3-grams eat eat ath thn ing ng  
          leatmly

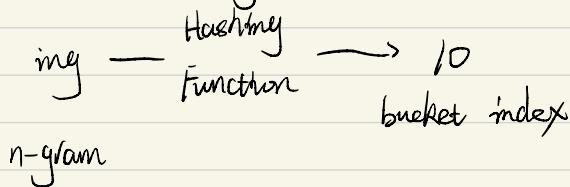
## sub-word generation

Since there can be huge number of unique n-grams, we apply hashing to bound the memory requirements.

hash 其实是很好的降维的方法 FNV hash

降维带来的问题：sub-word的准确性下降

Each character n-gram is hashed to an integer between 1 to B



## Byte Pair Encoding (BPE)

1. 准备足够的训练语料

2. 确定期望的 subword 词表大小

3. 将单词拆分为字符串，并在末尾添加后缀“</w>”，统计单词频率。  
本阶段的 subword 的粒度是字符。

ex, “low”的频率为5,那么我们将其改写为“low</w>”: 5

4. 统计每一个连续字符对的出现频率，选择最高频者合并成新的 subword

5. 重复第4步直到达到第二步设定的subword词表大小或下一个最高频的字节对出现频率为1

tokenizer的话：用python写的，效果没那么快，但应用性更强

GPT-2 & RoBERTa 使用的是BPE

Word Piece:

1. 准备足够大的训练语料

2. 确定期望的subword词表大小

3. 将单词拆分成字符序列

4. 基于第3步数据训练语言模型

5. 从所有可能的subword单元中选择加入语言模型后能最大程度地增加训练数据概率的单元作为新单元

6. 重复第5步直到达到第二步设定的subword词表大小或概率增量低于某一阈值

Unigram Language Model

1. 准备足够大的训练语料

2. 确定期望的subword词表大小

3. 给定词序列优化下一个词出现的概率

4. 计算每个subword的损失

5. 基于损失对subword排序并保留前N%，为避免OOV

6. 重复第3至第5步直到达到第二步设定的subword词表大小或第5步的结果

不再变化

## Pointer-Generator Network (PGN)

seq2seq Attention Mechanism

output

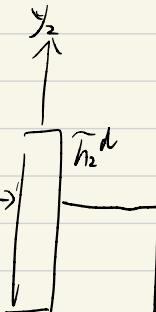
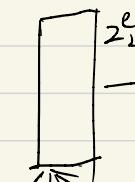
Context  
vector

Attention  
Weights

hidden

Embedding

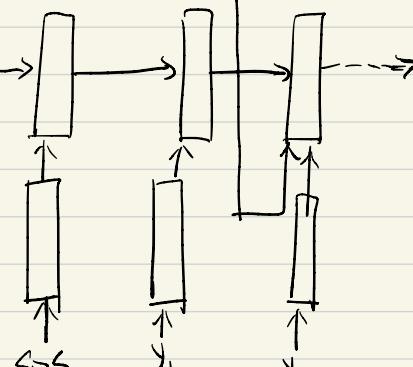
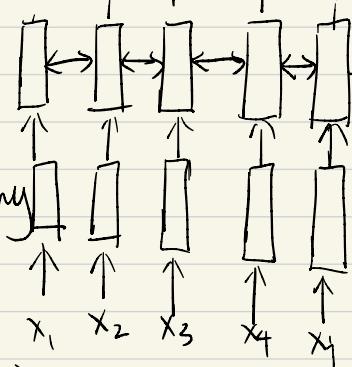
Input



$y_2$

$\tilde{h}_2^d$

$h_2^d$



Encoder

Decoder

$$u_j^i = V^T \tanh(W_1 e_j + W_2 d_i) \quad j \in \{1, \dots, n\}$$

$$a_j^i = \text{softmax}(u_j^i) \quad j \in \{1, \dots, n\}$$

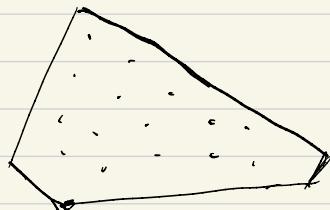
$$d_i^f = \sum_{j=1}^n a_j^i e_j$$

其中  $e_j$  是 encoder 的隐状态，而  $d_i$  是 decoder 的隐状态； $V, W_1, W_2$  都是可学习的参数，在得到  $u_j$  之后对其进行 softmax 操作即得到  $a_j$ 。

### Pointer Network (Ptr-Net)

什么是 凸包 (Convex Hull)，在一个多边形边缘或者内部任意两个点的连线都包含在多边形边界或内部。

正式定义：包含点集合中所有点的最小凸多边形称为凸包



正式定义：  
包含点集合中所有点的最小凸多边形称为凸包

Copy Net

(b) Generate-Merge & Copy-Merge

(c) State Update

Attention-based Encoder-Decoder (RNN Search)

Copy: Per-Net 去得到

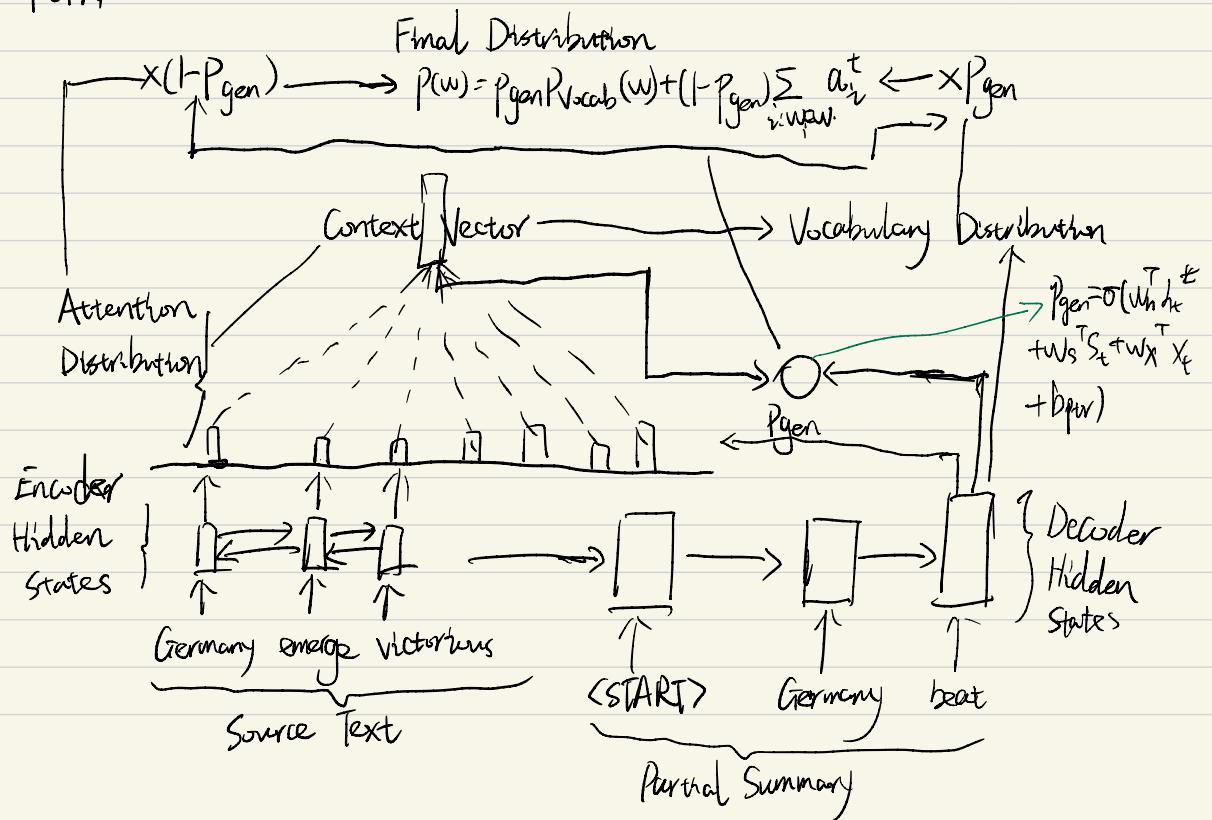
$$u_j^i = \sqrt{t} \tanh(W_i e_j + W_d d_i) \quad j \in \{1, \dots, n\}$$

$$a_j^i = \text{softmax}(u_j^i) \quad j \in \{1, \dots, n\}$$

$$d_i = \sum_{j=1}^n a_j^i e_j$$

其中  $e_j$  是 encoder 的隐状态，而  $d_i$  是 decoder 的隐状态， $v$ ,  $W_1$ ,  $W_2$  都是可学习的参数，在得到  $u_j^i$  之后对其进行 softmax 操作即得到  $a_j^i$ 。

PGN



## Coverage Mechanism

$$C^t = \sum_{i=0}^{t-1} a^i$$

$$e_i^t = V^T \tanh(W_h h_i + W_s s_t + W_c c_i^t + b_{attn})$$

$$\text{CoverLoss}_t = \sum_i \min(a_i^t, c_i^t)$$

## Text Summarization in Alibaba

### Multi-Source Pointer-Generator Network

商品标题摘要这个特殊问题天然存在对模型的两个限制：

- 1) 摘要中不能引入不相关信息
- 2) 摘要中必须保留源文本的关键信息(如品牌名和商品名)

其实与PGN的思想是一致的

## Hyper-Parameter Tuning

### Good Coding Style

将各个参数的设置部分集中在一起。

如果参数的设置分布在代码的各个地方，那么修改的过程想必会非常痛苦。

可以输出模型的损失函数值以及训练集和验证集上的准确率

可以考虑设计一个子程序，可以根据给定的参数，启动训练并监控周期性保存评估结果。再由一个主程序，分配参数以及并行启动一系列子程序。

## General to Specific

①建议先参考相关论文，以论文中给出的参数作为初始参数。

至少论文中的参数是个不差的结果。

②如果找不到参考，那么只能自己尝试了。可以先从比较重要，对实验结果影响比较大的参数开始，同时固定其他参数，得到一个差不多的结果以后，在这个结果的基础上，调整其他参数。

例如学习率一般就比正则值，dropout值重要的话，学习率设置的不合适，不仅结果可能变差，模型甚至无法收敛。

③如果实在找不到一组参数让模型收敛，那就需要检查，是不是其他地方出了问题，例如模型实现，数据等等。

## Speed up Experiment

· 对训练数据进行采样，例如原来 100W 条数据，先采样成 1W 进行实验。

· 减少训练类别。例如手写数字识别任务，本来是 10 个类别，那么我们可以先在 2 个类别上训练，看结果如何。

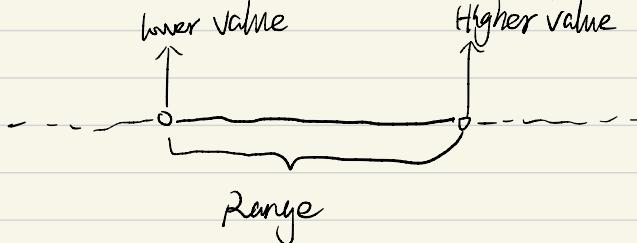
刚开始时模型很大，可以通过减少 layer 和 dense 来减少。

## HP Range

建议优先在对数尺度上进行超参数搜索。比较典型的是学习率和正则化项，我们可以从诸如 0.001, 0.01, 0.1, 1, 10, 100 的阶数进行尝试。

因为他们对训练的影响是相乘的效果。

不过有些参数，还是建议在原始尺度上进行搜索，例如 dropout 值：0.3, 0.5, 0.7



## Experiment Number

learning rate: 1 0.1 0.01 0.001, 一般从开始尝试，很少见 learning rate 大于 10<sup>-6</sup>

学习率一般要随着训练进行衰减。衰减系数一般是 0.5。

衰减时机可以是验证集准确率不再上升时，或固定训练多少个周期以后。

不过更建议使用自适应梯度的办法，例如 adam, adadelta, rmsprop 等，这些一般使用相关论文提供的默认值即可，可以避免再费劲调节学习率。

对 RNN 来说，有个经验，如果 RNN 处理的序列比较长，或 RNN 层数比较多，那么 learning rate 一般大一些比较好，否则有可能出现结果不收敛，甚至 Nan 等问题。

网络层数：先从 1 层开始

每层结点数：16 32 64 超过 100 的情况比较少见，超过 1W 很少见，没见过

batch size：128 上下开始，batch size 值增加，仍可能提高训练速度，但是有可能

收敛结果变差。如果显存大且允许，可以考虑从一个比较大的值开始尝试，因为 batch size 太大，一般不会对结果有太大的影响，而 batch size 太小的话，结果有可能很差。

clip C (梯度裁剪)：限制最大梯度其实是  $\text{value} = \sqrt{w_1^2 + w_2^2 + \dots}$ ，如果 value 超过阈值，就算一个衰减系数，让 value 的值等于阈值：5, 10, 15

dropout: 0.5

L2 正则：1.0 超过 1.0 的很少见

词向量 embedding 大小：128, 256

正负样本比例：这个是非常忽视，但是在很多分类问题上，又非常重要的参数。  
很多人往往习惯使用训练数据中默认的正负类别比例，当训练数据非常不平衡的时候，模型很有可能会偏向数目大的类别，从而影响最终训练结果。

除了尝试训练数据默认的正负类别比例之外，建议对数目较小的样本做过采样，例如进行复制。提高他们的比例，看效果如何，这个对多类问题同样适用。

在使用 mini-batch 方法进行训练的时候，尽量让一个 batch 内，各类别的比例平衡，这个在图像识别等多类任务上非常重。

Loss don't Decrease

train loss 不断下降，test loss 不断下降，说明网络仍在学习；

train loss 不断下降，test loss 趋于不变，说明网络过拟合；

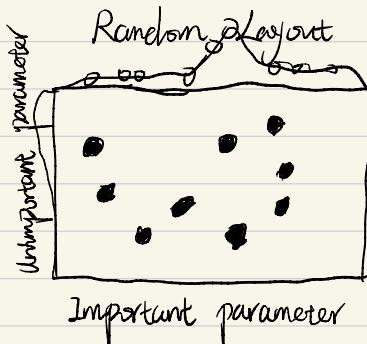
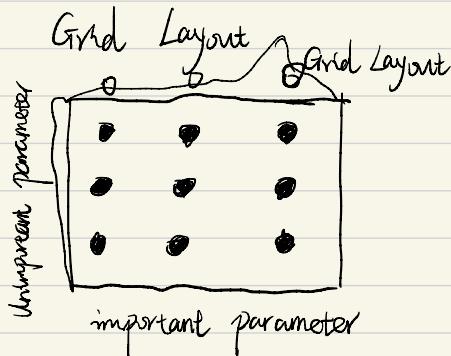
train loss 趋于不变，test loss 不断下降，说明数据集 100% 有问题；

train loss 趋于不变，test loss 趋于不变，说明学习遇到瓶颈，需要减少学习率或批量数；

train loss不断上升, test loss不断上升, 说明网络结构设计不当, 训练超参数设置不当, 数据集经过清洗等问题。

## Auto Tuning

Exhaustive search of the search space



Keras Tuner → tensorflow的一个自动调参仓库