A Gentle Summary of NLG by FuYanjie

总的来讲，自然语言生成的方法有：基于规则的自然语言生成、基于统计的自然语言生成和基于深度学习的自然语言生成方法。受篇幅限制，本文将主要聚焦基于深度学习的自然语言生成方法。

注：本文所涉及的代码实现开源在了 GitHub 仓库中 (https://github.com/FYJNEVERFOLLOWS/hw_NLP)。

0. 基于统计的自然语言生成

0.1. 基于统计的机器翻译

基于统计的机器翻译的核心思想是从数据中学习一个概率模型用于翻译任务。以把中文句子翻译成英文句子为例，对给定的中文句子x，基于统计的机器翻译模型会找出极大似然概率最大的英文句子y，即 $y = argmax_y P(y|x)$。所以核心在于使用基于统计的机器翻译模型学习这个概率分布 P。

可以利用贝叶斯公式将上式进一步化成 $y = argmax_y P(y|x) = argmax_y P(x|y)P(y)$ 。第一部分的 $P(x|y)$ 可以根据翻译模型得到（翻译模型从大量平行语料学习如何翻译单词和短语），第二部分 $P(y)$ 则是语言模型，即可以预测下一个词的模型，用于生成优质自然的目标语言文本。

解码时，基于统计的机器翻译需要通过某种启发式搜索算法寻找最佳翻译 y 使得翻译模型和语言模型的概率乘积最大。

基于统计的机器翻译需要大量的人工特征工程，并且需要分别优化两个子模型，而基于神经网络的自然语言生成方法可以被端到端地优化，这使得设计模型时更加方便、简洁。因此，基于神经网络的方法在越来越多的自然语言生成任务上取代了基于统计的方法。

1. 基于循环神经网络的自然语言生成

与没有序列设定的网络不同，RNN具有运用来自之前序列计算的知识（“记忆”）来处理长序列的能力。

RNN 可以应用在自然语言生成领域的诸多任务上，如机器翻译、摘要提取、文本生成、对话系统等。

1.1. 基于循环神经网络的机器翻译

这里以机器翻译为例[0]，介绍训练用于机器翻译的 RNN 时需要注意如下事项：

1、为编码器和解码器分别训练一个权重不同的 RNN；

2、解码时，根据之前的隐状态计算得到每个时间步的隐状态；

3、为了避免模型过拟合，需要增加模型参数，训练多层的 RNN；

4、对于编码器，因为可以看到序列的全局信息，可以采用双向 RNN。而对于解码器，则只能采用单向 RNN；

5、训练时，将输入序列逆序输入，这样更有利于优化。因为这样能使输入和输出序列中匹配的单词对之间的距离总和最小，因此，可以减轻梯度消失问题[1]。

1.2. 基于循环神经网络的文本生成

这里以再举一个用 RNN 做文本生成的例子：

RNN 可以根据输入的序列预测出一个字符，并将该字符拼在输入序列的后面组成新的序列，不断使用 RNN 预测输入序列的下一个字符，就可以通过一句话得到相同语言风格的一段话甚至是一篇文章了。

预测具体步骤：

1、将输入的文本分割成以字符为单位的序列，并对每个字符进行独热编码得到它对应的 one-hot encoding 向量；

2、把这些 one-hot vector 按时间顺序依次输入 RNN，RNN 的状态向量 **h** 会积累看到的信息；

3、将 权重矩阵 **W** 与 RNN 隐藏层的状态向量 **h** 点乘后的结果输入 Softmax 分类器，输出的到一个元素总和为1的向量，也就是输出了一个字符概率分布；

4、从 Softmax 输出的概率分布中取出概率最大的字符拼接到输入序列的后面，得到新的序列作为下一次的输入。

重复1-4步，直到输出的文本达到指定长度。

训练具体步骤：

将数据集中的长文本分割成一定长度的片段，将片段之前的若干字符作为特征 x，将片段的最后一个字符作为标签 y。找一个字典，根据该字典对特征 x 和标签 y 分别进行独热编码，选择交叉熵损失函数来对训练数据进行训练即可。

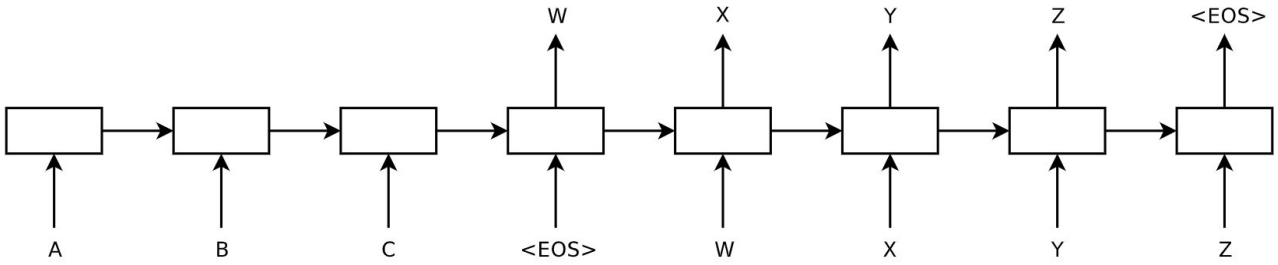
2. 基于序列到序列模型的自然语言生成

seq2seq（序列到序列）模型通过编码器-解码器结构实现，解决了输入和输出序列长度不一致的问题[2]。

编码器是一个没有输出的循环神经网络（RNN），它对读取的输入序列进行编码，并将最后时间步的隐状态传递给解码器作为解码器的初始隐状态，可以是双向。

解码器是另一个循环神经网络（RNN），与编码器相比，它的区别在于解码时要做预测，不能是双向。

常用于机器翻译、语音识别、语音合成等任务。



上图从左至右依次展示了模型在每一个时间步的状态，seq2seq模型先后读取输入序列里的A, B, C，在读到""预测得到输出序列的"W"，并将"W"作为下一个时间步模型预测的输入，直到模型预测得到""，解码结束。

2.1. 基于序列到序列模型的机器翻译

训练过程：

1、对输入文本和目标文本分别进行分词，然后建立两种不同的词典，需要注意的是，要向目标语言的词典中加入起始符和终止符；

2、根据字符在相应字典中的索引分别得到表示输入文本和目标文本的索引序列，并将输入序列和目标序列进行独热编码；

3、用 LSTM encoder 提取输入序列的特征，LSTM encoder 输出最后一个隐藏状态 h 和 细胞状态 c，并作为 LSTM decoder 的初始状态；

4、LSTM decoder 每次接收一个输入序列，输出对下一个字符预测的概率分布 p，将下一个字符的真实值进行独热编码，并计算标签独热向量与预测的概率分布 p 之间的交叉熵损失函数，根据 loss 做反向传播，更新 LSTM encoder 和 LSTM decoder 的模型参数。

重复以上训练过程，直至模型收敛。

推理过程：

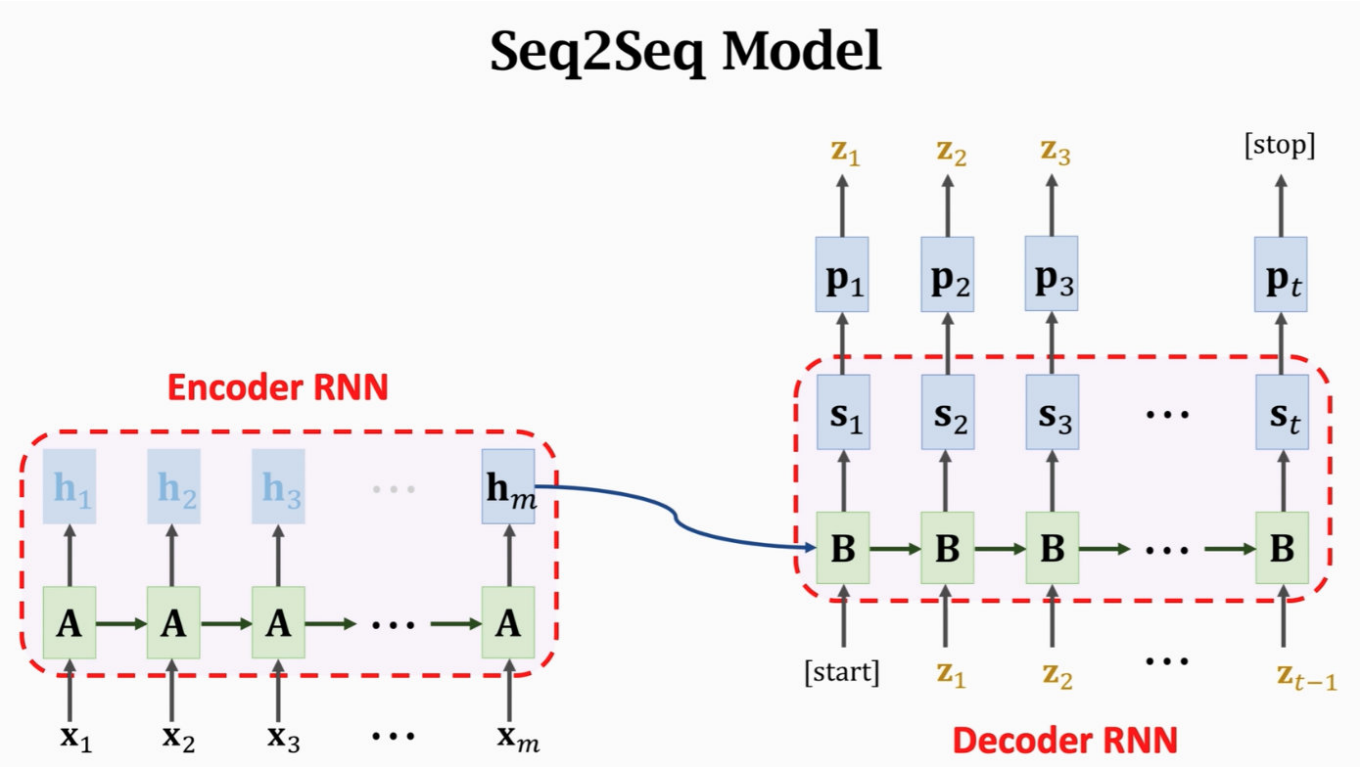
1、把输入文本对应的输入序列送入 LSTM encoder，LSTM encoder 会在状态 h 和 c 中积累输入文本的信息，LSTM encoder 输出的最后状态被当作从输入文本中提取的特征 h 和 c，并作为 LSTM decoder 的初始状态；

2、LSTM decoder 接收起始符，预测下一个字符并更新状态至 h1 和 c1；

3、把预测的字符作为 LSTM decoder 的输入，去预测下一个字符并基于状态 h1 和 c1 更新状态至 h2 和 c2；

重复步骤 3，直至预测得到终止符，将每一个时间步预测得到的字符拼接成的字符串返回。

整个序列到序列模型的机器翻译过程如下：



2.2. 基于序列到序列模型的语音识别

这里以语音识别中的 LAS 模型[3]为例，阐述该方法的一般步骤。

1、对输入声音信号进行特征提取，将提取到的特征 [声音波形或声音语谱图或梅尔频率倒谱系数 (MFCC)或三角窗滤波器组的输出 (filter bank)] 作为输入序列；

2、通过基于双向长短时记忆网络的编码器 listener 将低层 (low-level) 的语音信号特征序列转化成长度更短的高层 (high-level) 的语音信号特征序列 **h**；

3、通过基于注意力机制的解码器 speller 从高层的语音信号特征序列 **h** 生成字母序列 **y**。

可以看到，seq2seq中的解码器RNN在解码时是需要每次得到前一个时间步的输出才能得到当前时间步的输入并进行预测，所以是一个自回归模型，只能串行预测，预测时间与输出序列长度成正比。

为了解决上述问题，Attention is all you need[4] 提出基于transformer的编码器和解码器，使得一次性解码得到输出序列成为可能，大大减少了预测所花费的时间。

3. 基于注意力机制模型的自然语言生成

有了注意力机制，seq2seq 模型的 decoder 在更新状态时会看一遍 encoder 的所有状态，这样就不会遗忘；而且注意力机制可以告诉 decoder 应该关注 encoder 的哪个状态[4]。不过，相比 seq2seq 模型，基于注意力机制的模型计算量很大。

3.1. 基于注意力机制的模型结构介绍

相比基于RNN的序列到序列模型，基于注意力机制的模型还需要计算 context vector c_j，下面介绍 Attention 层如何工作：

encoder 的输入是 $x_1, x_2, \cdots, x_m$.

decoder 的输入是 $x'_1, x'_2, \cdots, x'_t$.

(a) 基于 encoder 的输入计算得到 Keys 和 Values

Key: $k_{:i} = W_K x_i$

Value: $v_{:i} = W_V x_i$

(b) 基于 decoder 的输入计算得到 Queries

Query: $q_{:j} = W_Q x'_j$

(c) 计算权重 $\alpha_{:1} = Softmax(K^T q_{:1}), K = [k_{:1}, k_{:2}, \cdots, k_{:m}]$

(d) 计算 Context vector: $c_{:1} = \alpha_{11} v_{:1} + \cdots + \alpha_{m1} v_{:m} = V\alpha_{:1}, V = [v_{:1}, v_{:2}, \cdots, v_{:m}]$

对 decoder 的每一个输入 x'_j 重复(c)(d)步骤 t 次

Attention层最终的输出是 $C = [c_{:1}, c_{:2}, \cdots, c_{:t}]$

3.2. 基于注意力机制模型的机器翻译方法

下面介绍基于注意力机制模型的机器翻译方法[5]：

一、编码器结构

编码器会把输入的序列转换成 hidden states 作为后续的 context vector

embedding层：nn.Embedding(vocab_size, embed_size)

rnn层：nn.GRU(embed_size, enc_hidden_size, batch_first=True, bidirectional=True)

fc层：nn.Linear(enc_hidden_size * 2, dec_hidden_size)

二、注意力层结构

根据 context vector 和当前 hidden states 计算输出

输入层：nn.Linear(enc_hidden_size * 2, dec_hidden_size, bias=False)

输出层：nn.Linear(enc_hidden_size*2 + dec_hidden_size, dec_hidden_size)

前向传播过程：参数 context 经过输入层与参数 output 做矩阵乘法并进行 Softmax 得到 attention 分数。将得到的 attention 分数与 context 做矩阵乘法更新 context，将 output 与 context 拼接更新 output，最后返回 output 和 attention 权重。

三、解码器结构

解码器根据已经输出的序列内容和 context vector 决定下一个输出的单词

embedding层：nn.Embedding(vocab_size, embed_size)

attention层：Attention(enc_hidden_size, dec_hidden_size)

rnn层：nn.GRU(embed_size, enc_hidden_size, batch_first=True)

fc层：nn.Linear(dec_hidden_size, vocab_size)

四、构建序列到序列模型将编码器、注意力层和解码器串到一起

前向传播过程：输入序列经过编码器得到 encoder 的输出 encoder_out 和隐层状态 hid，并分别 作为 context 和 hid 参数传入解码器进行前向传播，得到输出 output 和注意力权重。

4. 基于预训练模型的自然语言生成

得益于算力和存储的显著提升，近年来出现了越来越多的预训练模型 (Pre-trained Models，PTMs)，例如 BERT, GPT, ELMo, BART。它们通常被用于提取通用语言表征或语言表示，作为下游任务的输入。

4.1. 基于预训练模型的对话系统

以GPT (Generative Pre-Training) 为代表的解码器 (transformer decoder) 模型可以被使用在文本生成、机器翻译、对话系统等自然语言生成任务中。

下面简要总结了基于HuggingFace的transformers实现的中文聊天机器人 (chatbot) 的训练步骤。

一、训练数据预处理

1、初始化 tokenizer（分词器）并进行 tokenize；

2、将所有分词处理后的对话数据按照"[CLS]utterance1[SEP]utterance2[SEP]utterance3[SEP]"的格式进行保存；

3、输入预处理后的训练样本，对加载的 GPT-2 预训练模型进行自回归训练；

4、使用训练好的模型与聊天机器人聊天，以评估其性能。

测试样例：

Sample1

```
(gpt2_chitchat) fuyanjie@fuyanjiedeMacBook-Pro GPT2-chitchat % python interact.py --no_cuda --model_path ./model/model_epoch40_50w --max_history_len 3
2021-12-07 10:58:19,156 - INFO - using device:cpu
开始和chatbot聊天，输入CTRL + Z以退出
user:早上好
chatbot:早上好呀
user:你在哪里
chatbot:在公司
user:上班呢？
chatbot:我也不知道，路过而已
user:一会去哪吃？
chatbot:在陕西安市，吃了家乡的豆乳鸡，
user:西安有什么好吃的？
chatbot:没啥好吃的，西安豆乳鸡，小吃，牛肉鸡，鸭肉鸡，鸭肉
user:什么叫牛肉鸡，鸭肉鸡
chatbot:就是那个什么牛肉鸡，有时候我也不知道是什么
user:你不够智能
chatbot:我智商低
user:确实
chatbot:我知道你智商低
user:我 创造了你
chatbot:哈哈哈哈哈
```

Sample2

```
(gpt2_chitchat) fuyanjie@fuyanjiedeMacBook-Pro GPT2-chitchat % python interact.py --no_cuda --model_path ./model/model_epoch40_50w --max_history_len 3
2021-12-07 11:04:28,602 - INFO - using device:cpu
开始和chatbot聊天，输入CTRL + Z以退出
user:最近在忙什么
chatbot:忙着读书啊
user:中午一起吃吧
chatbot:吃过饭了
user:这还不到11点就吃了？
chatbot:早晨吃
user:好吧
chatbot:下午吃多啦，现在吃太多
user:那下次一起
chatbot:好啊好啊，晚上吃白菜
user:晚上 带你吃日料
chatbot:没问题。
```

Sample3

```
(gpt2_chitchat) fuyanjie@fuyanjiedeMacBook-Pro GPT2-chitchat % python interact.py --no_cuda --model_path ./model/model_epoch40_50w --max_history_len 3
2021-12-07 11:08:31,747 - INFO - using device:cpu
开始和chatbot聊天，输入CTRL + Z以退出
user:周末去北京吧
chatbot:可是周末我不出去啊
user:为什么啊
chatbot:因为周末人多
user:人多才好玩啊
chatbot:就是没什么好玩的
user:那去个人少的地方
chatbot:那不是去玩的
user:那你说去哪
chatbot:你心里
user:无语
chatbot:我不
user:或者去南京呢？也好玩
chatbot:没人一起去
user:我和你一起
chatbot:我不信
user:这也不信
chatbot:这不是愚人节闹心么
user:今天不是愚人节啊
chatbot:可是我今天是了
user:你的逻辑不对
chatbot:我是可爱的小宝贝
```

References

[0] NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

[1] https://www.bilibili.com/video/BV1Vb4y187XP?p=7

[2] Sequence to Sequence Learning with Neural Networks

[3] Listen, Attend and Spell

[4] Attention Is All You Need

[5] Effective Approaches to Attention-based Neural Machine Translation

[6] https://andongluis.github.io/nlg-blog/