机器学习与深度学习面试系列十八（Seq2Seq）

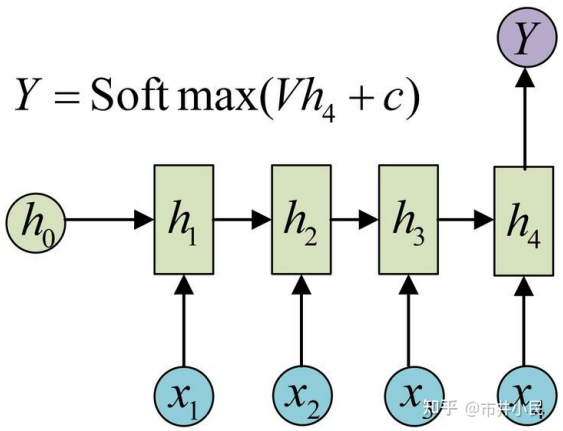
什么是Seq2Seq模型?

不严谨的说，Seq2Seq模型是实现将一个序列映射到另一个序列的模型，这样的场景在机器学习中有很多。[1]例如：

- 将一个序列映射到一个值，这种我们在下文中成为N对1的Seq2Seq模型。这样的模型比较适合用于情感分析、文本分类的任务。
- 将一个值映射到一个序列，这种我们在下文中成为1对N的Seq2Seq模型。这样的模型比较适合用于从图像生成文字（Image caption）或者从类别生成语音或音乐的任务。
- 将长度为N的序列映射到长度为N的序列，这种我们在下文中成为N对N的Seq2Seq模型，又叫同步的Seq2Seq模型，比较适合用于序列标注的任务。
- 将长度为N的序列映射到长度为M的序列，这种我们在下文中成为N对M的Seq2Seq模型，又叫异步的Seq2Seq模型，比较适合用于机器翻译的任务，这也是最广为使用的模型。

值得注意的一点是，这里对于序列的解析可以有多种方法，例如RNN，LSTM、GPU、CNN、注意力机制(Transformer)。RNN是比较简单的模型，下面的说明我们都基于RNN。

N对1的Seq2Seq模型?

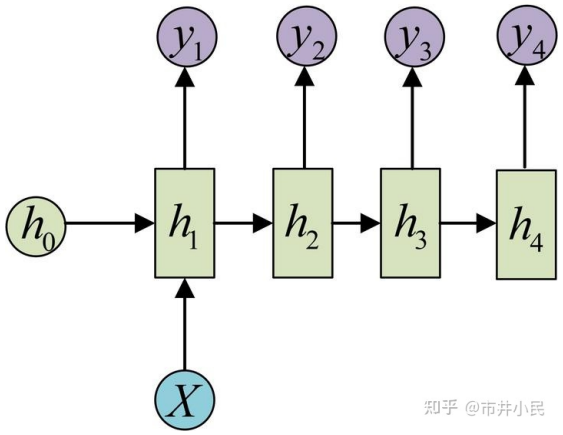


N对1的Seq2Seq模型主要用于序列数据的分类问题:输入为序列，输出为类别。比如在文本分类中，输入数据为单词的序列，输出为该文本的类别。我们可以将样本 $\boldsymbol{x}$ 按不同时刻输入到循环神经网络中，并得到不 同时刻的隐藏状态 $\boldsymbol{h_1, h_2, \ldots, h_T}$. 我们可以将 $\boldsymbol{h_T}$ 看作整个序列的最终表示，通过一个分类器得到最终类别。分类器的选择上，简单的如图中所示的Softmax或者二分类可以直接使用sigmoid，复杂的可以再将 $\boldsymbol{h_T}$ 喂入一个全连接前馈神经网络进行分类。

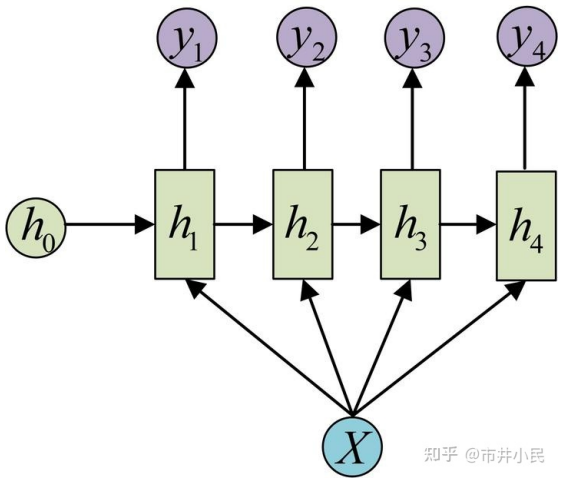
这里我们仅使用最后一个 $\boldsymbol{h_T}$ 作为最后分类的输入，也可以使用 $\boldsymbol{h_1, h_2, \ldots, h_T}$ 的平均值 $\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{h_t}$ 作为最后分类的输入。

1对N的Seq2Seq模型?

最简单的1对N即在第一个时刻，输入 $\boldsymbol{x}$ ，后续每个时刻直接得到输出。

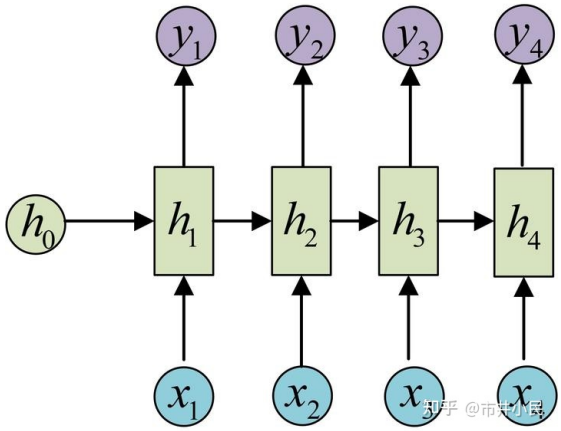


还有一种结构是把输入信息X作为每个阶段的输入：



例如Image caption任务，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。

同步的Seq2Seq模型?



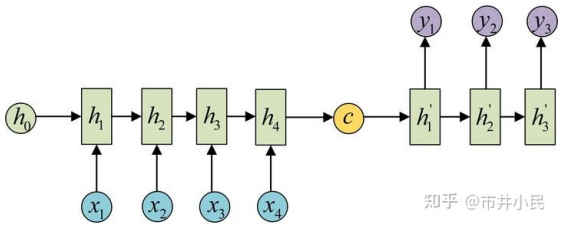
这种结构比较适合用于序列标注(Sequence Labeling)任务，即每一时刻都有输入和输出，输入序列和输出序列的长度相同。比如在词性标注(Part-of-Speech Tagging)中，每一个单词都需要标注其对应的词性标签。

输入为一个长度为 T 的序列 $x_{1:T} = (x_1, x_2, \ldots, x_T)$ ，输出为序列 $y_{1:T} = (y_1, y_2, \ldots, y_T)$ 。样本 x 按不同时刻输入到RNN中，并得到不同时刻的隐状态 $h_1, h_2, \ldots, h_T$ 每个时刻的隐状态 h_T 代表了当前时刻和历史的信息，并输入给分类器得到当前时刻的标签 y_T 。

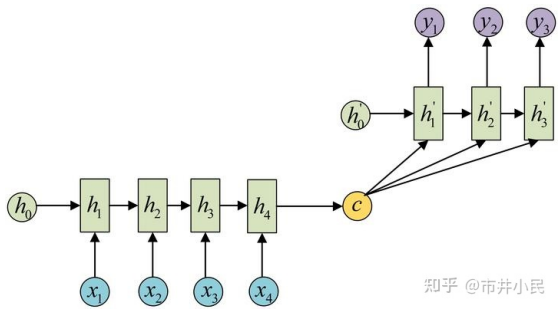
异步的Seq2Seq模型?

这是最广为使用的Seq2Seq模型，通常没有额外说明的Seq2Seq，就是指这个模型，它也称为编码器-解码器(Encoder-Decoder)模型，即输入序列和输出序列不需要有严格的对应关系，也不需要保持相同的长度。比如在机器翻译中，输入为源语言的单词序列，输出为目标语言的单词序列。

它的基本思路首先通过Encoder将输入序列编码为一个上下文向量 c ，然后将 c 交由Decoder解码得到输出序列。一种常见的结构如下图所示：

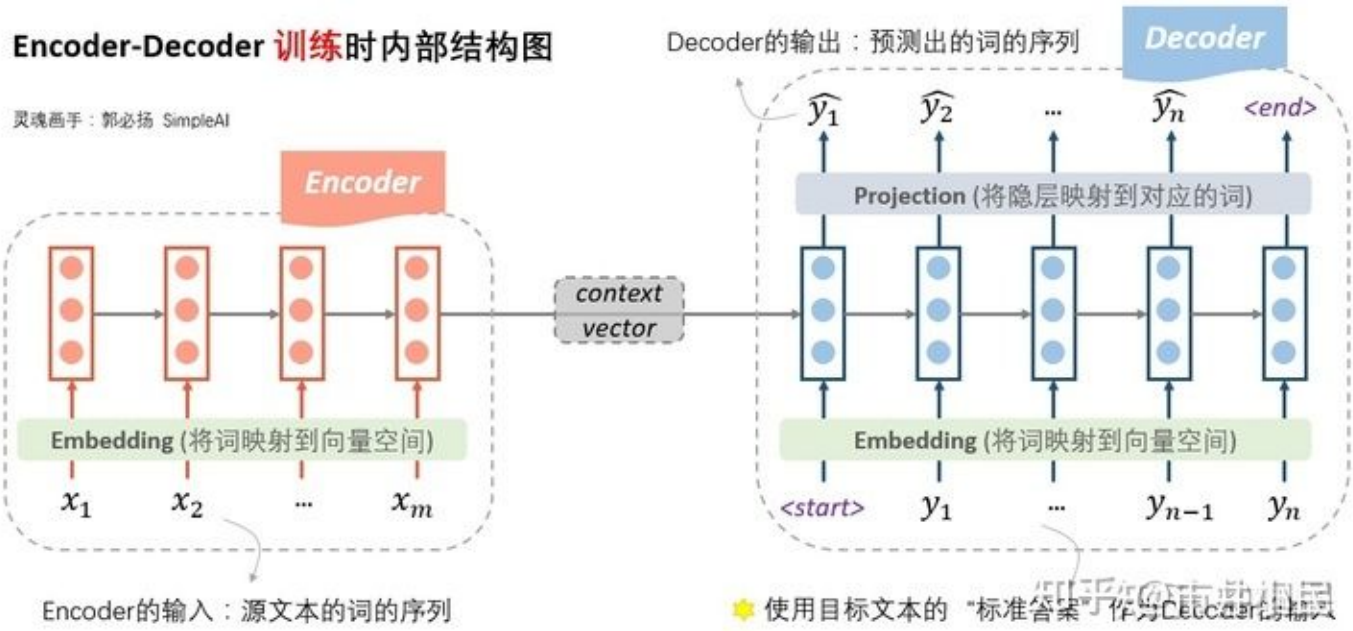


还有一种做法是将c当做Decoder中每一步的输入：

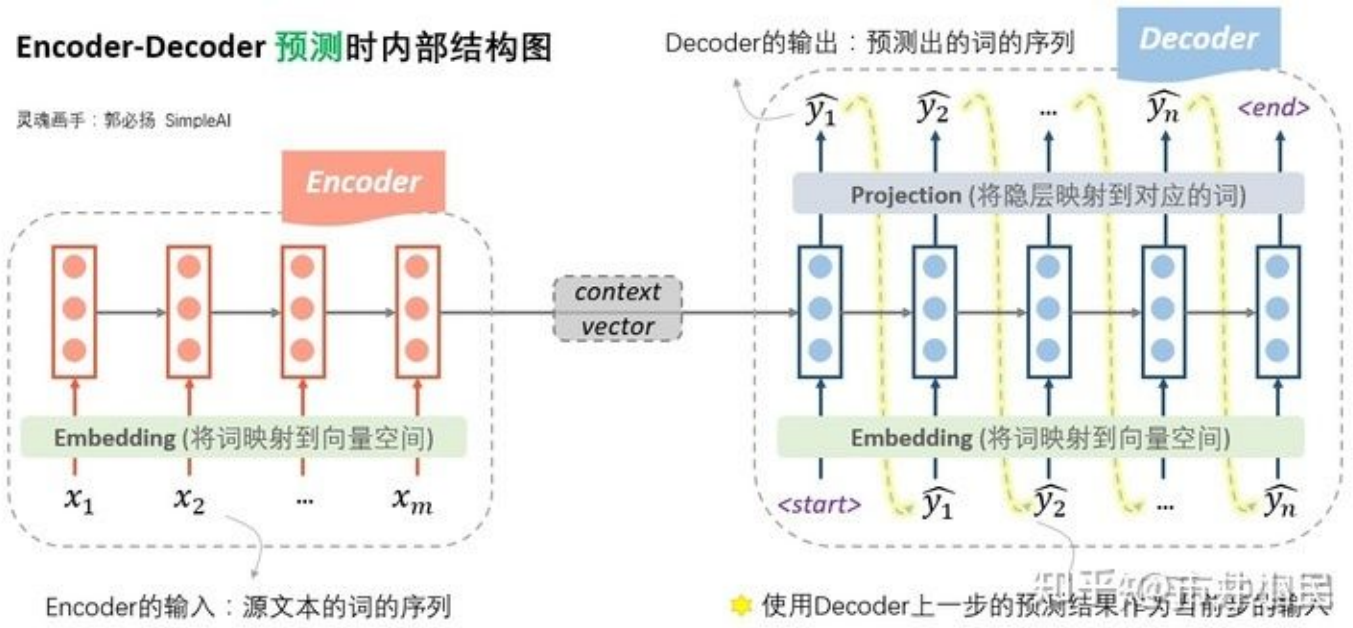


teacher forcing机制是怎样的?

上个问题中我们给出的图示是最简单的Seq2Seq的示意图，其中忽略了很多细节。



这张图，展示了在**训练时**，seq2seq内部的详细结构。在Encoder端，我们将输入文本的词序列先经过embedding层转化成向量，喂进Encoder中，得到一个context vector。Decoder端的输入除了context vector以外，还将接受上一个时刻的输出。这里Decoder在训练和测试时是不一样的。**在训练时，我们使用真实的目标文本，即"标准答案"作为输入**（注意第一步使用一个特殊的 `<start>` 字符，表示句子的开头）。每一步根据当前正确的输出词、上一步的隐状态来预测下一步的输出词。而在预测时，由于Decoder端此时没有所谓的"真实输出"或"标准答案"了，所以只能**自产自销：每一步的预测结果，都送给下一步作为输入**，直至输出 `<end>` 就结束。[2]



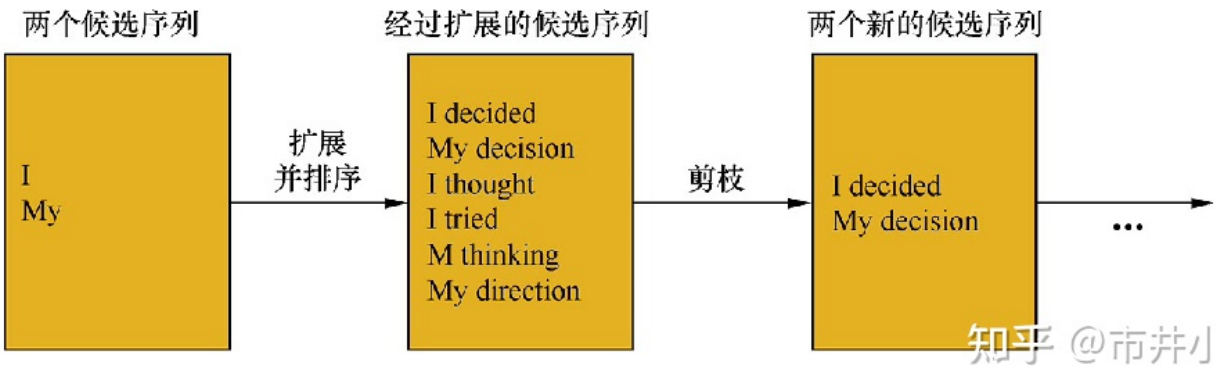
我们称这两种模式，根据标准答案来decode的方式为**teacher forcing**，而根据上一步的输出作为下一步输入的decode方式为**free running**。teacher forcing机制是为了加快训练速度，因为在训练

时如果没有任何的引导，一开始会完全是瞎预测，正所谓“一步错，步步错”，而且越错越离谱，训练起来就很费劲。但是如果在训练时全盘使用teacher forcing，模型难以学习到在出现错误时如何纠正自己，那么在预测时正确率也不会太高，因为如果出现预测错误，后面可能就错的越来越离谱了。更好训练办法是，设置一个概率p，每一步以概率p靠自己上一步的输来decode，以概率1-p根据正确的标准答案来decode。

Beam search是什么?

前面展示的预测过程，其实就是最简单的decoding方式——**Greedy Decoding**，即每一步都预测出概率最大的那个词，然后输入给下一步。这种Greedy的方式，简单快速，但是每一步最优，不一定全局最优。

改进的方法，就是使用**Beam Search**方法，它是一种启发式算法。该方法会保存beam size(后面简写为b)个当前的较佳选择，然后解码时每一步根据保存的选择进行下一步扩展和排序，接着选择前b个进行保存，循环迭代，直到结束时选择最佳的一个作为解码的结果。



由图可见，当前已经有解码得到的第一个词的两个候选:I和My。然后，将I和My输入到解码器，得到一系列候选的序列，诸如I decided、My decision、I thought等。最后，从后续序列中选择最优的两个，作为前两个词的两个候选序列。很显然，如果b取1，那么会退化为前述的贪心法。随着b的增大，其搜索的空 间增大，最终效果会有所提升，但需要的计算量也相应增大。在实际的应用(如 机器翻译、文本摘要)中，b往往会选择一个适中的范围，以8~12为佳 。

Seq2Seq框架在编码-解码过程中是否存在信息丢失?有哪些解决方案?

如果只用固定大小的状态向量来连接Encoder和Decoder，这就要求Encoder将整个输入序列的信息压缩到状态向量中，而这是一个有损压缩过程，序列越长，信息量越大，编码的损失就越大，最终会让Encoder无法记录足够详细的信息，从而导致Decoder翻译失败。

在“Sequence to Sequence Learning with Neural Networks”[3] 这篇论文中，作者提出将待翻译序列的顺序颠倒后 再输入到编码器中，例如原句为“Tom likes fish”，则输入编码器的 句子为“fish likes Tom”。经过这样的处理，编码器得到的状态向量能够较好地关注并保留原句中靠前的单词，

这样在解码时，靠前的单词的识别 / 理解准确率会得到较大提升。翻译过程中序列间的依赖关系使靠前的单词的准确率更为重要，因而这种方法能使模型更好地处理长句子。

另外一种解决方案就是大名鼎鼎的注意力机制，我们知道即便是LSTM，也只是长的“短时记忆”，远达不到长期记忆的目的，这就是导致压缩过程是有损的原因。注意力机制是一种新的提取变量间局部依赖关系的方法，它仅仅关注与当前处理单词最相关的若干个单词，这样极大的解决了远距离依赖RNN和CNN处理不好的场景，正如Google这篇著名论文[4]的标题所说的：Attention is all you need!

参考

1. ^ https://zhuanlan.zhihu.com/p/28054589
2. ^ https://zhuanlan.zhihu.com/p/147310766
3. ^ https://arxiv.org/pdf/1409.3215.pdf
4. ^ https://arxiv.org/pdf/1706.03762.pdf