OOV 与 word_repetion 问题的改进

一、Seq2seq 模型出现的问题

1、encoder 到实际输出有一定距离，反向传播收到限制（当 seq2seq 中 decoder 部分特别长，最终蜕变成 语言模型类型）

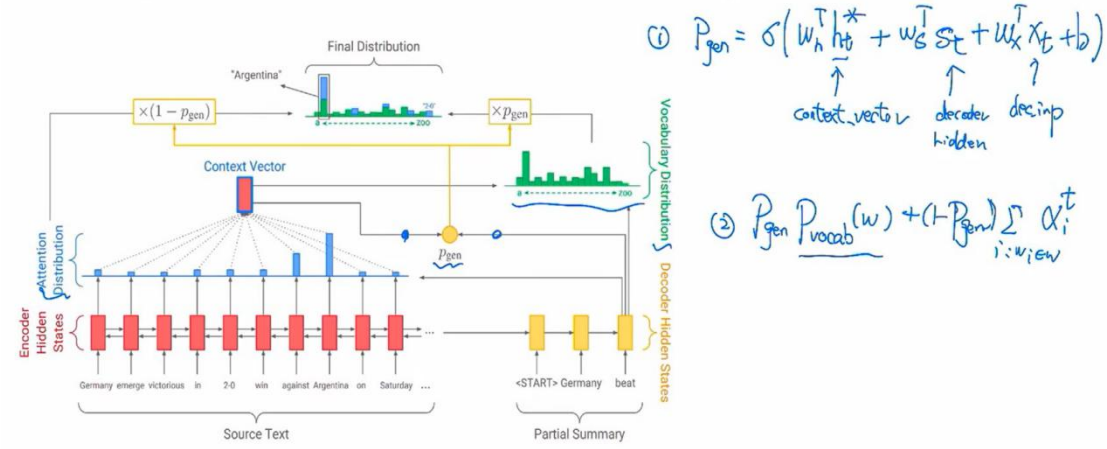
2、OOV 问题（例如:”比分 2-1” 与“比分 2-0”是有区别的）

3、Word-repetition 问题（摘要结果生成重复信息）

二、pointer-generator network

oov 问题描述：我们建立一个 vocab 字典，当预测生成不在这个字典之中，默认返回<unk>;

pgn 原理：设置 p_gen 门，使可以有几率从 context 输入中找回这个词，p_gen 通过 attention 的概率分布中的关注点去找对应的 token_id，然而要找回 token_id,unk_id 与 oov_id 就要有一定关联。因此，在每一次输入设立 oov_vocab，用来放置本次输入没在 vocab 中的 token 并编上号(vocab_size 往上编),输入在转 token_id 碰到的是 unk_id,经过 atenttion 概率与预测概率的融合，有可能变成 vocab_size+1,+2...，从而找到真正的 token_id。从代码层面分析，①从 context 中不在 vocab 中的放在 oov_vocab,vocab 的 id 扩充之后叫 vocab_extend；encode 输入为 context 在 vocab 中对应的 id(oov 部分均是 unk)②decode 输入与 label 是错位的，输入：abstract 在 vocab 中对应 id(包含 unk)，label：abstract 在 vocab_extend 中对应 id(包含扩展)。③label 包含 oov_id，需将预测概率维度 vocab_size 扩展至 vocab_extend_size;encode 的输入 oov_id 都融在 unk 上了,需将 attention 中维度扩展至 vocab_extend_size(decode 的 label 是 oov_word,会反向通过 attention 找到特定位置的 unk,通过训练，unk 也能找到 oov_word)。④将扩展后的预测概率与 attention 概率进行融合一起训练，最终会训练出 unk 与 oov_word 的关系。



Pgn 网络图

具体公式如上图，p_gen 门与 attention、当前 decoder_hidden、dec_inp 有关，最

终训练出 p_gen（数）；最终的概率分布=(1-p_gen)*att 概率分布+p_gen*预测概率分布。

（1）attention 的概率分布如何扩充到 vocab_extend_size 维度？

答：attention 在扩展时采用的是 encode_extend_inp(包含 oov_id)，它是与 enc_inp（oov_id 用 unk 代替）一起传进来的；如果输入的 token_id 是 11,将 11 的位置采用 attention 值作为概率，其余没命中的置为 0.

（2）P_vocab 概率分布如何扩充到 vocab_extend_size 维度？

答：p_vocab 的扩展，直接后面多余的 size 部分补 0.

（3）P_vocab 概率与 attention 概率如何融合？

答：在扩展之后，维度都是一样的，直接按公式，与门 p_gen 相乘完，按位相加。

（4）PGN 的优势：

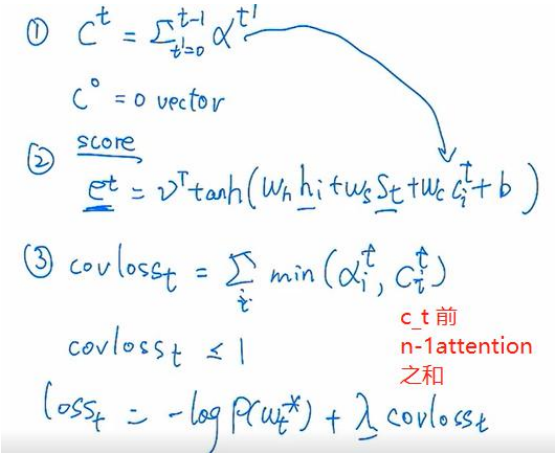
①能够很容易的复制输入的文本内容，可以通过 P_gen 调节

②能够从输入的文本内容中复制 oov 词汇，可以采用较小的词汇表，较少的计算量和存储空间

③训练很快，训练过程用更少的迭代次数就能取一样的效果

三、coverage 机制

coverage 机制最早于机器翻译中使用，用于在训练阶段解决生成重复的一种策略。原理：将 c_t 放入到 attention 里的 mlp 融合中去学习，c_t 为前 n-1 个 attention_weight 之和，loss 则加上 covloss，具体公式如下图：



公式图

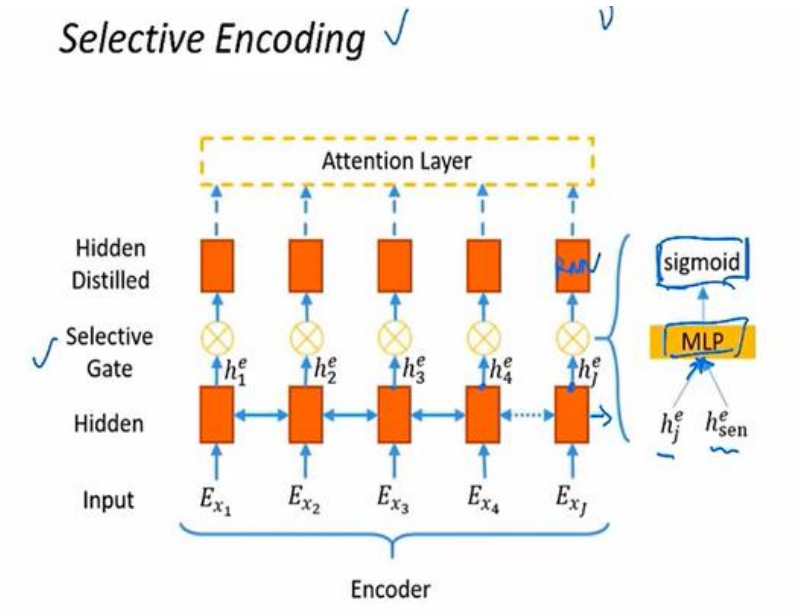
Loss 分析：covloss 采用对 attention 进行惩罚，上一次关注的词下一次就尽量少关注来进行惩罚。

代码分析：①attention 累加融合到 attention 计算中：首次进来,不放到 MLP 融合中,coverage 为生成的 attention_weight；非首次进来，放到 MLP 融合，求得 attention_weight，coverage+=attention_weight;②遍历所有 attention,每一次与 coverage 比较，小的进行 reduce_sum,然后存储起来，求平均返回。

四、模型改进

1、改进模型的角度-Encode

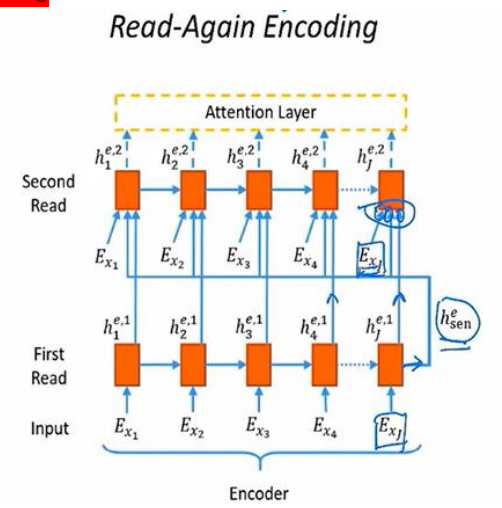
1.1 selective Encoding



selective Encoding

中间设置门机构来对 hidden_state 进行选择。每一个门的比率，采用当前的 hidden_state 与最后时刻的 hidden_state 进行 MLP 融合。

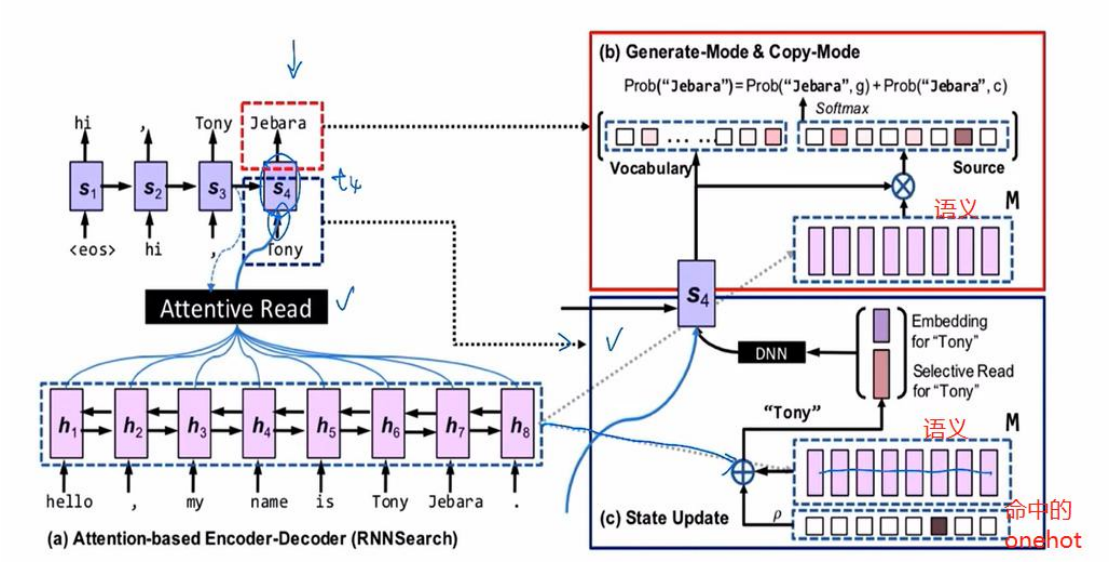
1.2 Read-Again Encoding



Read-again encoding

采用 2 层的 lstm，第 2 层的输入采用的每一个时刻对应的 hidden_state,最后的 hidden_state，每一时刻对应的输入。

1.3 CopyNet



Copynet

Copynet 在 decode 的输入做的改进：根据 s3 将选择命中的“Tony”的 hidden_state 与“Tony”的 embedding 进行 DNN 融合，与 attention_vector 一起作为输入；decode 的输出做的改进：将 lstm 的输出做预测 p_g,将 lstm 的输出与所有 hidden_state 融合再做出预测 p_c,p_g+p_c 则是最终的输出概率。

2、改进的角度-Decode

- **共享参数**

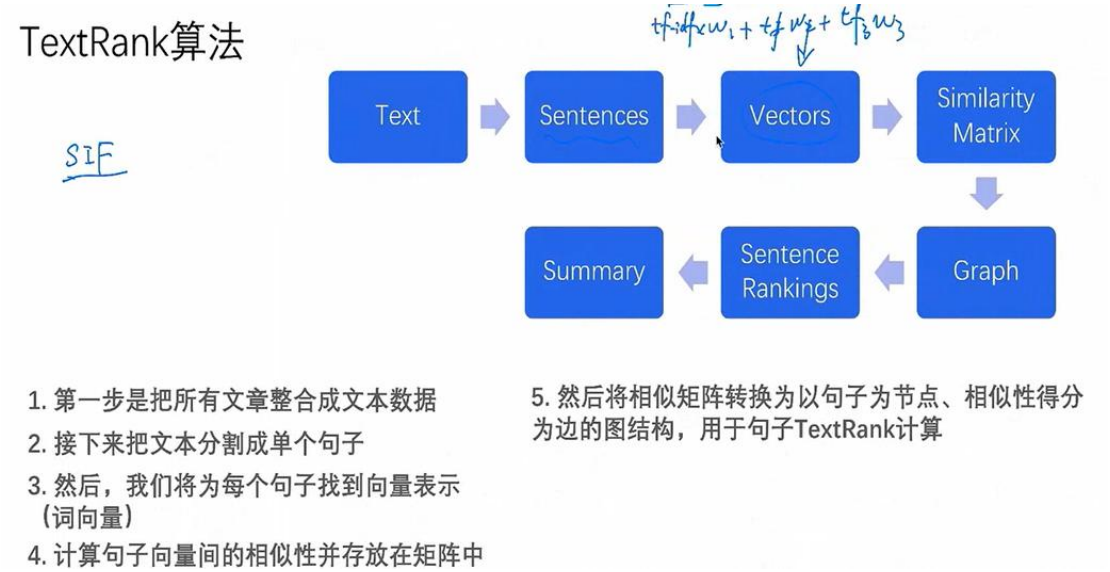
五、文本摘要-抽取式

1、传统方式：

主要是对输入进行文本表征，然后对表征进行打分，排序，取前 k 个进行组合。表征的方式：①主题表征：主题词、主题模型、TF-IDF ②indicator 表征：图结构

(Textrank)、机器学习。

2、TextRank



具体流程：文本--（拆分）-->句子集--->求解句子向量--->采用句向量构建相似矩阵--->矩阵传入 TextRank 训练---->排序---->求前 k 个做摘要。

词向量可以用 glove、word2vec 求得，句子向量该如何求？

答：①句子向量可用 tf-idf 作为权重，将所有词向量进行加权平均（将重要信息给平均了，不合适） ②用 sif 代替 tf-idf 做权重，求句向量；③lstm 模型训练取最后 hidden_state

```
In [25]: def sentence_to_vec(sentence_list, embedding_size, a):
             sentence_set = []
             for sentence in sentence_list:
                 vs = np.zeros(embedding_size)
                 sentence_length = sentence.len()
                 for word in sentence.word_list:
                     a_value = a / (a + get_word_frequency(word.text))  # smooth inverse frequency, SIF
                     vs = np.add(vs, np.multiply(a_value, word.vector)) # vs += sif * word_vector

                 vs = np.divide(vs, sentence_length)  # weighted average
                 sentence_set.append(vs) # add to our existing re-calculated set of sentences

             # calculate PCA of this sentence set
             pca = PCA()
             pca.fit(np.array(sentence_set))
             u = pca.components_[0]  # the PCA vector
             u = np.multiply(u, np.transpose(u))  # u x uT

             if len(u) < embedding_size:
                 for i in range(embedding_size - len(u)):
                     u = np.append(u, 0)  # add needed extension for multiplication below

             # resulting sentence vectors, vs = vs -u x uT x vs
             sentence_vecs = []
             for vs in sentence_set:
                 sub = np.multiply(u,vs)
                 sentence_vecs.append(np.subtract(vs, sub))

             return sentence_vecs
```

Sif 求句向量图

Graph 采用什么方式？

答：TextRank、PageRank。

检索常用的方式？

答：tf-idf--->bm25--->检索