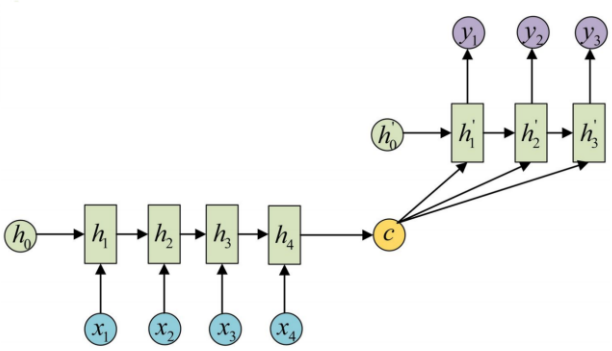
Text Summarization

Review & Interview

一、 文本摘要总结

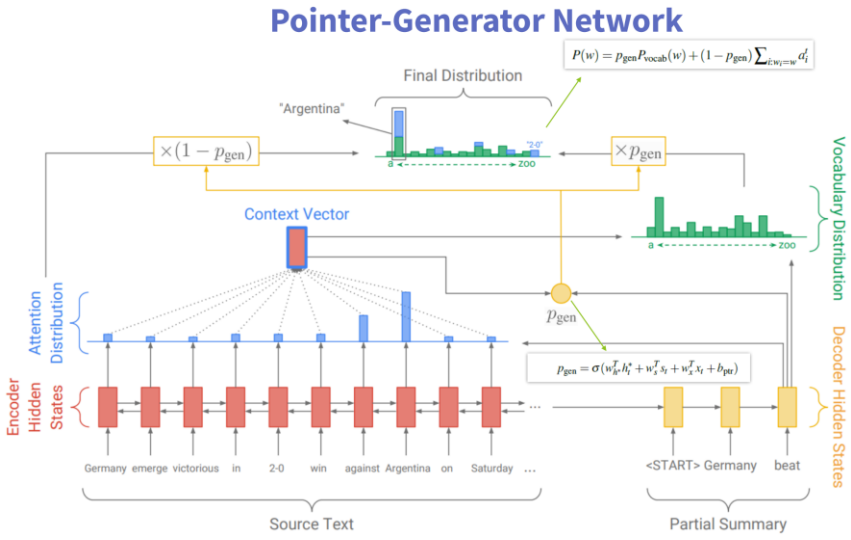
1. 模型

1.1. Seq2seq



Seq2seq 利用 RNN 分别构建 encoder 和 decoder。将原文按照顺序输入 encoder 中，得到一个 encoder 的隐含层变量，在解码的时候，将 decoder 的隐含层变量与 encoder 的隐含层变量进行注意力计算（加性、乘性），与当前 decoder 的输入（teacher forcing 方法）拼接送入 decoder。

1.2. PGN

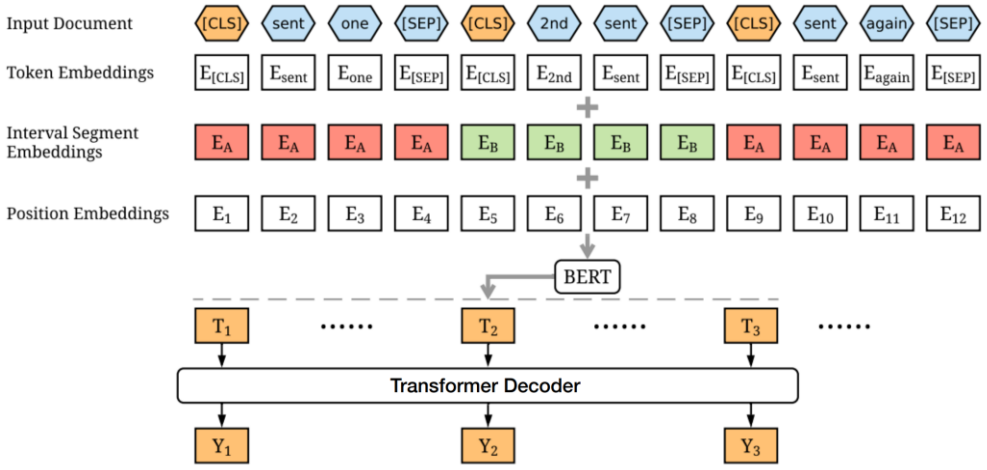


为了解决 OOV 的问题，PGN 使用了 copy-net 的思想，将 encoder-decoder 计算注意力得到的 attention 分布与 decoder 预测的 vocabulary 分布结合，利用 source text 的词替换可能是 OOV 的词。

并且 PGN 使用了 Coverage Mechanism，在损失函数中，将重复关注的地方加大惩罚，

达到减少重复的目的。

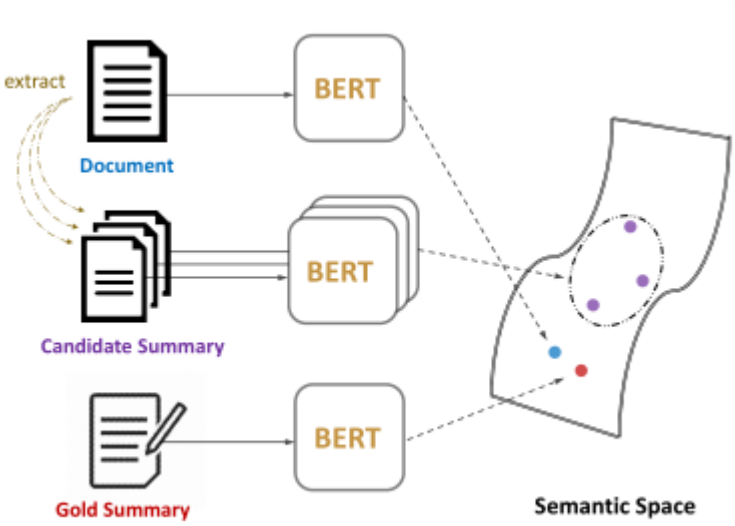
1.3. Bert_sum



Bert for summarization 是基于 Bert 的抽取式摘要模型，它将模型的输入格式进行了修改，原本的 Bert 输入为一个或两个句子，开头添加 CLS，句与句之间添加 SEP，而 Bert for summarizatio 是输入多个句子，每个句子开头添加 CLS，结尾添加 SEP。

将每个 CLS 标记取出，在 Bert 结构之后添加一个分类器，选择哪些 CLS 可以作为摘要抽取出来。这个分类器可以使用简单的一层或两次深度网络，也可以使用复杂一点的 transformer 的 encoder。

1.4. MatchSum



其模型结构与 Triple Siamese 相似，有原文、候选摘要、标准摘要三个输入，其中候选摘要是原文通过 bertsum 选出来得分较高的 m 个句子。通过从候选摘要里选出 n 个句子组成不同的摘要组合，与原文进行相似度计算，选出最佳摘要组合。MatchSum 的 loss 由两部分组成：

第一部分是基于候选摘要与原文档的相似度，其目标函数为：

$$L_1 = \max\left(0, f(D, C) - f(D, C^*) + \gamma_1\right)$$

同样是想让候选组合C与原文D的距离与标准摘要C^*和原文D的距离差小于边界值γ_1。

第二部分考虑候选摘要之间的差异性，即基于 margin loss 的思想，认为得分靠前的与得分靠后的有较大的差异，其损失函数可表示为：

$$L_2 = \max\left(0, f\left(D, C_j\right) - f(D, C_i) + (j - i) * \gamma_2\right), (i < j)$$

这里是对C_k进行了排序，当$f(D,C_j) > f(D,C_i)$，$i,j \in k$时，令$i < j$。这里的思想是，假设$f(D,C_i) = d_1$，那么排在后面的$f(D,C_j)$*则至少大于*$d_1 + (j-i) * \gamma_2$.，以此来拉开C_i*和*C_j的距离。

2. 解码策略

2.1. Greedy Decoding

每次选择当前解码出的最大概率的字当作下一次的输入。计算量小，但是生成的句子可能不是最好的那一个，也有可能陷入循环。

2.2. Beam Decoding

Greedy Decoding 的改进，每次解码选择当前概率与历史概率加和最大的 K 个字，用这 K 个字当作下一次的输入，当获得<STOP>时视为得到一个 hypothesis。解码停止的条件可以是事前定义好的时间步 T 或者是已获得 hypothesis 的数量 n。

2.3. Top-k sampling

由于 Beam Decoding 总是选择概率最大的几个词进行生成，与人类说话的情况不同。

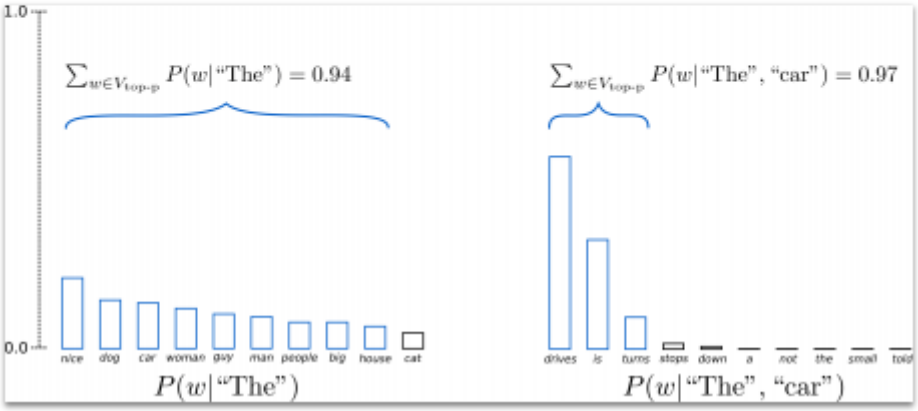
因此引入随机取样。Top-k sampling 选择每个时间步中概率最大的 K 个词，从中进行随机选取。

2.4. Top-p(Nucleus) Sampling

由于 Top-k sampling 的采样在预测分布比较平缓时丢弃概率较大的词，或者比较集中时保留概率较小的词，

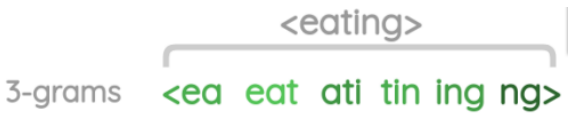
因此 Top-p(Nucleus) Sampling 设定了一个阈值 p，将解码概率从大到小排列，选择相加值大于 p 的前几个词作为采样对象。

$$\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p.$$



3. OOV

3.1. Sub-word generation



根据 n-gram 将单词循环拆分成子词，由于这样会生成很多子词，因此加上一个哈希函数，将 n-gram 子词映射到一个 1 到 B 之间的整数。

3.2. Byte Pair Encoding(BPE)

基于词频的方法，确定期望的 subword 词表大小，将所有单词拆分为字符序列并在末尾添加后缀“\w”，每次循环寻找出现频率最大的一个连续字节对，将其合并为一个新的 subword，直至达到设定的 subword 词表大小或下一个最高 频的字节对出现频率为 1。

3.3. WordPiece

wordPiece 和 BPE 类似，确定期望的 subword 词表大小，将所有单词拆分为字符序列并在末尾添加后缀“\w”，但是 WordPiece 需要在训练集上训练一个语言模型，每次挑选能最大化减少 loss 的 subword，直至达到设定的 subword 词表大小或概率增量低于某一阈值。

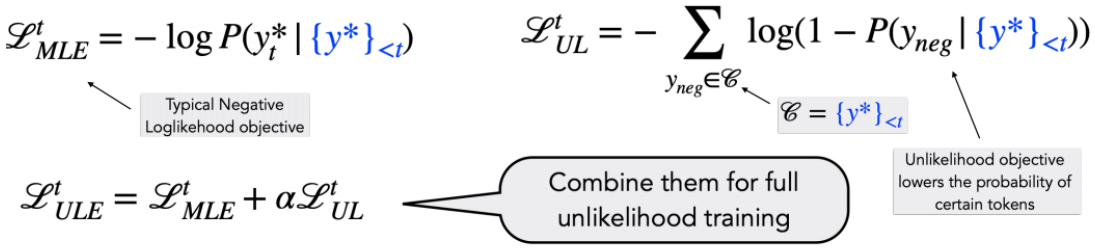
3.4. Unigram Language Model

该模型需根据给定词序列优化下一个 subword 出现的概率，计算每个 subword 的损失，基于损失对 subword 排序并保留前 X%。为了避免 OOV，建议保留字符级的单元。

4. Word-Repetition

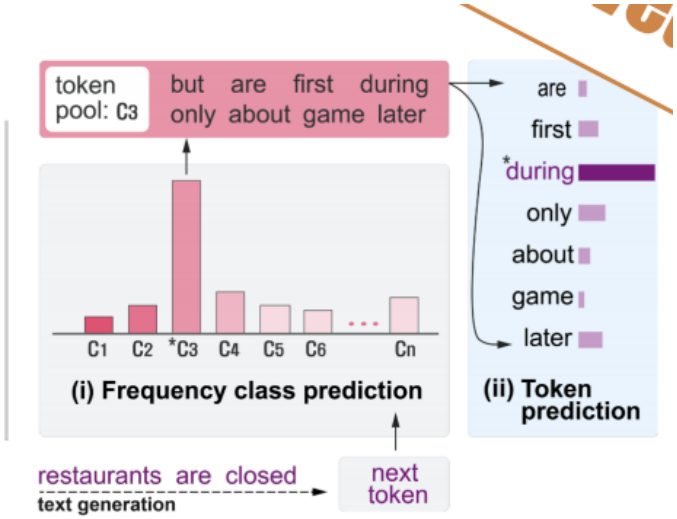
4.1. Unlikelihood Training

为了解决生成语句重复的问题，在极大似然估计的基础设添加了惩罚项，y_neg 是已经生成过的词。



4.2. F2 Softmax

将 softmax 进行了拆分，先根据词频划分不同的类别，再在每个类别中划分具体的单词。



原来的概率计算公式由 $P(y_t = w_n | \{y\}_{<t}) = \frac{e^{U_n h}}{\sum_{m=1}^{M} e^{U_m h}}$ 变为

$$P(y_t = w_n | \{y\}_{<t}) = \left(\frac{e^{V_f h}}{\sum_{c=1}^{C} e^{V_c h}} \right) \left(\frac{e^{U_n h}}{\sum_{m=1}^{M_f} e^{U_m h}} \right)$$

两个部分。

二、 训练技巧

1. Tensorflow pipeline

Tensorflow 提供了现成的结构进行数据的载入的优化操作。

```
import tensorflow as tf

def preprocess(record):
  ...

dataset = tf.data.Dataset.list_files(".../*.tfrecord")
dataset = dataset.interleave(TFRecordDataset, num_parallel_calls=Z)
dataset = dataset.map(preprocess, num_parallel_calls=Y)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=X)

model = ...
model.fit(dataset, epochs=10)
```

tf.data.experimental.AUTOTUNE

2. Before Training

2.1. Before Training Check

1.loss:在用很小的随机数初始化神经网络后，第一遍计算 loss 可以做一次检查(当然要记得把正则化系数设为 0)。

2.接着把正则化系数设为正常的小值，加回正则化项，这时候再算损失/loss，应该比刚才要大一些。

3.试着去拟合一个小的数据集。最后一步，也是很重要的一步，在对大数据集做训练之

前，先训练一个小的数据集，然后看看你的神经网络能够做到 0 损失/loss (当然，是指的正则化系数为 0 的情况下)，因为如果神经网络实现是正确的，在无正则化项的情况下，完全能够过拟合这一小部分的数据

2.2. Parameter Tuning

神经网络的训练过程中，不可避免地要和很多超参数打交道，需要手动设定，大致包括：

1.初始学习率 2.学习率衰减程度 3.正则化系数/强度(包括 l2 正则化强度，dropout比例)

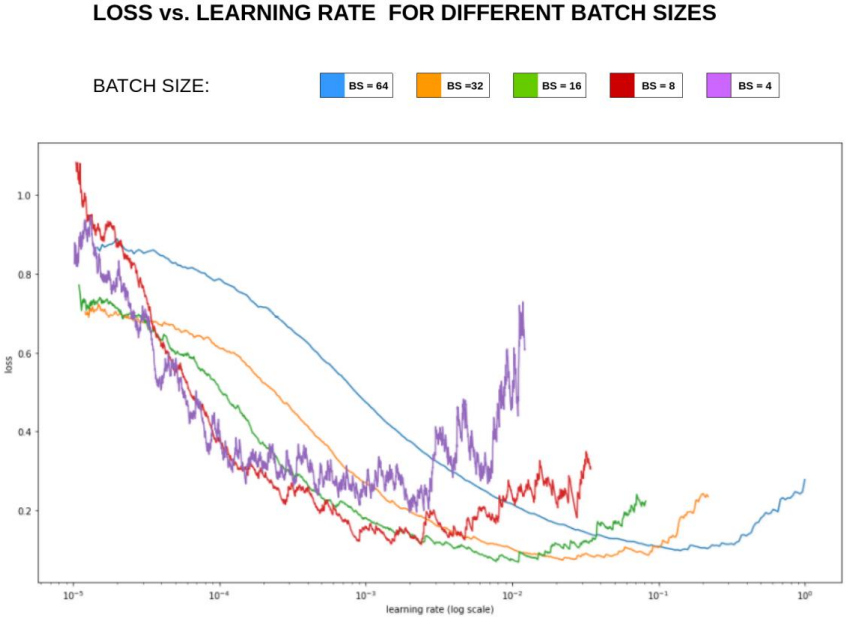
对于大的深层次神经网络而言，我们需要花一些时间去做超参数搜索，以确定最佳设定。最直接的方式就是在框架实现的过程中，设计一个会持续变换超参数实施优化，并记录每个超参数下每一轮完整训练迭代下的验证集状态和效果。实际工程中，神经网络里确定这些超参数，我们一般很少使用n 折交叉验证，一般使用一份固定的交叉验证集就可以了。

一般对超参数的尝试和搜索都是在 log 域进行的。例如，一个典型的学习率搜索序列就是 learning_rate = 10 ** uniform(-6, 1)。另外还得注意一点，如果交叉验证取得的最佳超参数结果在分布边缘，要特别注意，也许取的均匀分布范围本身就是不合理的，也许扩充一下这个搜索范围会有更好的参数

2.3. Find Proper Learning Rate

一般来说，越大的 batch-size 使用越大的学习率。

原理很简单，越大的 batch-size 意味着我们学习的时候，收敛方向的 confidence 越大，我们前进的方向更加坚定，而小的 batch-size 则显得比较杂乱，毫无规律性，因为相比批次大的时候，批次小的情况下无法照顾到更多的情况，所以需要小的学习率来保证不至于出错。



3. Training

3.1. Loss not Decrease

1）数据的输入是否正常，data 和 label 是否一致。

2) 网络架构的选择，一般是越深越好，但是也分数据集。并且用不用在大数据集上 pre-train 的参数也很重要的。

3）loss function 对不对

4. After Training

4.1. Hard Negative Mining

在任何一个深度学习任务中，我们都会遇到一些比较“棘手”的数据，这些数据相比较于其他的普通数据更难识别，这种特比容易识别错误的例子就称为 hard-negative。

方法：我们可以先用初始的正负样本(一般是正样本+与正样本同规模的负样本的一个子集)训练分类器，然后再用训练出的分类器对样本进行分类，把其中负样本中错误分类的那些样本(hard negative)放入负样本集合，再继续训练分类器，如此反复，直到达到停止条件(比如分类器性能不再提升)。也就是不停的将困难样本拿去训练，让分类器更好地学习到难以学习的特征，简单来说就是熟能生巧。