7月21日

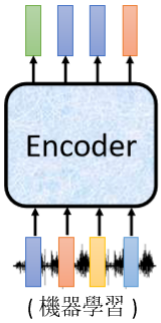
一、李宏毅2021春机器学习课程第5.2节：Transformer（二）

1 解码器 (AT版) 的具体架构

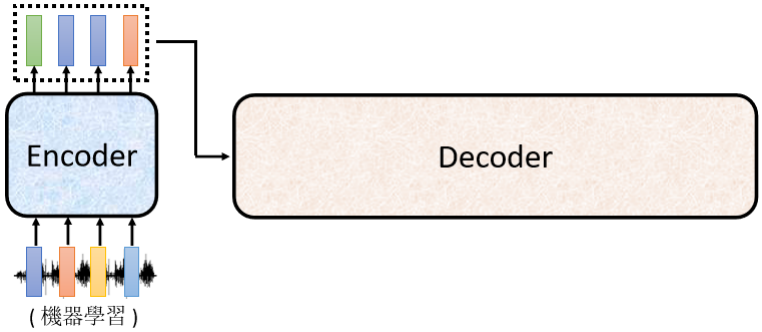
Decoder其实有两种，比较常见的是 Autoregressive Decoder。

> 根据上文内容预测下一个可能跟随的单词，就是常说的自左向右的语言模型任务，或者反过来也行，就是根据下文预测前面的单词，这种类型的LM被称为Autoregressive的语言模型。

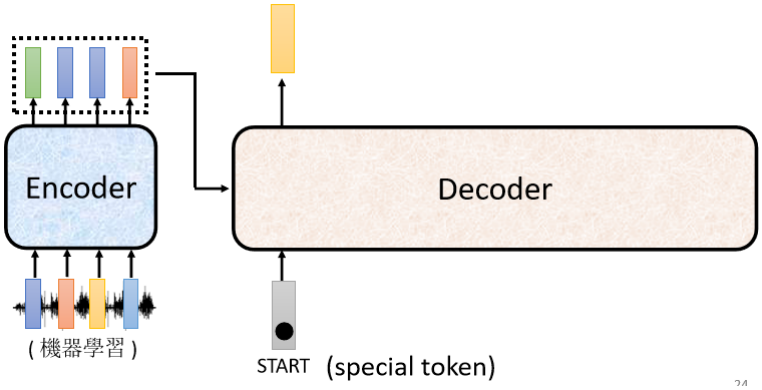
继续举语音辨识的例子来说明Autoregressive Decoder具体是怎么做的，**语音辨识**就是**输入一段声音，输出一串文字**，我们把一段声音输入给 Encoder，表示声音讯号的一排向量进入 Encoder以后，输出也是一排向量。



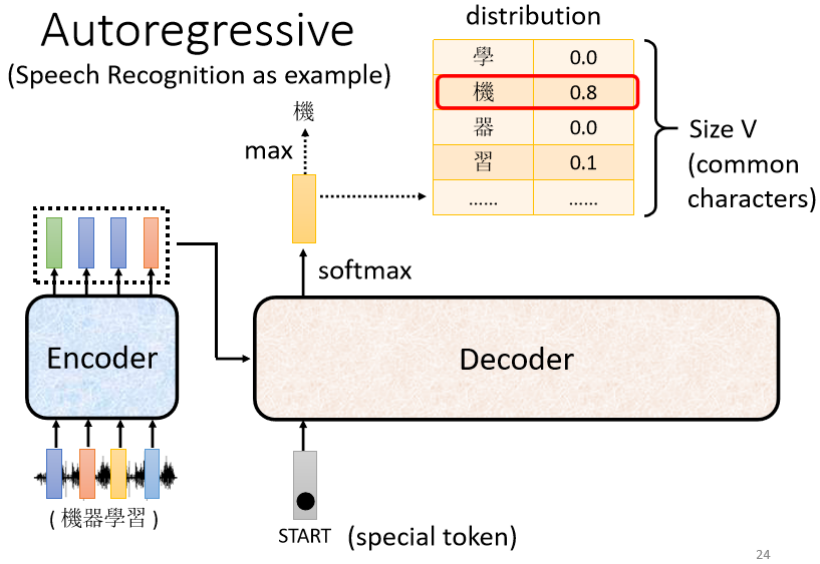
接下来就轮到 Decoder 上场了，**Decoder 要做的事情就是把 Encoder 的输出先读进去，然后产生最终输出**，也就是**产生语音辨识的结果。**



那Decoder怎么产生一段文字呢？首先，你要先给它一个特殊的符号，**这个特殊的符号代表开始**，图中用了**START**来表示，有时候也用**BOS（Begin Of Sentence）**来表示。

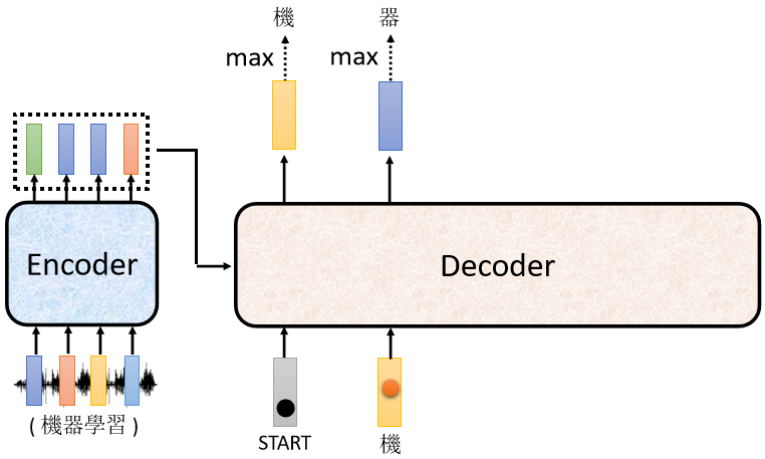


假设我们将最终要输出的**每一个 Token，都用一个 One-Hot 的 Vector 来表示，START 也用 One-Hot Vector 来表示**，那输入 START 后接下来 Decoder 会**先输出一个向量，这个 Vector 的长度跟 Token 的 One-Hot Vector 一样长。**

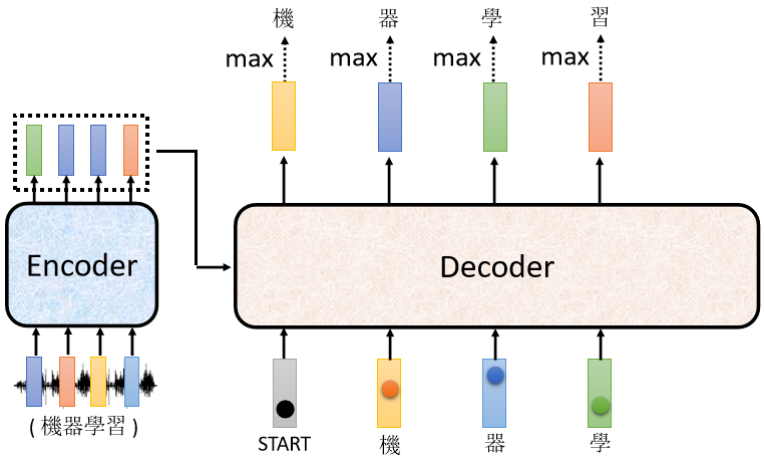


每一个中文的字因为One-Hot编码，其向量表示里面只有一个特定的位置是1，而 Decoder 的输出通过一个 Softmax 层之后，输出向量变为了一个概率分布，**概率最大的位置对应的中文字即为真正的输出**，在这个例子中，“机”的概率最高，所以“机”就是这个 Decoder 的第一个输出。

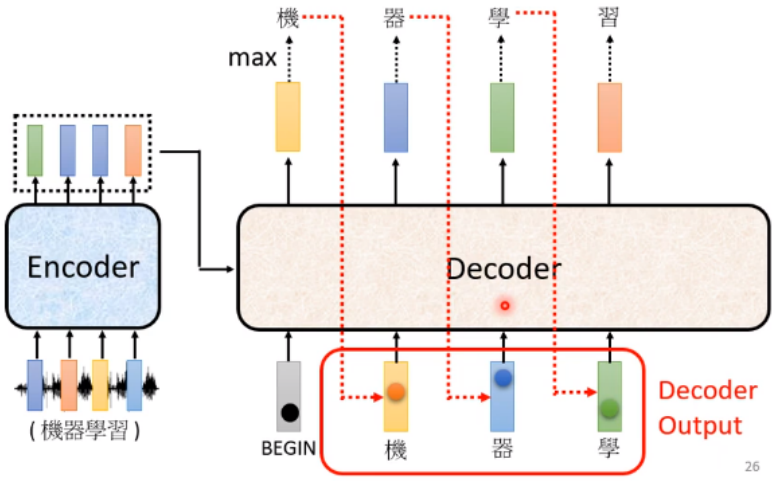
接下来，我们**把“机”当做是 Decoder 新的 Input**，原来 Decoder 的 Input 只有 START 这个特别的符号，现在它除了 START 以外，还有“机”作为它的 Input。



根据这两个输入，它输出一个蓝色的向量，根据这个蓝色的向量里面概率最大的位置对应的中文字即为真正的输出，假设"器"的概率最大，那么**"器"就是输出。**



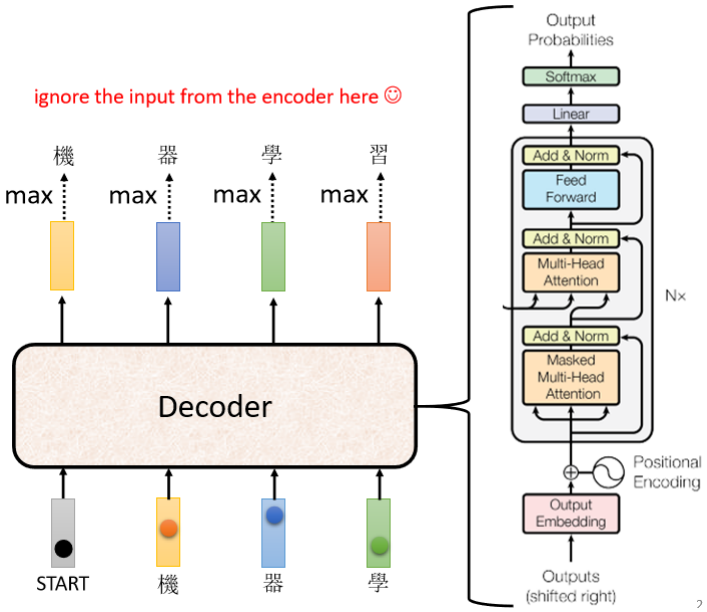
然后这个过程就反复持续下去，这边有一个关键的地方，我们用红色的虚线把它标出来：



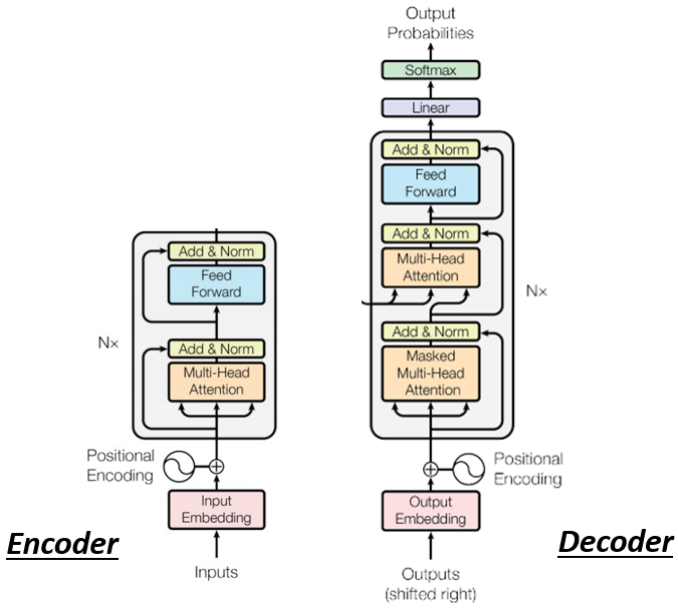
我们看到 Decoder 看到的输入，其实是它在前一个时间点自己的输出，**Decoder 会把自己的输出，当做接下来的输入。**那这时就有一个问题，如果 Decoder 产生了错误的输出，这个错误的输出又被当做 Decoder 下一个时间点的输入，会不会造成 Error Propagation，也就是类似于“一步错，步步错” 的问题呢?

那事实上也的确是有可能的，这个等会儿再继续讨论，这里先把Decoder的具体架构介绍完。

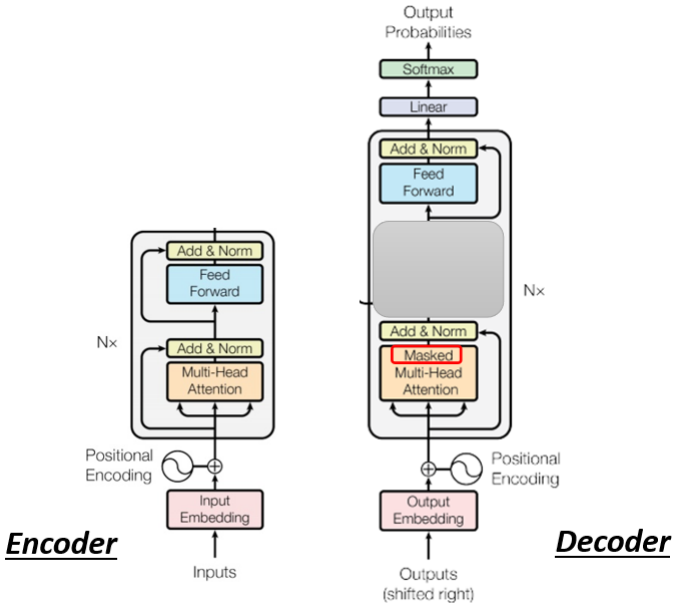
我们来看一下 **Decoder 内部的具体结构**是什么样的：



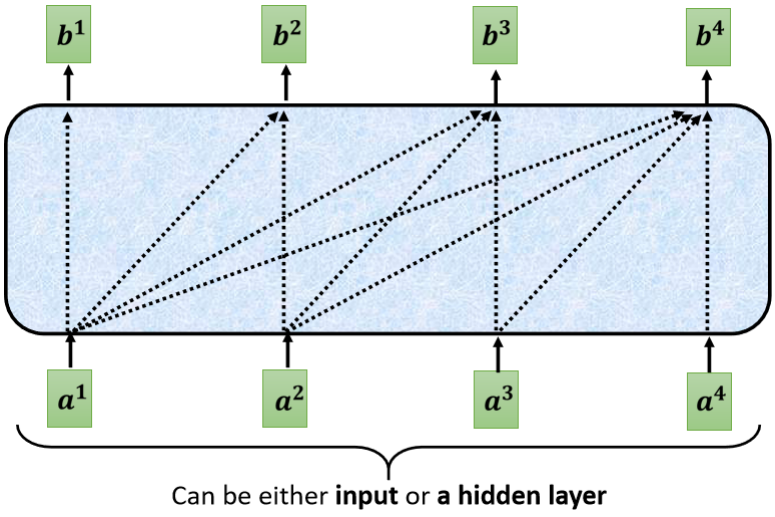
由于上一节刚刚讲过了 Encoder 的架构，我们现在把 Encoder 和 Decoder 放在一起来看：



稍微比较一下它们之间的差异，你会发现**如果我们把 Decoder 中间这一块遮起来的话，其实 Encoder 跟 Decoder 并没有很大的差别。**

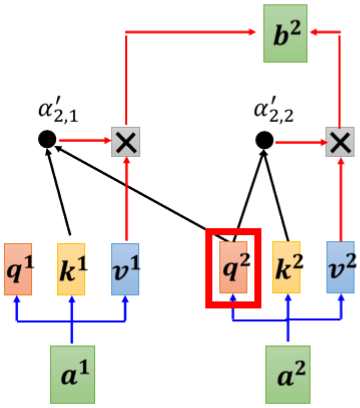


这里我们着重来看一看不一样的地方，**首先是Decoder的Multi-Head Attention上面多了一个修饰词**——masked，所谓的 Masked Attention，**就是我们现在产生输出的时候不能再看右边的部分**，也就是产生 b^1 的时候，我们只能考虑 a^1 的资讯，而不能够再考虑 a^2 a^3 a^4，产生 b^2 的时候，只能考虑 a^1 a^2 的资讯，不能再考虑 a^3 a^4 的资讯，以此类推。



那为什么需要加上 Masked 这个限制呢？这个理由其实也很自然，我们刚刚就提到了**Autoregressive Decoder的输出是顺序产生的**，所以是先有 a^1 再有 a^2,再有 a^3 再有 a^4。**这跟原来的 Self-Attention 不一样**，原来的 Self-Attention，a^1 到 a^4 是一次整个输入 Model 里面的，然后一次全部输出。

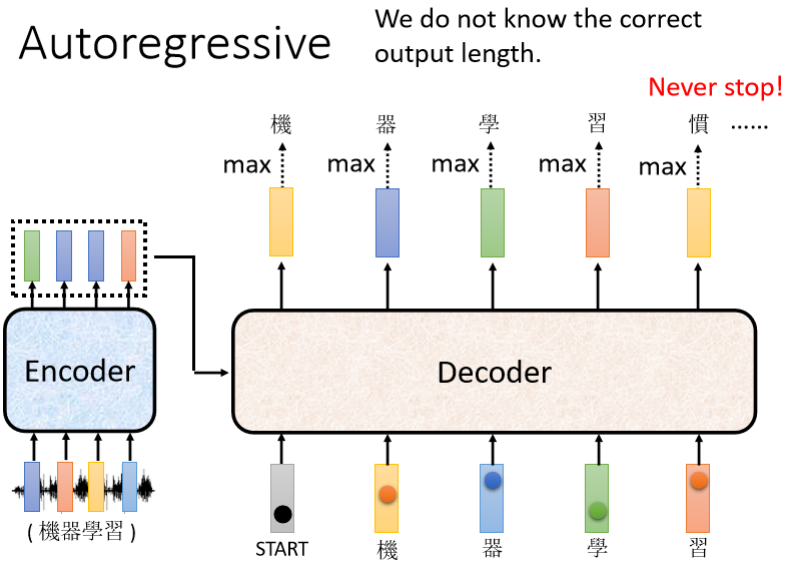
Self-attention → ***Masked Self-attention***



Why masked? Consider how does decoder work

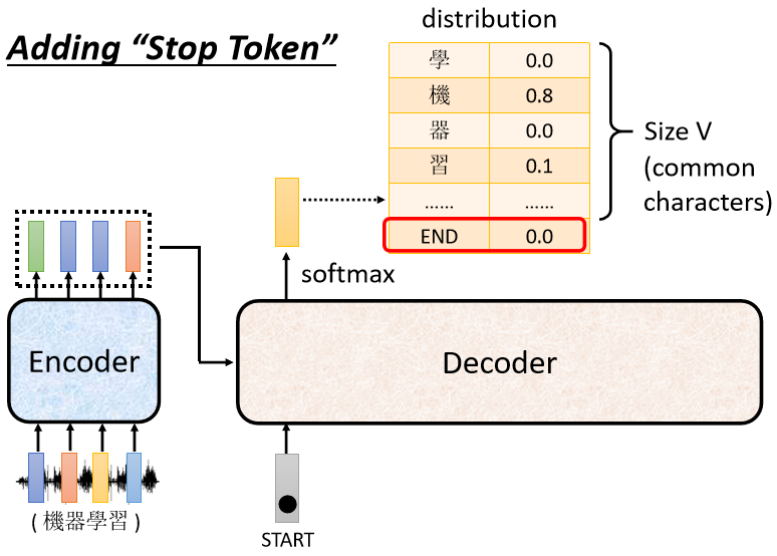
但是对 Decoder 而言，先有 a^1 才有 a^2，才有 a^3 才有 a^4，所以**实际上当你要计算 b^2 的时候，a^3 跟 a^4 还没有产生**，所以你根本就没有办法把 a^3 a^4 考虑进来。

另外对于 Decoder 来说还有一个非常关键的问题，**Decoder 必须自己决定输出 Sequence 的长度。**

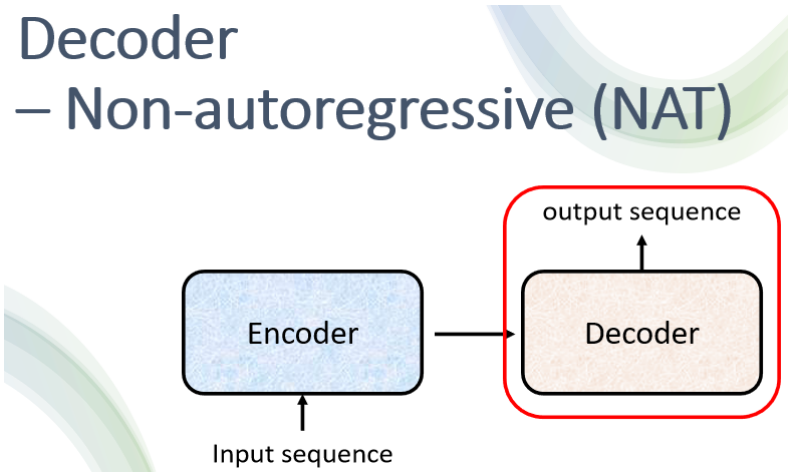


但在我们目前的这整个 Decoder 的运作机制下，**机器似乎并不知道它什么时候应该停下来**，它产生完“习”以后，完全可以继续重复一模一样的 Process，就把“习”当做输入，然后也许 Decoder 接下来就会产生一个“惯”，然后接下来又把“惯”作为输入，这个过程就会一直持续下去。

那怎么样机器知道什么时候应该停下来呢？其实这就跟我们要让机器知道什么时候开始一样，**我们也准备一个特殊的符号END**，如果机器输出的向量中对应概率最大的符号是END，就表明它现在应该停下来了。

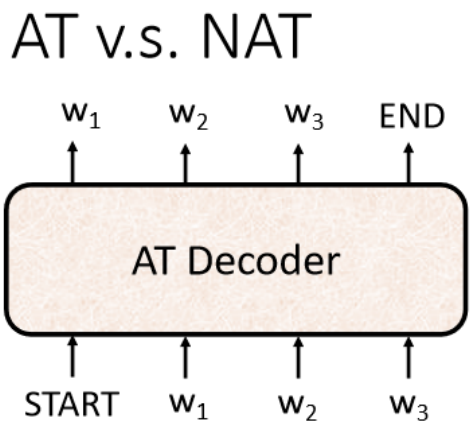


2 解码器 (NAT版) 的特别之处

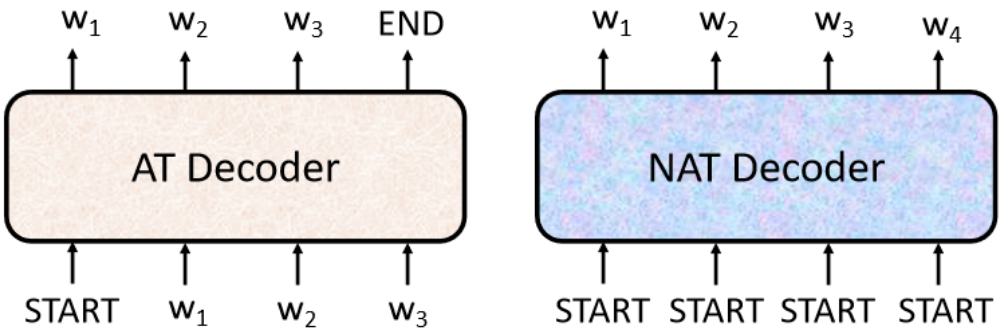


接下来我们来看看 Non-Autoregressive 的 Decoder 又是如何运作的，它与刚刚讲过的 Autoregressive 的 Decoder 又有哪些异同。

Autoregressive 的 Encoder 运作方式如下，**先输入 STRAT**，然后输出 w_1，再把 w_1 当做输入，输出 w_2，这个过程反复持续下去，**直到输出 END 为止。**

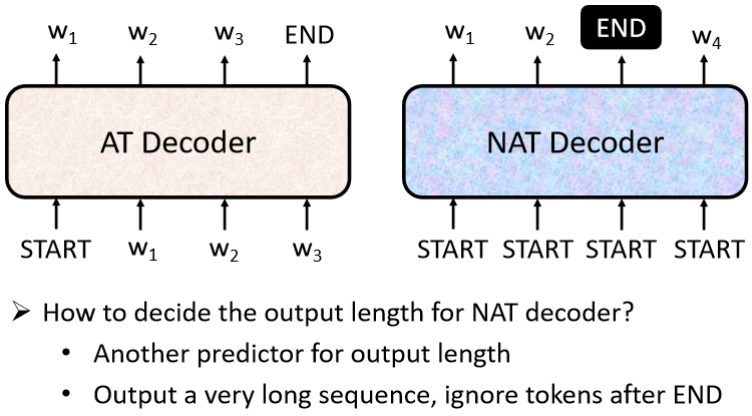


而 Non-Autoregressive 的运作方式却有所不同，它**不是依次产生$w_1w_2w_3w_4$，而是一次性全部产生。**



假设我们现在产生中文的句子，它不是依次产生每一个字，它是**一次把整个句子都产生出来。**NAT 的 Decoder **输入的可能是一整排的 START**，同时输入再一次输出全部的 Token 就结束了。举一个例子来说，如果你给它输入4个START，它就产生 4 个中文的字，变成一个句子，就结束了。所以它只要一个步骤就可以完成句子的生成。

那这似乎和最开始的说法矛盾了，刚才不是说不知道输出的长度应该是多少吗，这里怎么知道要输入多少个 START 来产生对应数量的输出?



的确这件事没有办法很自然的知道，但是仍然有几个做法能够做到预测输出的长度：

没错 这件事没有办法很自然的知道,没有办法很直接的知道,所以有几个,所以有几个做法

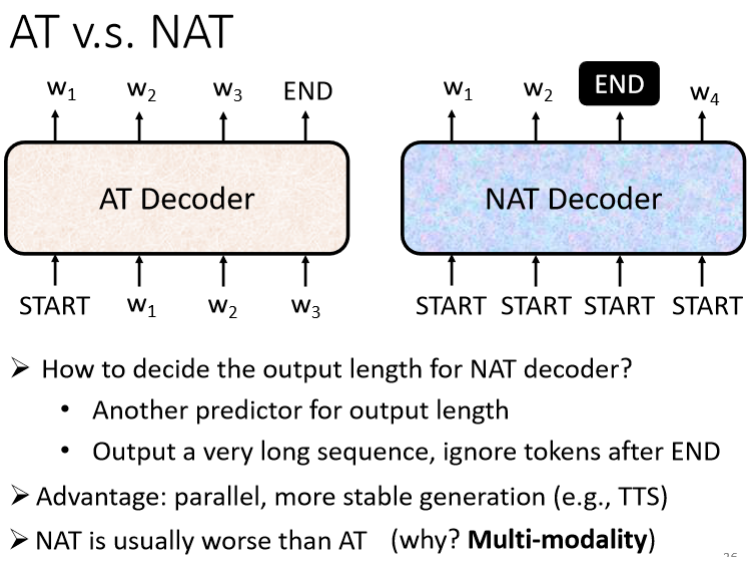
- 一个做法是，**另外训练一个 Classifier**，这个 Classifier 的输入就是 Encoder 的 Input，然后输出是一个数字，这个数字就代表 Decoder 应该要输出的长度，这是一种可能的做法。
- 另一种可能做法就是，直接不管三七二十一，**给机器输入一堆 STRAT**，假设说我们知道现在输出的句子长度,绝对不会超过 300 个字，然后就给机器输入 300 个 BEGIN，然后就会输出 300 个 Token，这时再看看**什么地方输出了 END，输出 END 右边的全部忽略掉就行了。**

那为什么还要设计这样一个 NAT 的 Decoder 呢，它**相较于Autoregressive Decoder有什么样的优势**呢?

- 它第一个优势是能够**并行化**产生输出，AT 的 Decoder 在输出它的句子时是一个字一个字产生的，但是 NAT 的 Decoder 不是这样，不管句子的长度如何，都是**一个步骤就产生出完整的句子**，所以在速度上，NAT 的 Decoder 显然会比 AT 的 Decoder 要快很多，这是一个很大的优势。

- 另外一个优势就是，你能够**一定程度上控制输出的长度**，拿语音合成为例，有一个模型叫 FastSpeech，它是 NAT 的 Decoder，你可能有一个 Classifier 来决定 NAT 的 Decoder 应该输出的长度，那如果在做语音合成的时候，假设你现在突然想要让你的系统讲快一点，那你就把那个 Classifier 的 Output 除以二，这样它讲话速度就变两倍快，就会变得言简意赅。

那 NAT 的 Decoder 最近之所以是一个热门研究主题，就是它虽然表面上看起来非常多的优势，但是实践证明 **NAT 的 Decoder，它的 Performance 往往都不如 AT 的 Decoder。**

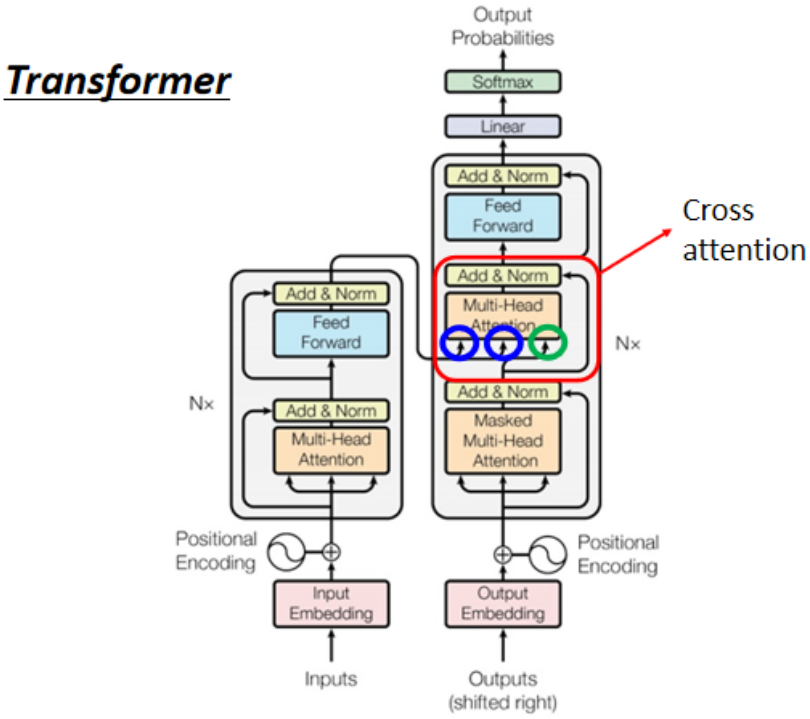


所以你会发现有很多的研究试图让 NAT 的 Decoder 的 Performance 越来越好，试图去逼近 AT 的 Decoder，不过这并不是一件容易的事情，至于为什么 NAT 的 Decoder Performance 不好，它其实存在一个叫做 Multi-Modality （多模态）的问题，如果想要这个深入了解 NAT，可以参看这个链接的讲解：(https://www.bilibili.com/video/BV1Wv411h7kN?p=36)。

---

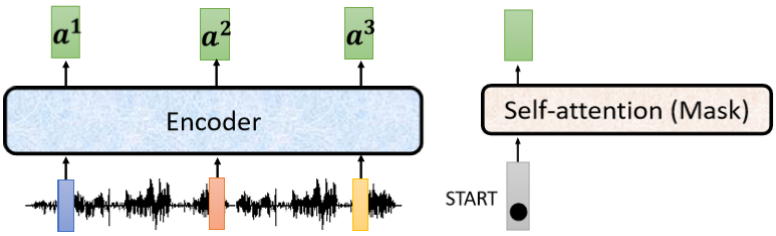
3 编码器与解码器之间的信息传递

接下来就要讲 **Encoder 跟 Decoder 之间是怎么传递资讯**的了，也就是刚刚遮起来的那一块到底做了什么。

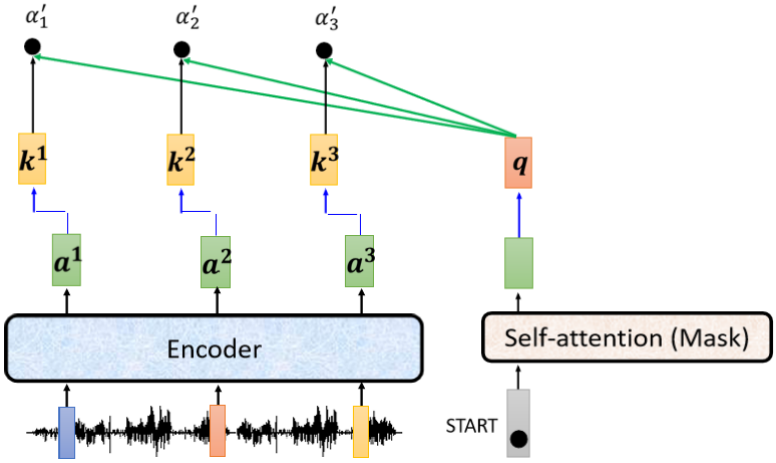


遮住的这块叫做 Cross Attention，它是连接 Encoder 跟 Decoder 之间的桥梁，在这一块里面，我们会发现有**两个输入来自于 Encoder，另外一个输入来自于 Decoder**，所以从左边这两个箭头，Decoder 就可以获取到 Encoder 的输出。

那接下来就通过一个具体的例子展示一下这个模块的运作过程：

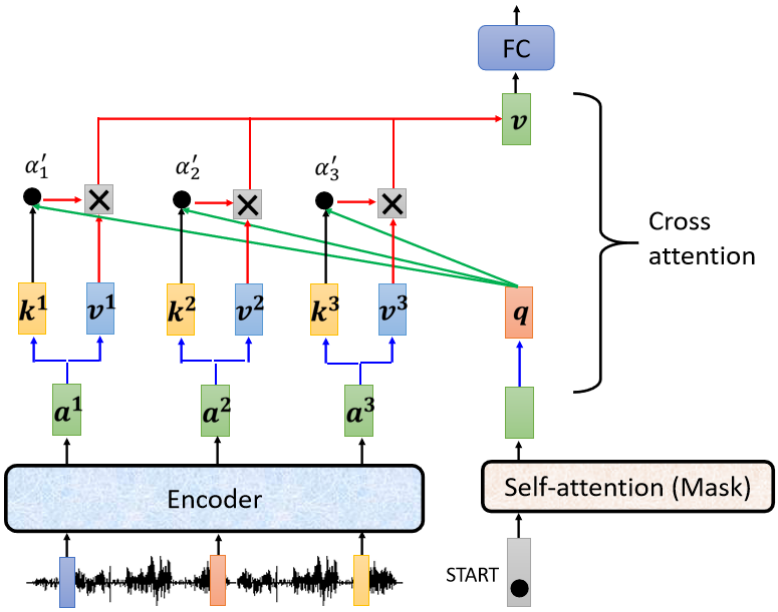


首先是我们的 Encoder，输入一排向量，输出一排向量$a^1a^2a^3$，接下来来看 Decoder，Decoder 会先输入代表START 的向量，然后会经过 Masked Self-Attention，然后输出一个向量，然后接下来这个向量乘上一个矩阵做一个 Transform，得到一个 Query 叫做 q：



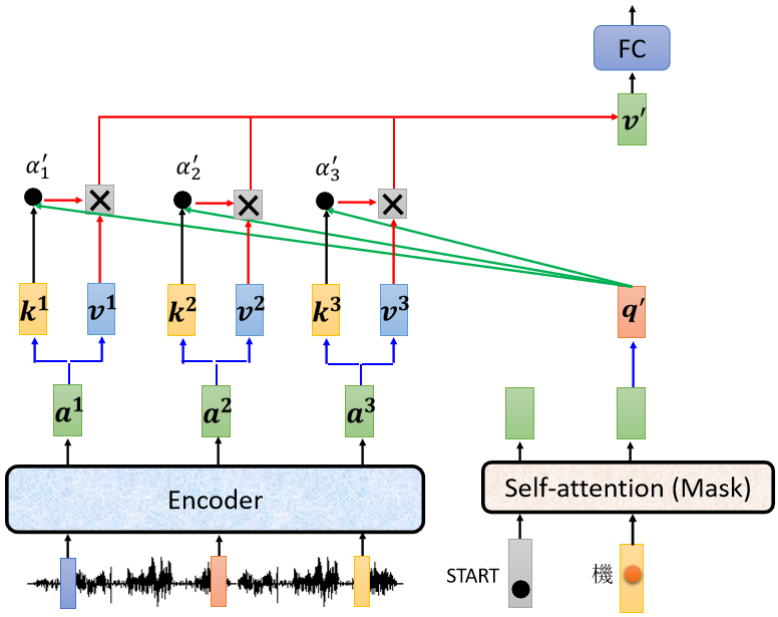
然后这边的 $a^1a^2a^3$ 也都通过乘上一个矩阵产生 Key，$k^1k^2k^3$，把这个 q 跟 $k^1k^2k^3$ 去计算 Attention 的分数，得到 $\alpha_1\alpha_2\alpha_3$，把 $\alpha_1\alpha_2\alpha_3$ 做一下 Normalization，得到 $\alpha'_1\alpha'_2\alpha'_3$。

接下来再把 $\alpha'_1\alpha'_2\alpha'_3$ 乘上 $v^1v^2v^3$，再把它们乘出来的结果加在一起就得到 v。这个 v 会作为接下来的 Fully-Connected Network 的输入，做进一步的处理。那产生 v 的这一系列过程就是 Cross Attention。

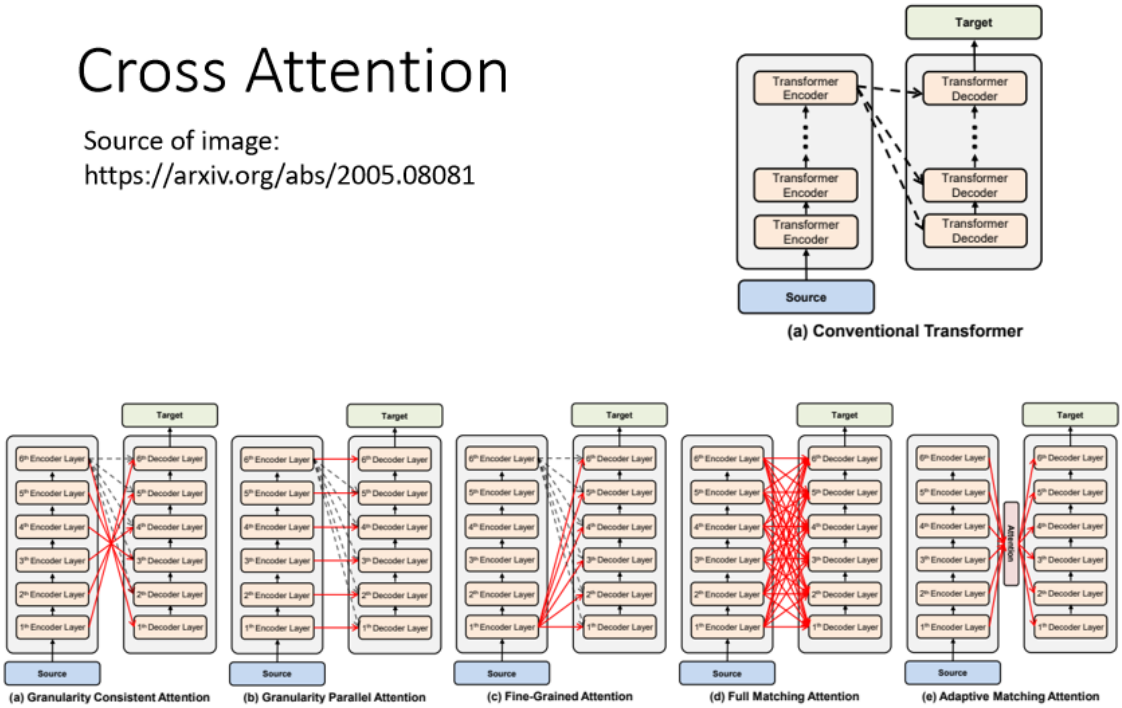


所以 **Decoder 就是通过产生一个 q，去 Encoder 的输出中提取有用的信息，作为接下来的 Decoder 的 Fully-Connected Network 的 输入。**

现在假设第一个字“机”以及产生出来了，现在产生第二个输出，接下来的运作过程也是一模一样的:



这里也许有人会有疑问：那这个 Encoder 有很多层，Decoder 也有很多层，从刚才的描述里面好像听起来，这个 Decoder 不管哪一层,都是**拿 Encoder 的最后一层的输出。在原始的论文里面确实是这样做的，但不一定非得这样**，下面引用一篇论文 Layer-Wise Cross-View Decoding for Sequence-to-Sequence Learning (arxiv.org)，其实也有人尝试不同的 Cross Attension 的方式，**这完全是一个可以进一步研究的问题。**

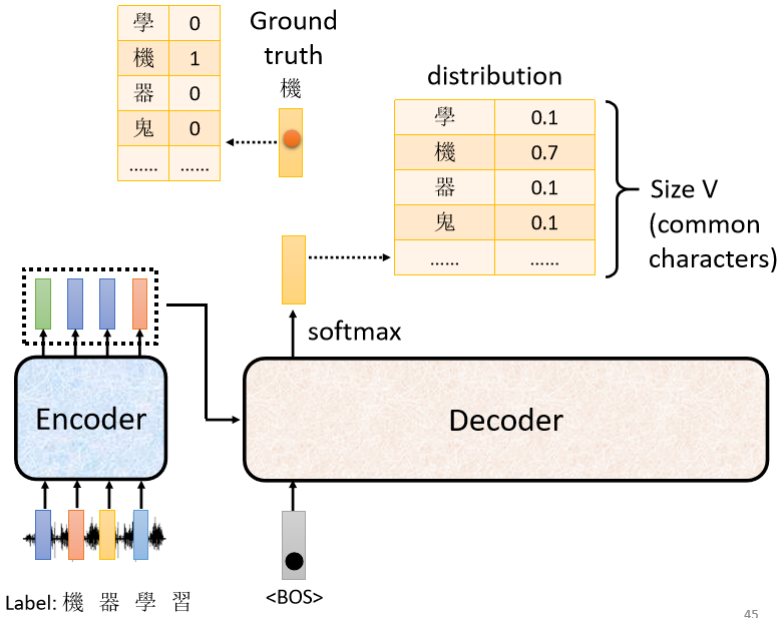


4 怎么来训练我们的Transformer呢

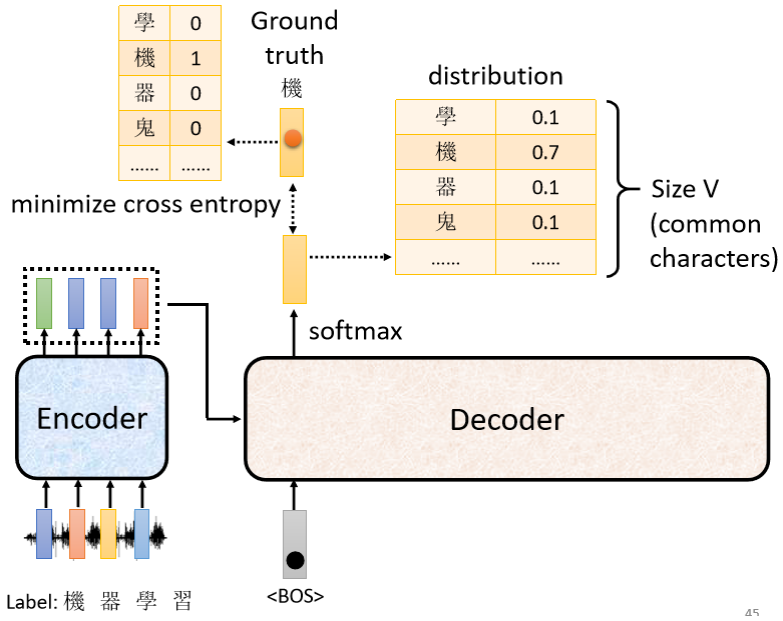
还是假设我们现在要做语音辨识的任务，那**首先需要的是训练资料**，我们需要收集一大堆的声音讯号，并且每一句声音讯号都要有对应的词汇信息。

Label: 機 器 學 習

那现在我们知道输入这段声音讯号，第一个应该要输出的中文字是“机”，所以今天当我们把 BOS，输入给 Transformer 的时候，它的第一个输出应该要跟“机”越接近越好。



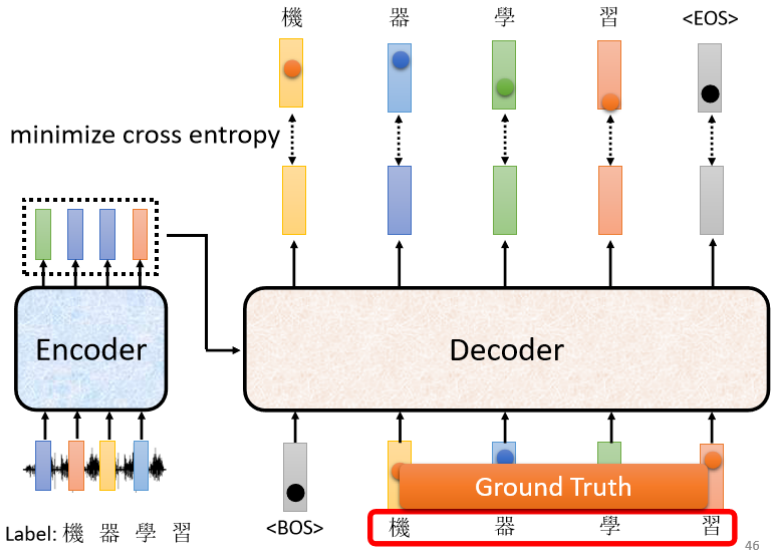
之前提到过，Decoder 的输出是一个概率的分布，我们会希望这一个概率的分布，跟"机"的 One-Hot Vector 越接近越好，所以我们会去计算这个 Ground Truth 跟这个 Distribution 它们之间的 Cross Entropy，然后我们希望这个 Cross Entropy 的值越小越好。



实际上你会发现这件事情**跟分类很像**，每一次 Decoder 产生一个中文字的时候，其实就是做了一次分类的问题，中文字假设有四千个，那就是**做有四千个类别的分类的问题。**

所以实际上训练的时候，我们已经知道输出应该是"机器学习"这四个字，所有我们希望我们的输出，跟这四个字的 One-Hot Vector 越接近越好，由于每一个输出都会有一个 Cross Entropy，**我们希望所有的 Cross Entropy 的总和越小越好。**

Teacher Forcing: using the ground truth as input.



在 Decoder 训练的时候，**我们会在输入的时候给它正确的答案**，这就叫做 Teacher Forcing

但这个时候你马上就会有一个问题产生：训练的时候 Decoder 偷看到正确答案了，但是在测试的时候，显然没有正确答案可以给 Decoder 看。这**中间显然有一个 Mismatch**，这个问题我们后面会通过 Scheduled Sampling 的方法来解决，这里先不做介绍。

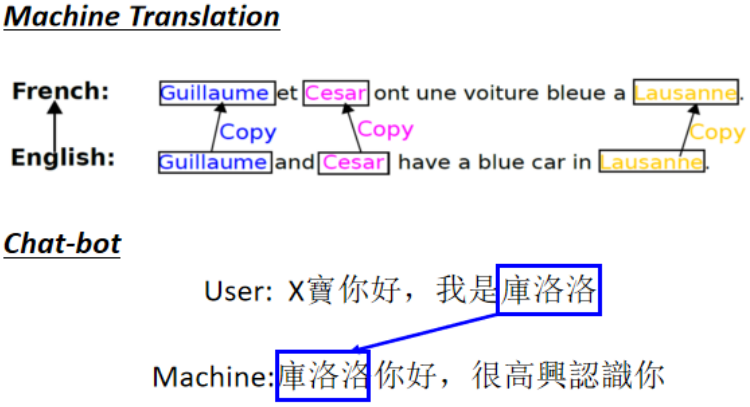
---

5 训练 Seq2seq model 的一些 Tips

5.1 Copy Mechanism

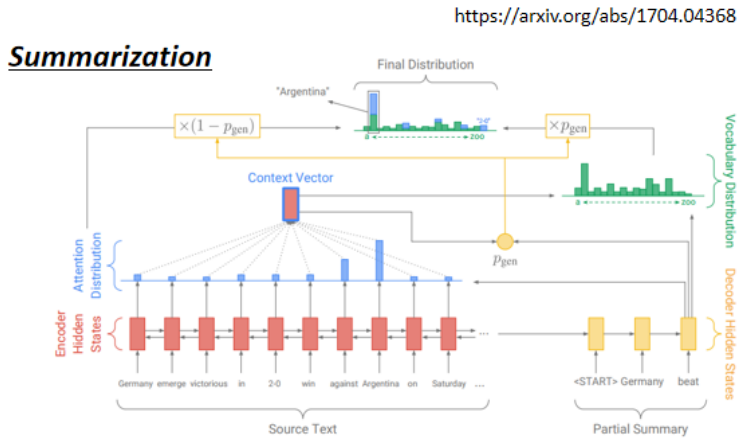
在刚才的讨论中，我们都要求 Decoder 自己产生输出，但是对很多任务而言，也许 **Decoder 没有必要自己创造全部的输出**，它**有些输出也许只需要从输入中复制过来**就可以。比如说聊天机器人的任务：

- 人对机器说：你好，我是库洛洛
- 机器应该回答说：库洛洛你好，很高兴认识你



对机器来说，它**完全没有必要自己创造**库洛洛这个词汇，这对机器来说一定会是一个非常怪异的词汇，所以它可能很难正确地产生这个词，但是假设今天机器在学习的时候，学到的是看到输入的时候说我是某某某，就直接把某某某复制出来说某某某你好，这样显然**训练起来会比较容易。**

或者机器要完成的任务是对一篇文章做摘要的，这种时候可能更需要 Copy 这样的技能。



要做摘要这样的任务，**只有一点点的资料是训练不起来的**，你要让机器学会说合理的句子，**通常上百万篇文章是需要的**，而对摘要这个任务而言，**从文章里直接复制一些资讯出来**，可能是一个很关键的能力。那 Sequence-To-Sequence Model 是如何做到这件事情的呢？这里不会细讲，这里介绍一个最早有从输入复制东西的能力的模型，叫做 Pointer Network，后来还有一个变形,叫做 Copy Network。

5.2 Guided Attention

举一个语音合成的例子来说明为什么需要 Guided Attention 这个做法。

我们训练了一个 Sequence-To-Sequence 的 Model，方式如下：

- 收集很多的声音，文字跟声音讯号的对应关系
- 然后接下来告诉你的 Sequence-To-Sequence Model，看到这段中文的句子，你就输出这段声音
- 就这样直接训练下去就结束了

像这样的方法做出来结果其实还不错，举例来说我叫机器连说 4 次发财，机器输出的结果是:发财 发财 发财 发财，甚至还有一些抑扬顿挫。

高雄發大財我現在要出征

發財發財發財發財

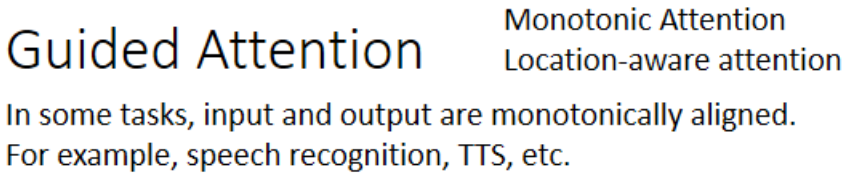
發財發財發財

發財發財

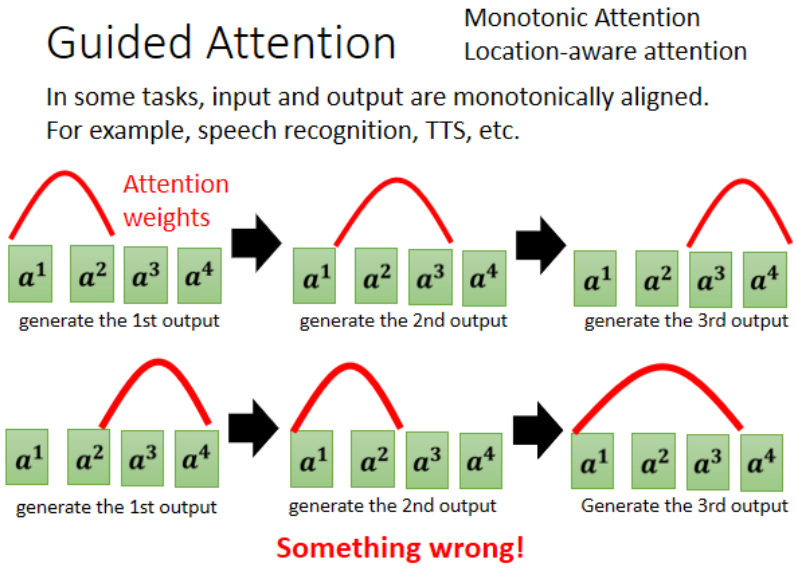
發財　　(Missing an input character!)

那你让它讲 3 次发财也没问题，讲 2 次发财也没问题，但是让它讲 1 次发财，**它只念了“财”而没念“发”。** 一个可能的解释是，**也许在训练资料里面，这种非常短的句子很少**，所以机器不知道要怎么处理这种非常短的句子。

那我们刚才发现说机器居然漏字了，输入有一些东西它居然没有看到，那我们能不能够**强迫它一定要把输入的每一个东西通通看过**呢，这个是有可能的，这个做法就叫做 Guided Attention。



Guiding Attention 要做的事情就是，**要求机器在做 Attention 的时候，需要遵循一个固定的方式**，举例来说，对语音合成或者是语音辨识来说，我们想像中的 **Attention应该就是由左向右。**



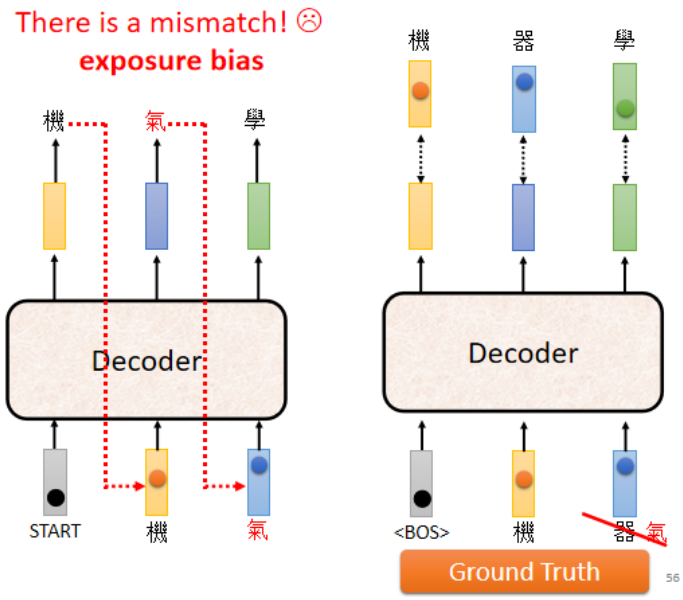
如果你今天在做语音合成的时候发现机器的 Attention 是**颠三倒四的**，它先看最后面，接下来再看前面，再胡乱看整个句子，那显然有些是做错了。所以 Guiding Attention 要做的事情就是，强迫 Attention 有一个固定的模式，就直接把这个限制放进你的 Training 里面，要求机器学到的 Attention 就应该要由左向右。

那这件事怎么做呢，有一些关键词汇比如说 Mnotonic Attention 和 Location-Aware 的 Attention，这个部分也是个大坑，所以也不细讲，感兴趣的可以自己进一步研究。

5.3 Scheduled Sampling

之前我们提到，训练的时候 Decoder 偷看到正确答案了，但是在测试的时候，显然没有正确答案可以给 Decoder 看。这**中间显然有一个 Mismatch**，那处理这个问题的其中一个做法，就叫做Scheduled Sampling。

测试的时候，Decoder 看到的是自己的输出，所以测试的时候 Decoder 会看到一些错误的东西，但是在训练的时候 Decoder 看到的是完全正确的，这个不一致的现象就叫做 Exposure Bias。



假设 Decoder 在训练的时候，永远只看过正确的东西，那在测试的时候，只要有一个错，那就会**一步错，步步错**，所以要怎么解决这个问题呢?

我们可以想到的是，既然是因为 Decoder 在训练的时候没有看到过错误的东西导致了这个问题，**那在测试的时候给 Decoder 的输入加一些错误的东西，是不是就能解决这个问题呢?** 事实就是这样，偶尔给它一些错的东西，它反而会学得更好，这个办法就叫做 Scheduled Sampling。