

end2end asr

[seq2seq](#)

[encoder](#)

[decoder](#)

[attention](#)

[attention图示](#)

[attention公式](#)

[attention通用定义](#)

[self-attention](#)

[scaled dot-product attention](#)

[multi-head attention](#)

[transformer](#)

[ctc](#)

[rnn transducer](#)

[开源实现](#)

[总结](#)

序列到序列，一般是声学特征到文本。

seq2seq

encoder and decoder, 其中包含两个rnn, 输入和输出可以不等长。

encoder使用rnn将声学特征进行编码, decoder的输出为目标文本, 两者之间通过encoder rnn的最后一时刻隐状态联系在一起, 最后时刻隐状态包含了输入声学特征的所有信息。

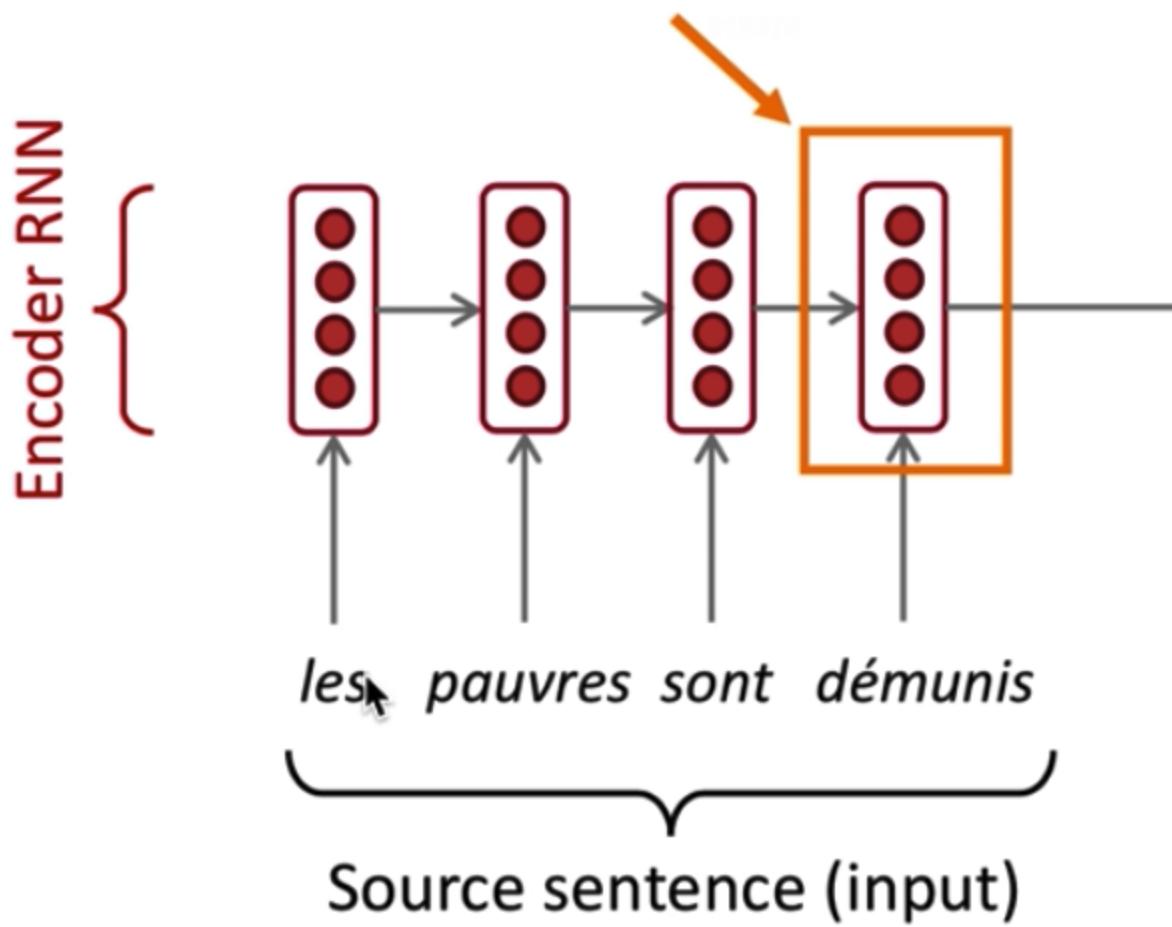
encoder

以机器翻译为例, encoder rnn将输入句子编码成一个个向量

The sequence-to-sequence model

Encoding of the source sentence.

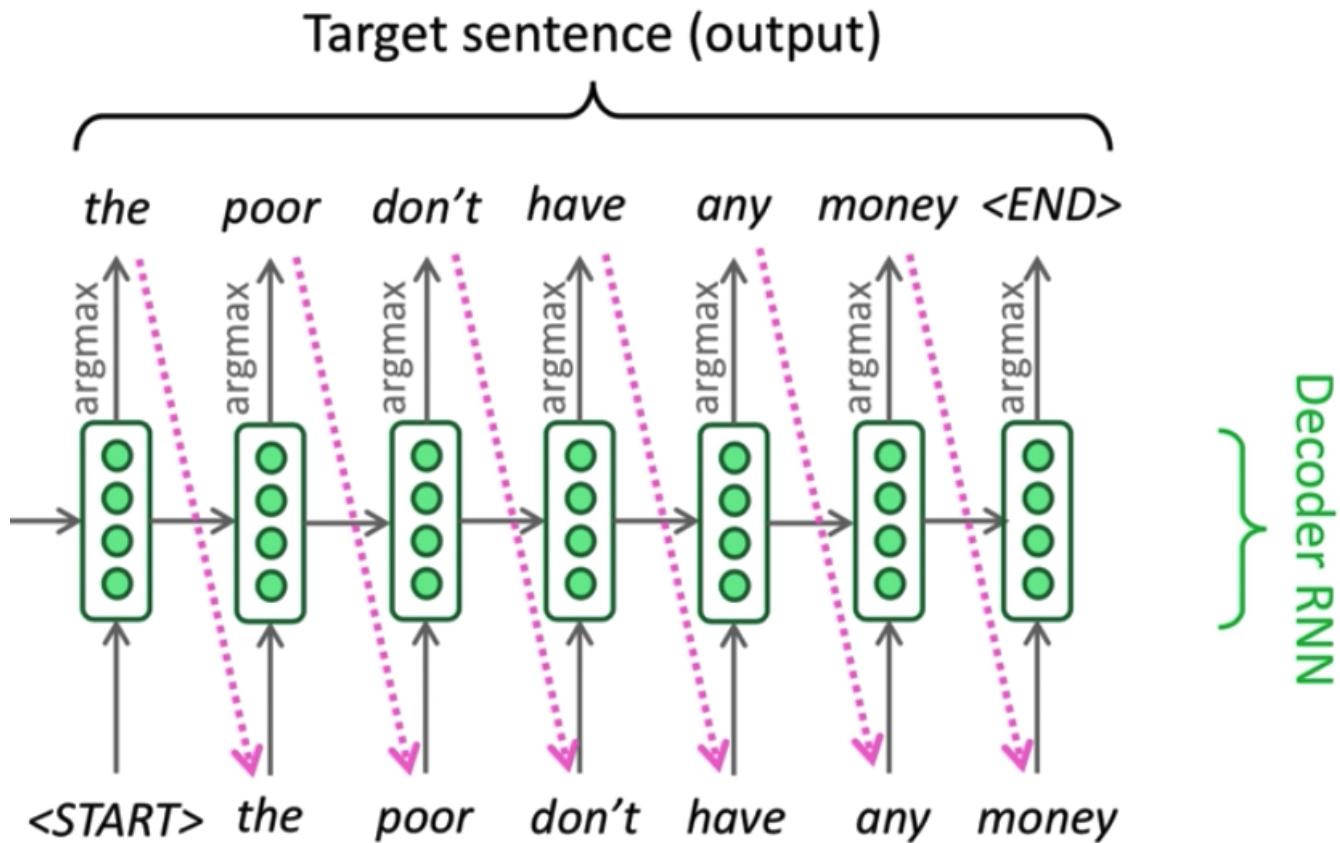
Provides initial hidden state
for Decoder RNN.



Encoder RNN produces
an encoding of the
source sentence.

decoder

decoder rnn初始时刻以<start>开始，结合encoder最后时刻隐状态向量，通过rnn生成一个label，这个label作为下一时刻输入，继续进行rnn计算，直到遇到<end>结束。这是一个自回归的过程。

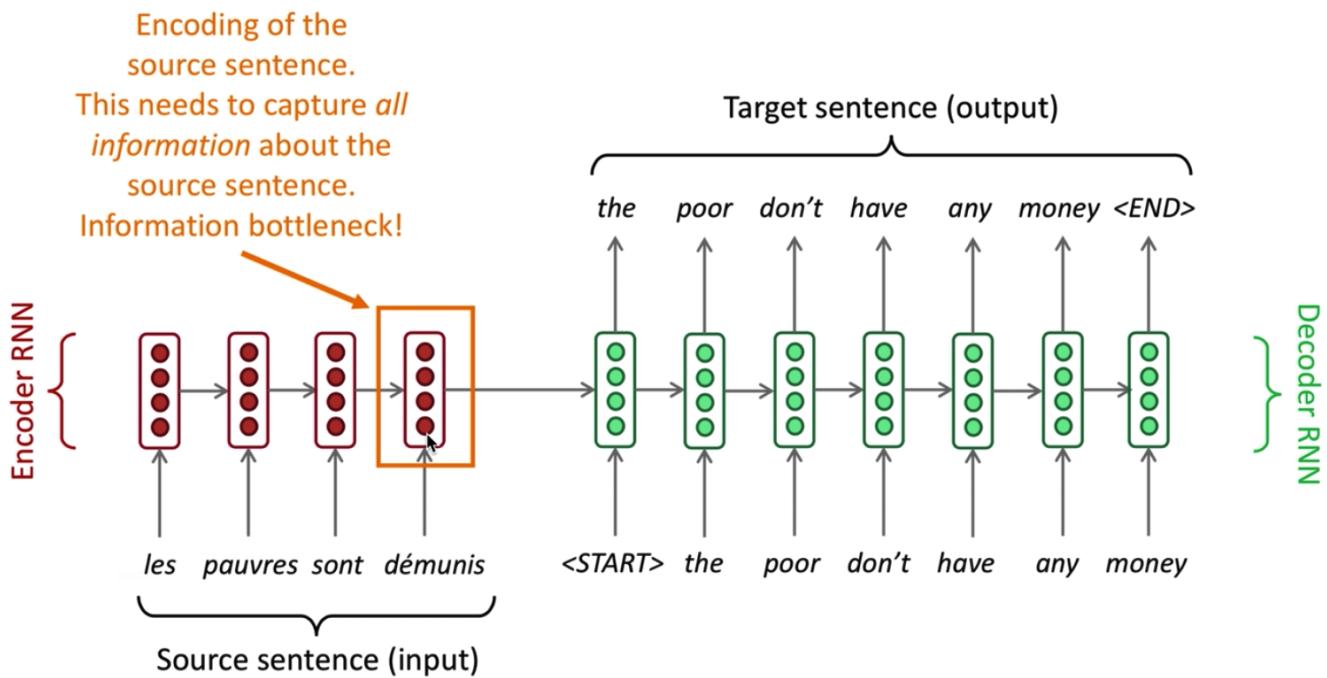


Decoder RNN is a Language Model that generates target sentence conditioned on encoding.

Note: This diagram shows test time behavior:
decoder output is fed in> as next step's input

attention

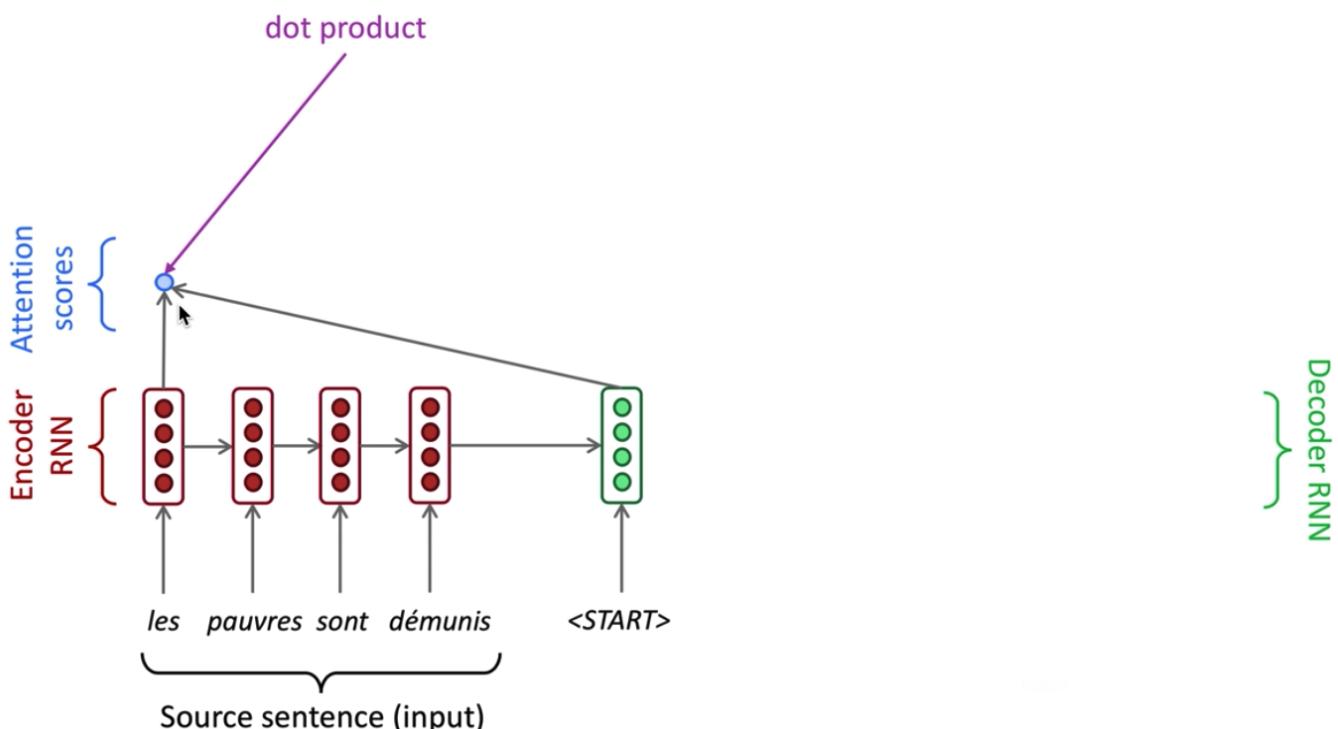
encoder decoder模型中，将encoder最后一个时刻的输出作为decoder的输入，当句子太长或者向量维度较小时，可能无法将原句子信息编码到这个向量里面，导致成为性能瓶颈。



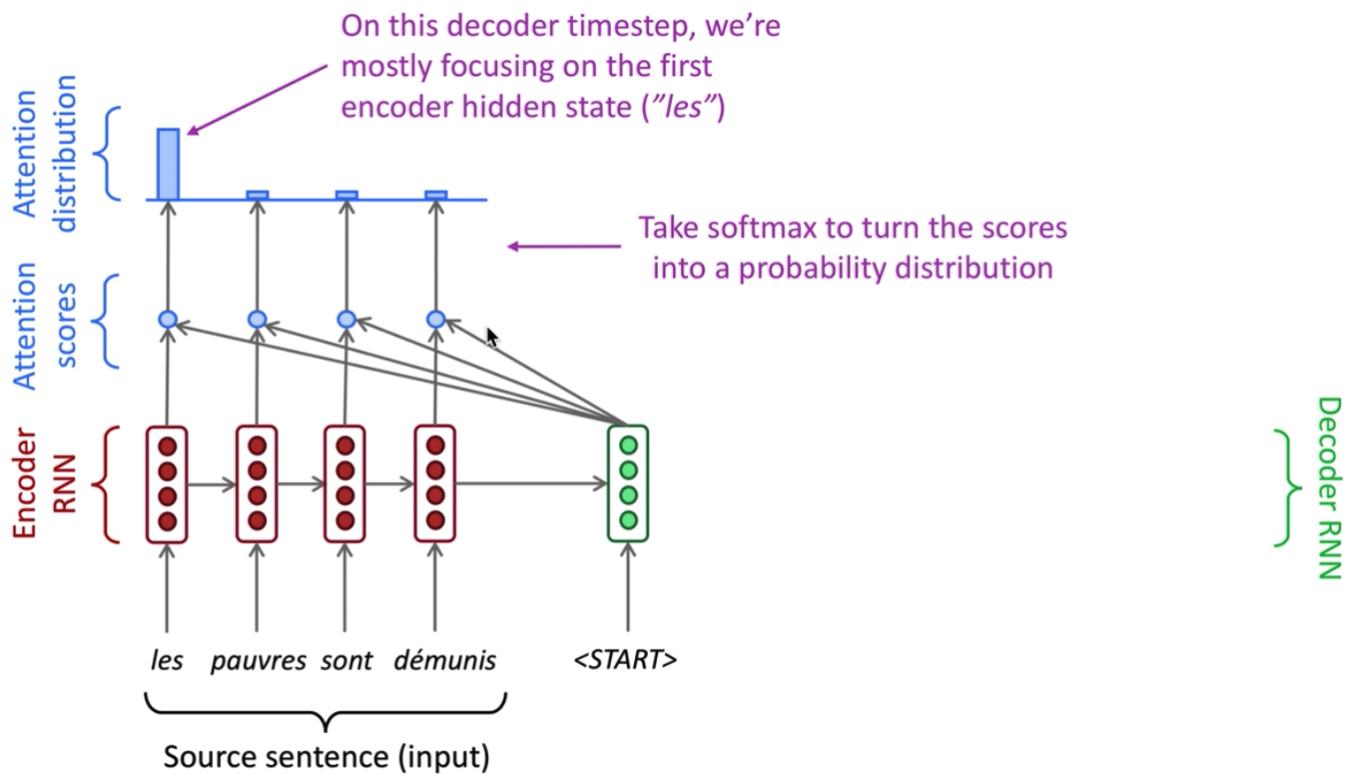
解决方法的关键点在于，在decoder的每一步，只关注原句子的特定部分。

attention图示

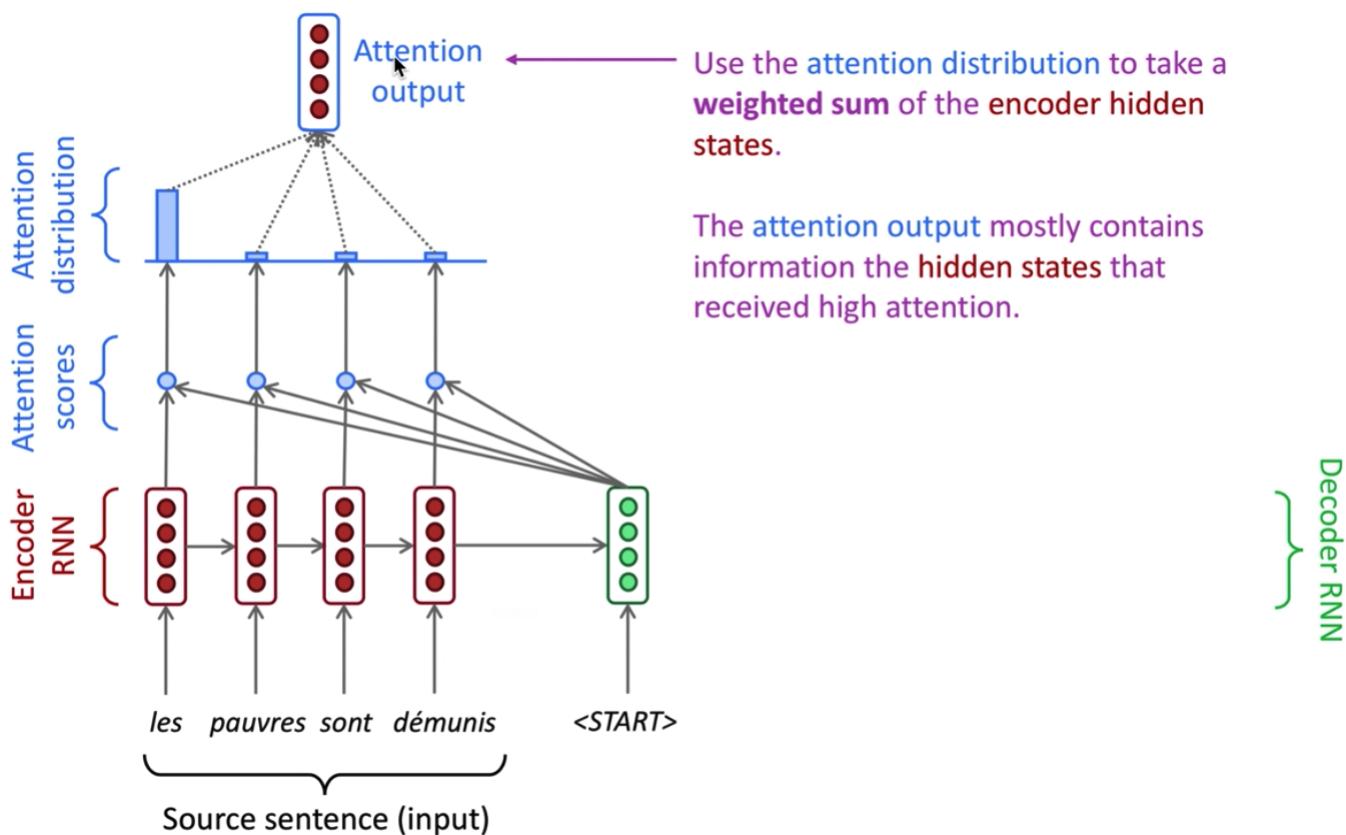
step1: 假设encoder已经将文本输入转成向量序列，每个向量是四维。attention时，将decoder的输入与encoder的每个向量输出做点积，即decoder输入向量和每个四维的encoder输出向量做点乘然后相加得到一个score。



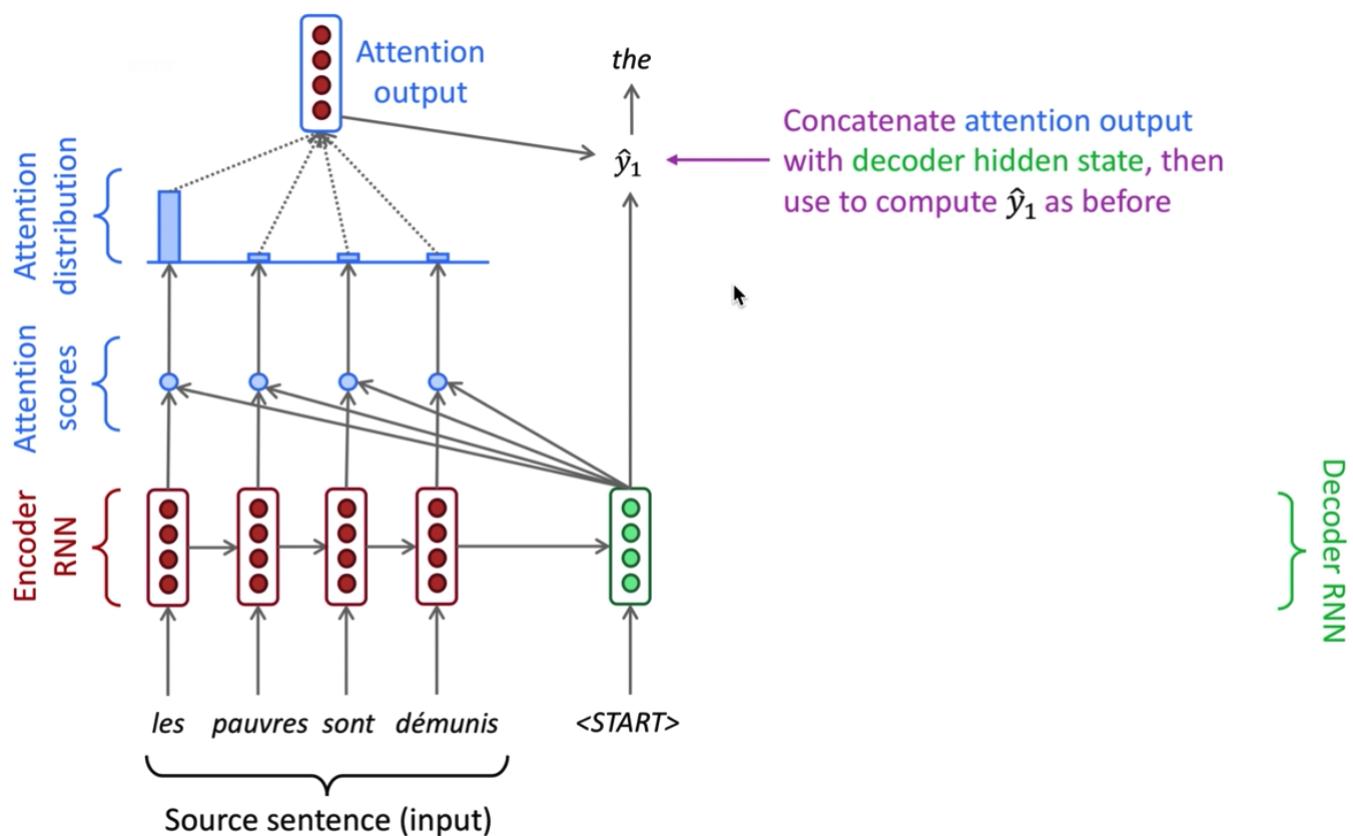
step2: 对每个分数做一个softmax，得到一个概率分布。



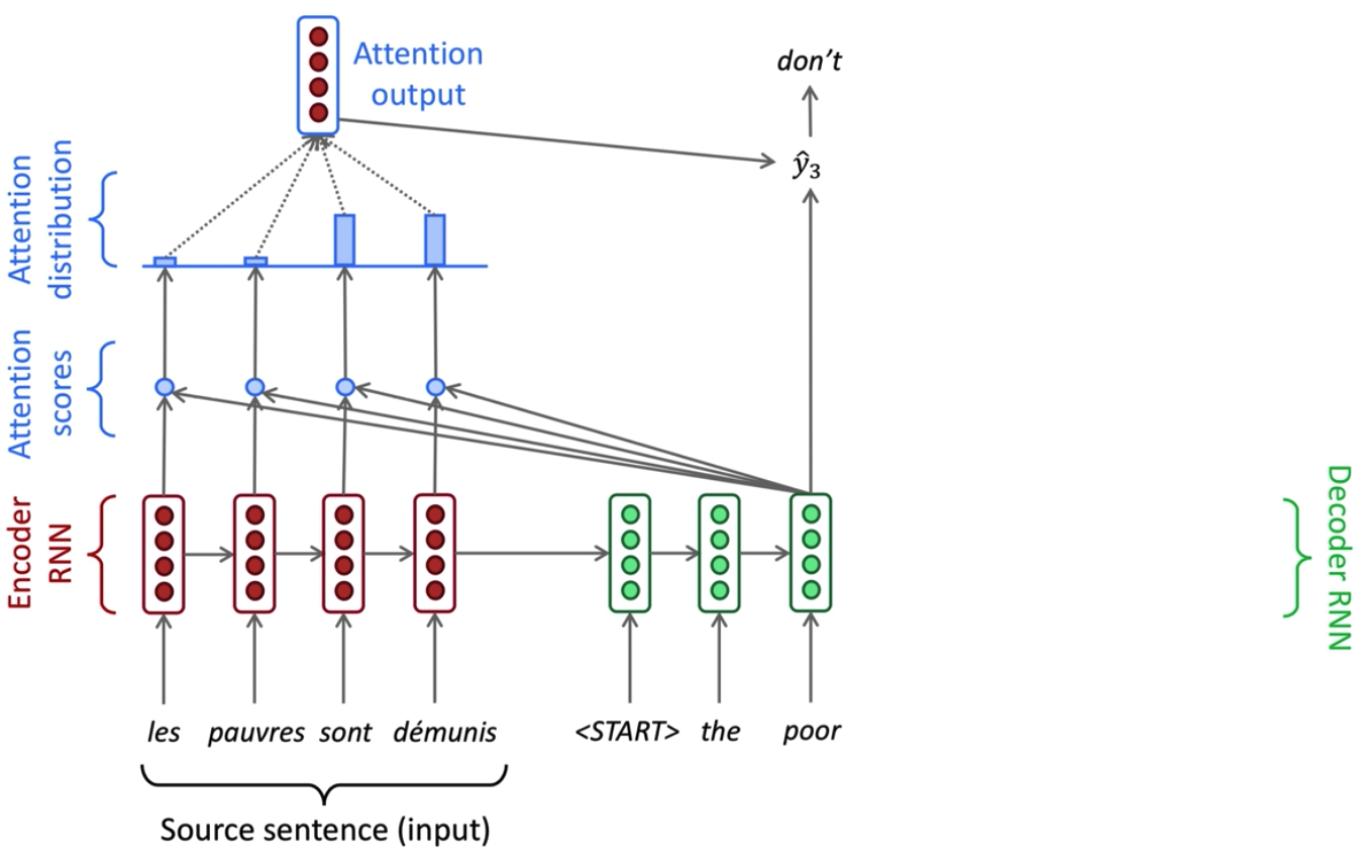
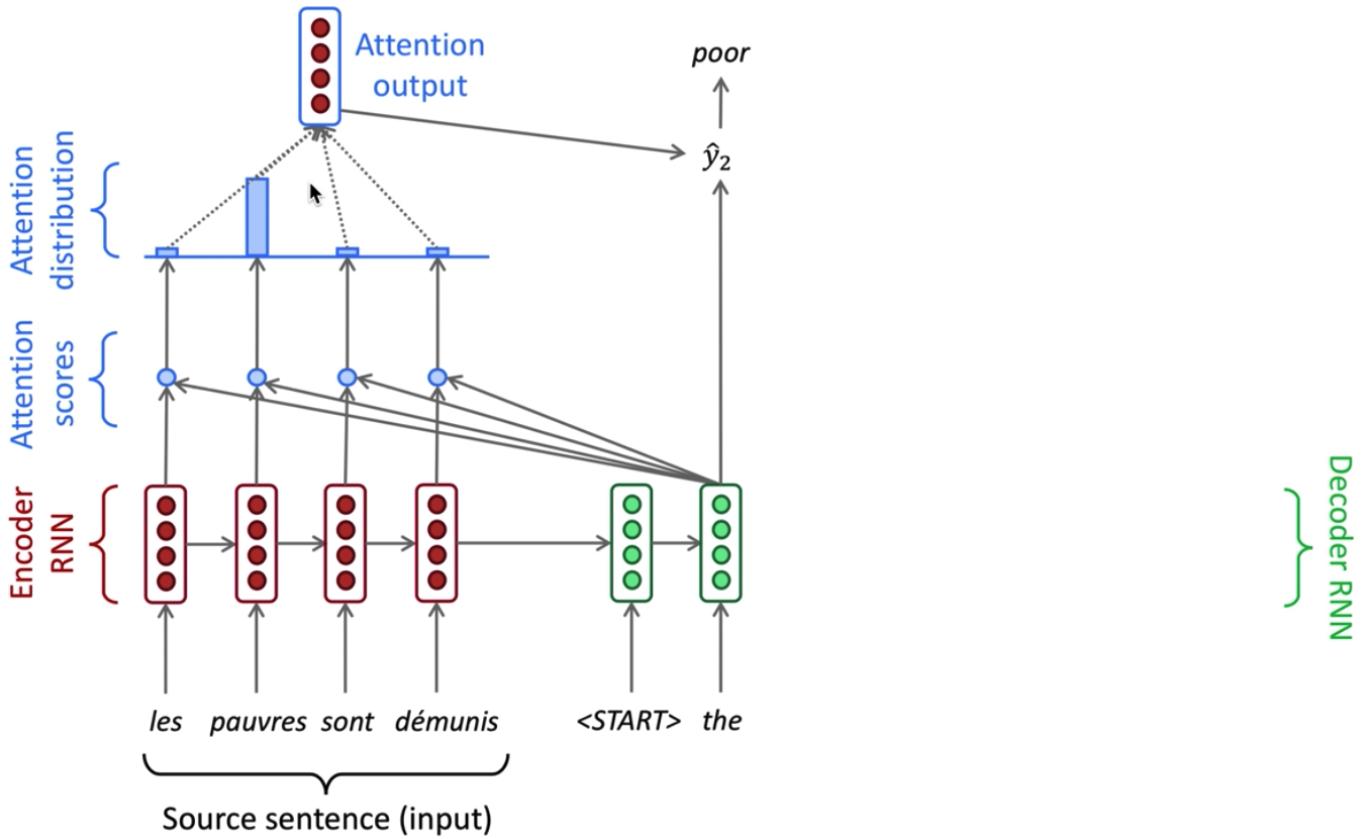
step3: 将这个四个概率作为权值，与encoder输出的四个向量做加权和，得到一个四维的向量，这个向量代表第一步，decoder关注的整个encoder的总和。可以看到第一个时刻关注的信息大部分来自第一个时刻的encoder向量。这样就实现了decoder在每个时刻关注encoder特定部分的信息。

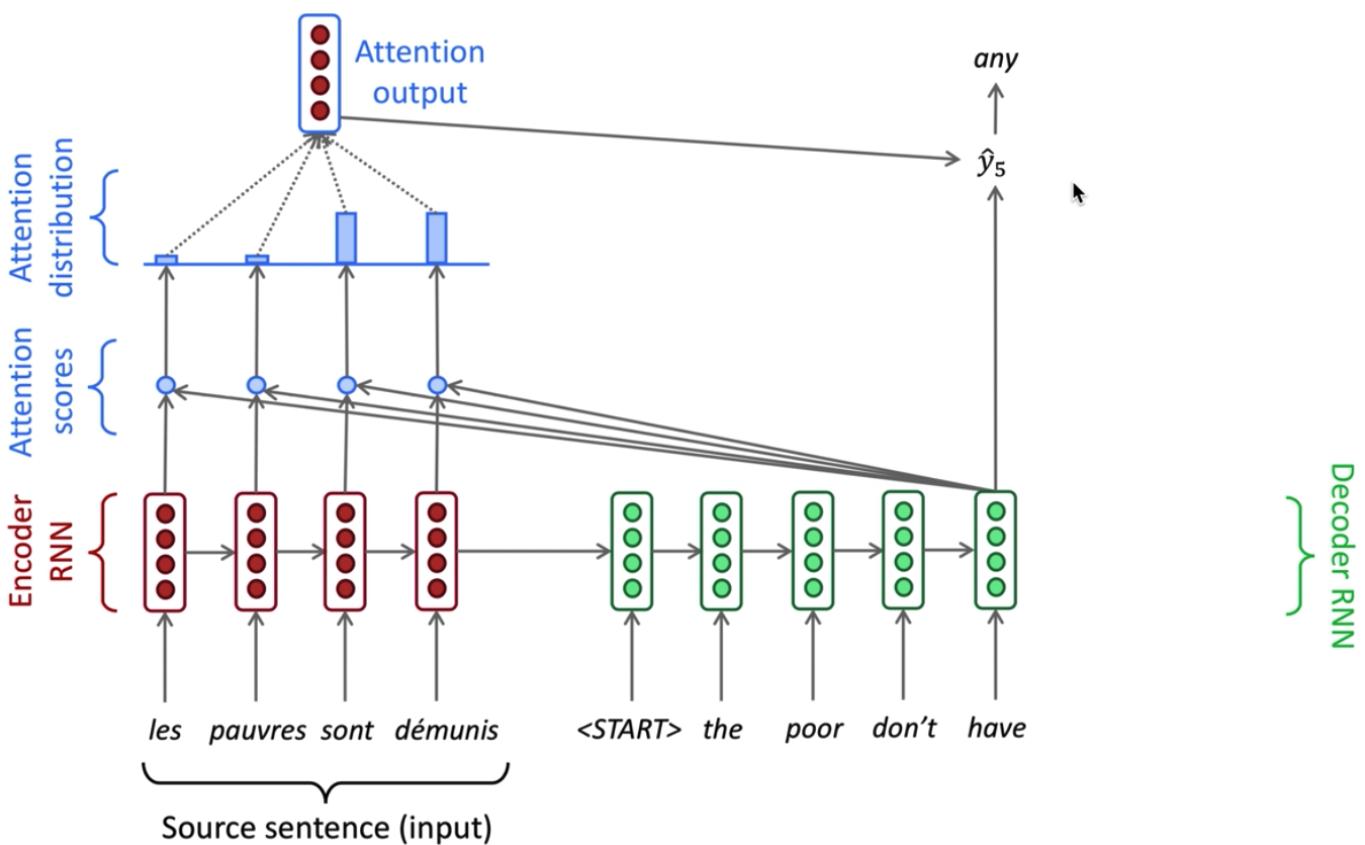
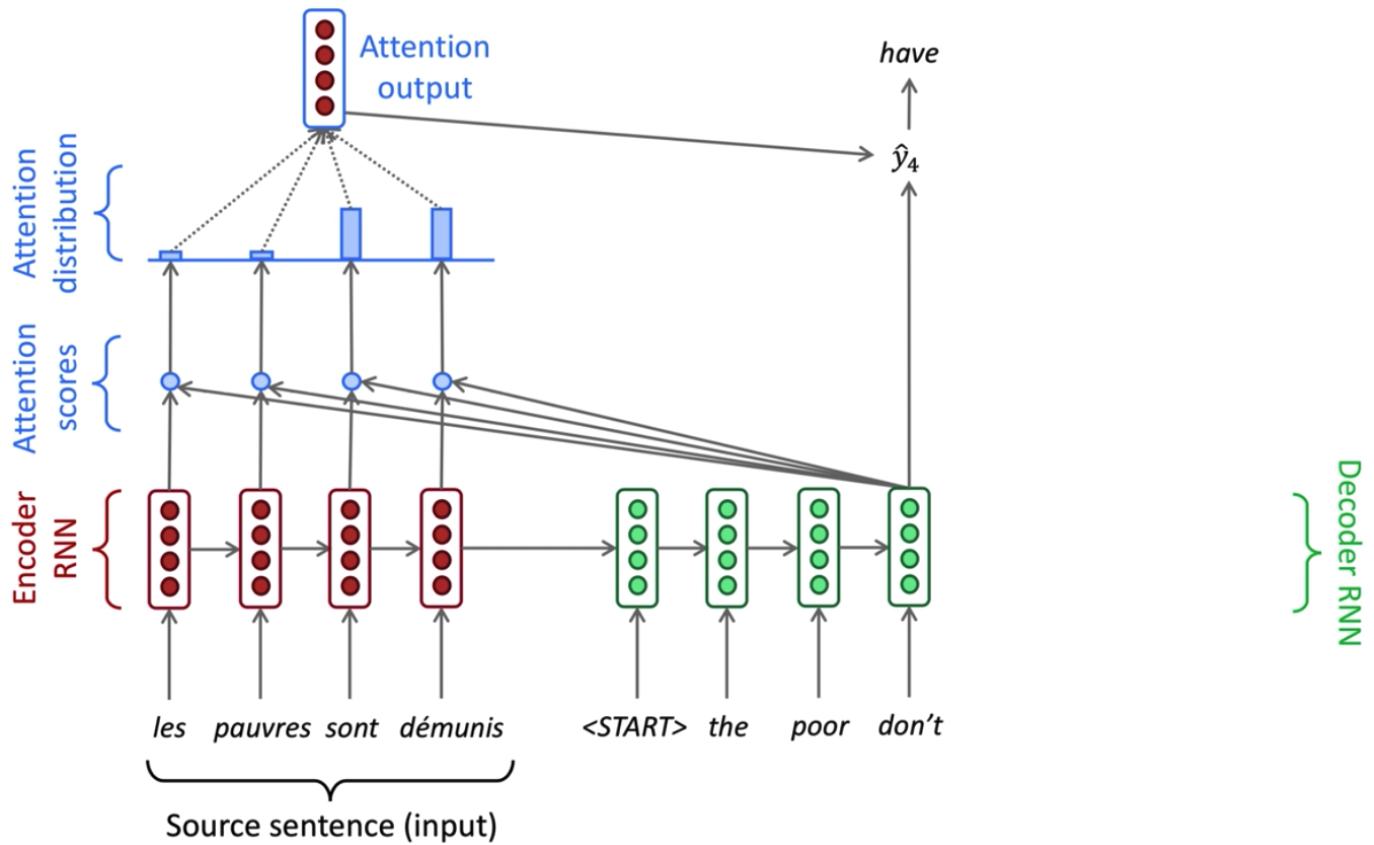


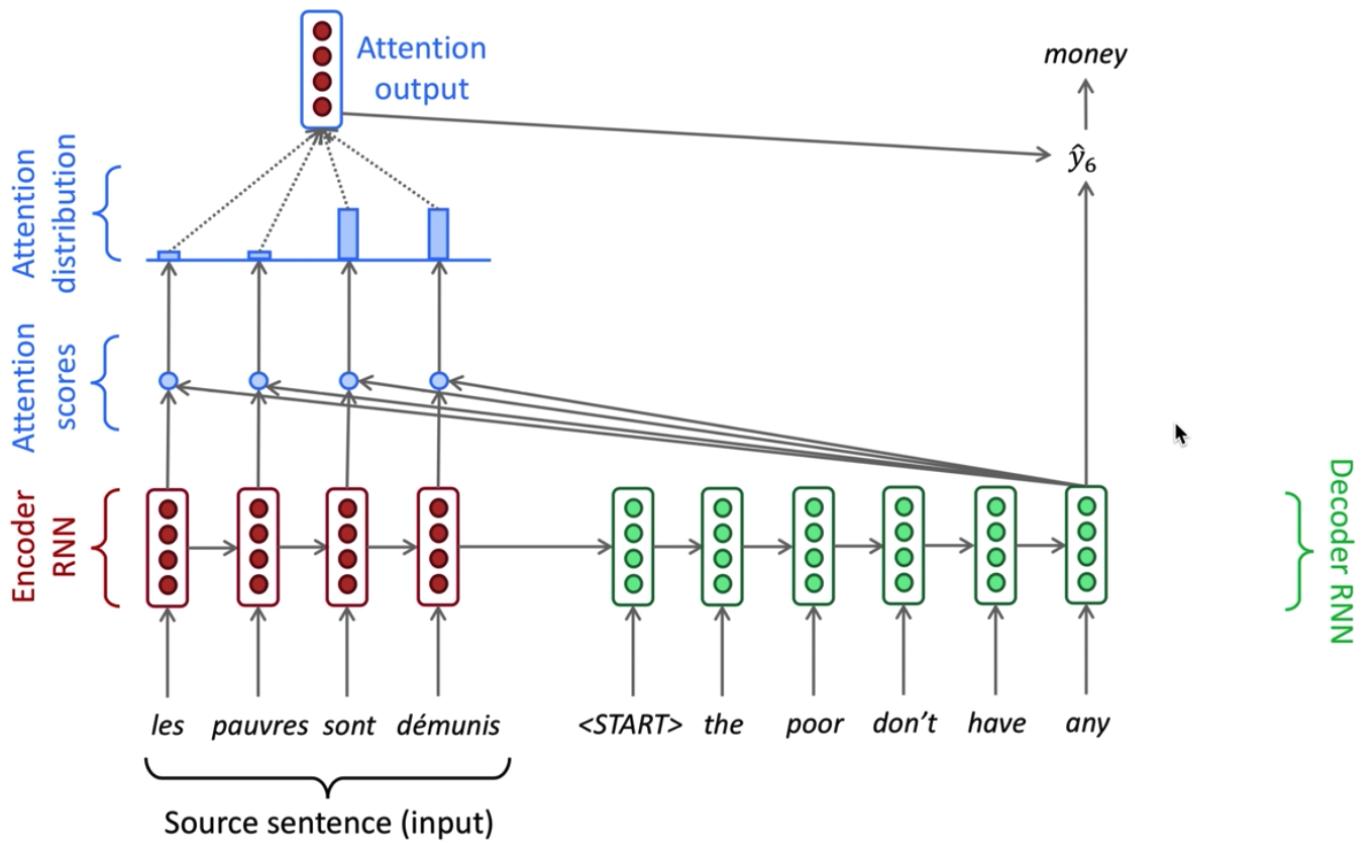
step4: 然后将attention的向量和decoder输出向量拼在一起，通过一个dnn、mlp或者再加一层rnn，求出最后的输出。



step5: 后面每个时刻计算和前面一样。







一直到预测的`<end>`结束，注意图中attention的分布，不同时刻关注encoder的不同部分。

attention公式

Attention : 公式

- We have encoder hidden states $h_1, \dots, h_N \in \mathbb{R}^h$
- On timestep t , we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

- 第一步通过encoder得到隐状态表示，总共N个时刻，每个词汇是一个 h 维的向量
- 在decoder的t步，有decoder的隐状态 s_t
- 把 s_t 和encoder的每一步算一个dot product，点积后会得到一个实数 $s_t^T h_n$ ，一共有N步，所以得到N维的向量
- 对 e^t 做softmax，转成一个概率分布 α^t ，N个值加起来总和为1
- 用 α^t 作为权值，对encoder的N个隐状态向量做点积和加权和，得到一个 a_t ，是一个 h 维的向量
- 最后将attention的输出和decoder的隐状态拼起来作为输入，输入进一个dnn、mlp或者rnn，计算最后的输出

attention通用定义

Given a set of vector **values**, and a vector **query**, attention is a technique to compute a weighted sum of the values, dependent on the query.

给定一个输入向量**values**的集合和一个查询向量**query**，根据query对value进行加权和。

对应到前面，输入向量为encoder的多个隐状态向量，查询向量为decoder的每个时刻的隐状态向量，两者进行点乘得到权值。

向量的加权和相当于把**values**多个向量的信息做一个有挑选的总结，在decoder的每一步关注输入不同部分，关注的程度由查询向量decoder决定。

attention将一系列的长度不固定的输入向量转化为长度固定的一个向量表示。

self–attention

values and **query** come from the same layer

values和query来自encoder同一层，比如在encoder的第9个时刻，用此时的隐状态向量作为query，去和其他10个向量（包含它自身）做点乘求和，获得一个attention向量。

优点是可以获得一个序列不同时刻彼此之间的关系，能够替换rnn，对时序进行建模。

rnn计算的缺点是只能一个时刻一个时刻顺序地进行计算，很难在时间维度上进行并行计算。但是使用self–attention可以在每个时刻和其他时刻进行self–attention计算，所有时刻可以和其他时刻进行并行计算。

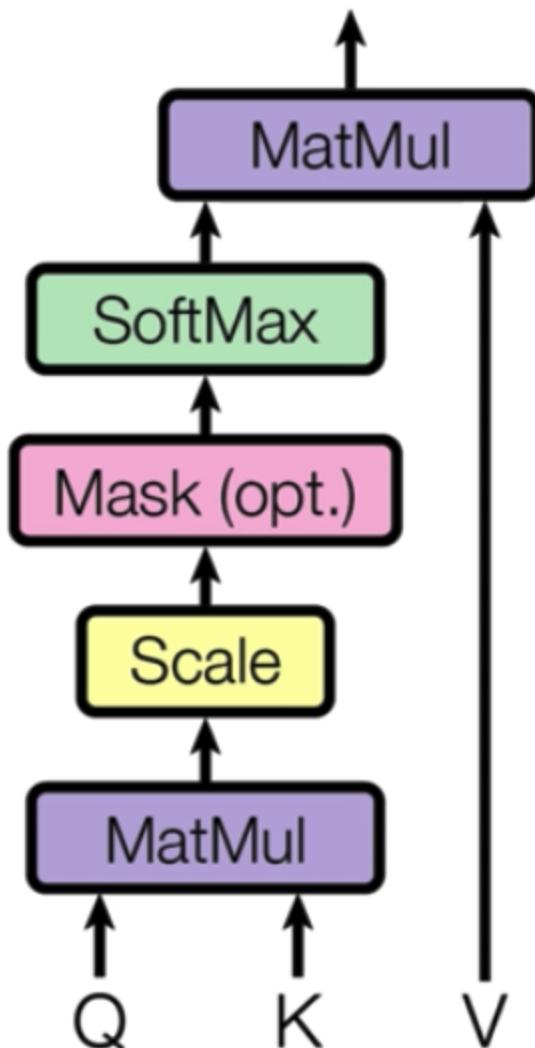
scaled dot–product attention

首先Q和K做一个乘法，然后做一个缩放，mask可选（用于掩盖掉一些值），用softmax求一个attention的概率，然后在V上做一个加权和，得到最终的attention的输出。

K和V一般是同一个向量，当Q, K, V是同一个向量或者矩阵时，即是self–attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention



multi-head attention

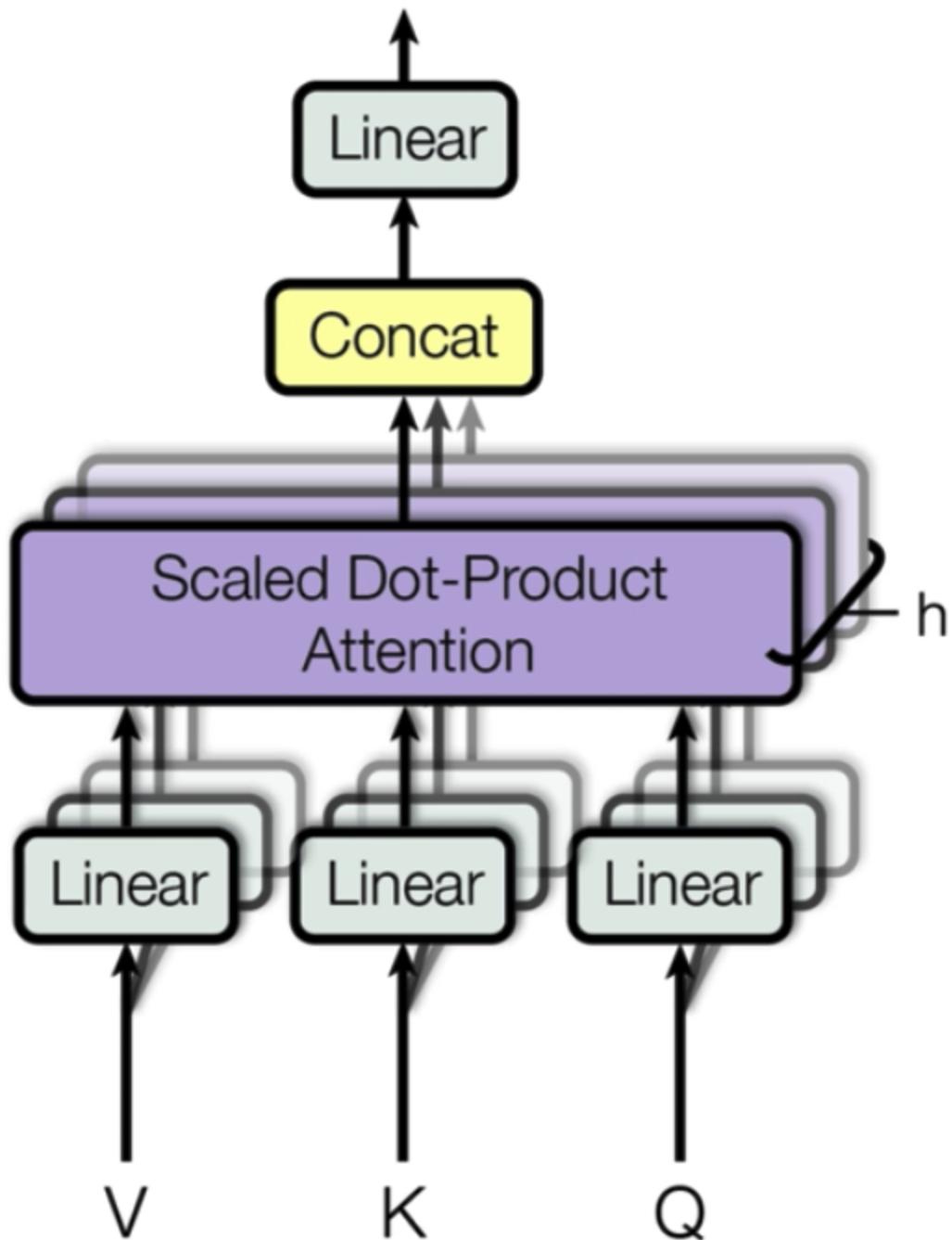
多个attention可以关注不同范围内的信息，有助于提取encoder的信息。

先把Q、K、V通过多个线性层分成多个不同的输入，对每组Q、K、V算一个attention，将多个attention向量拼在一起，经过一个线性层，最后得到一个attention向量。

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Multi-Head Attention



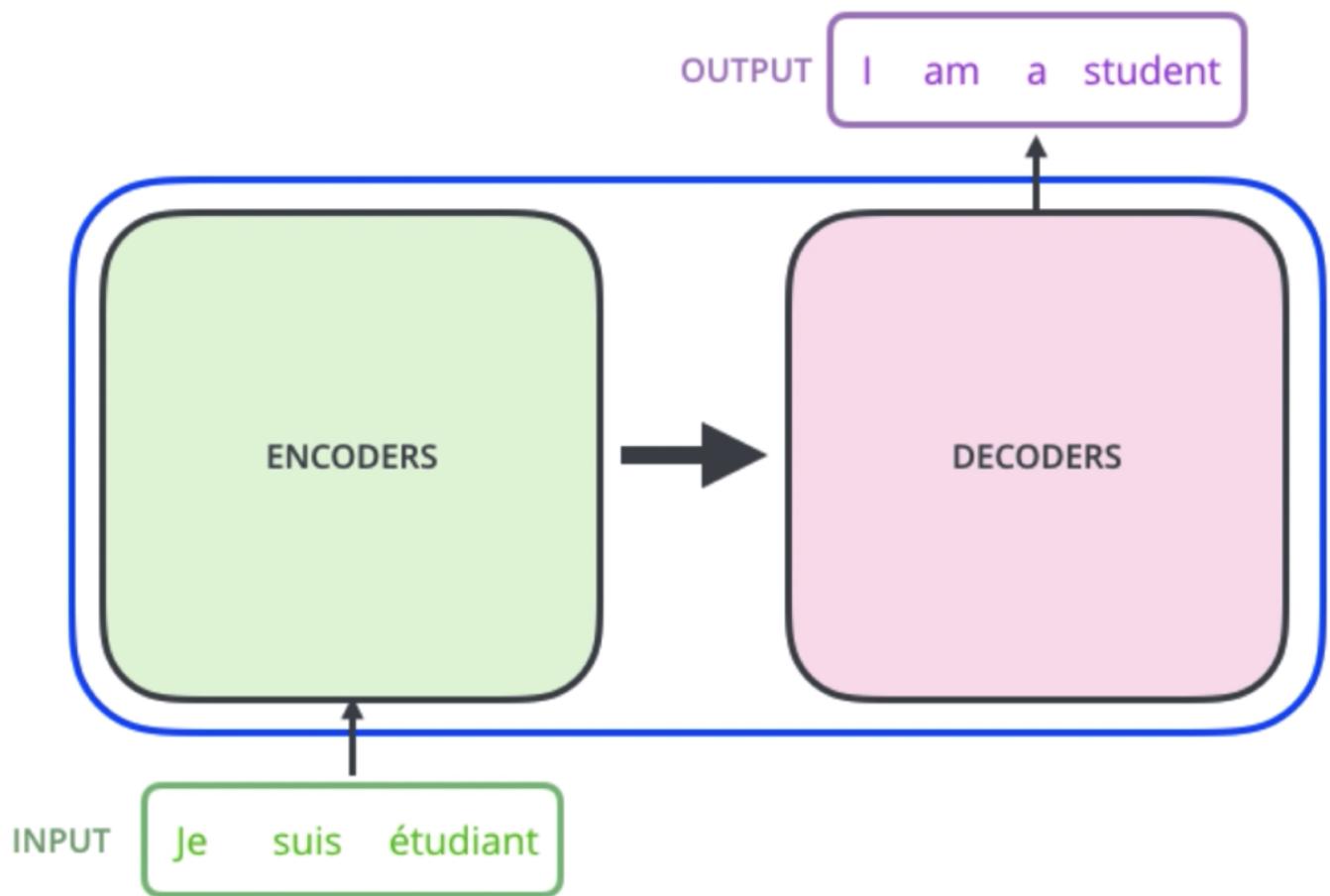
transformer

attention is all you need

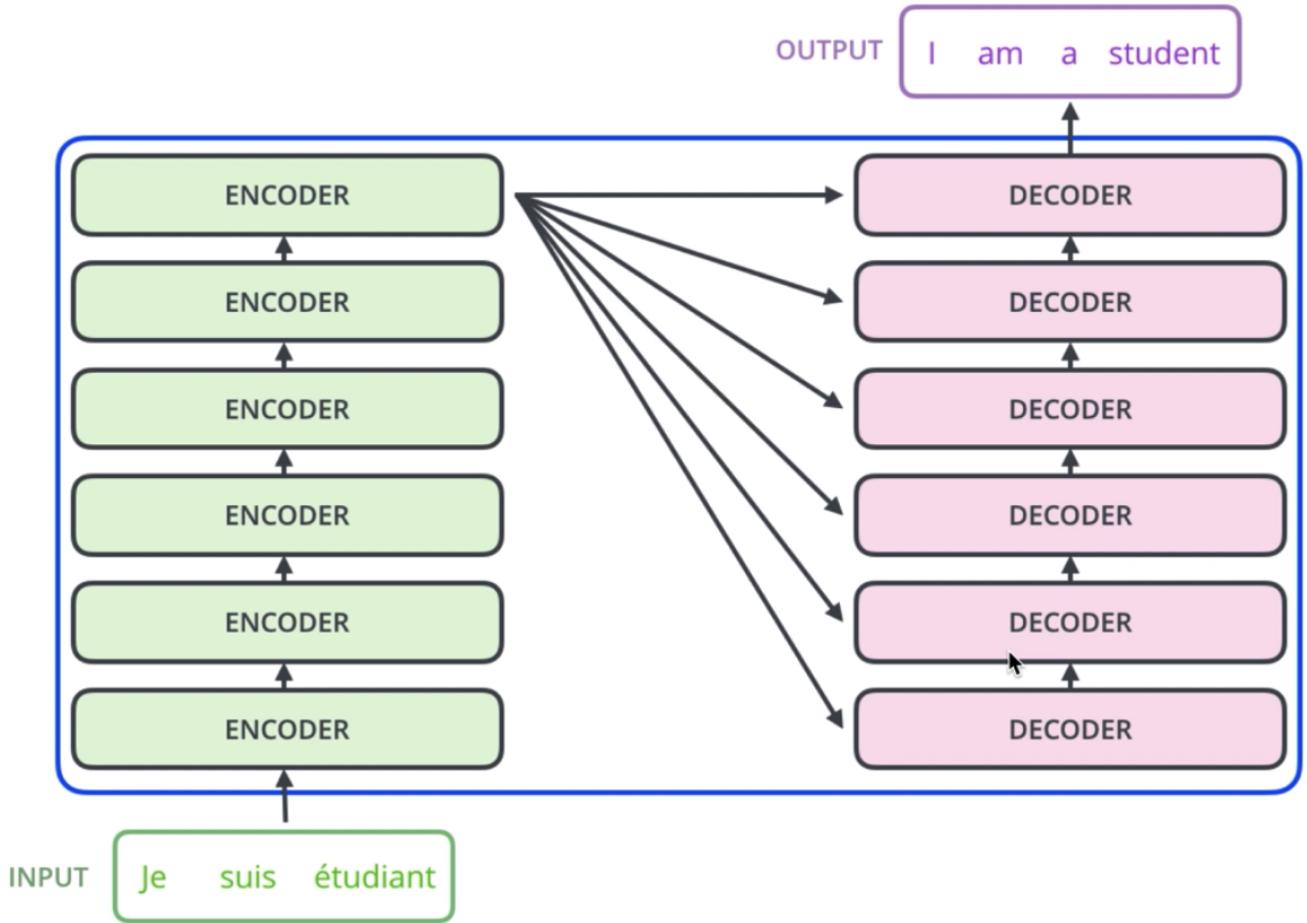
transformer通过一个模型把输入映射成输出。



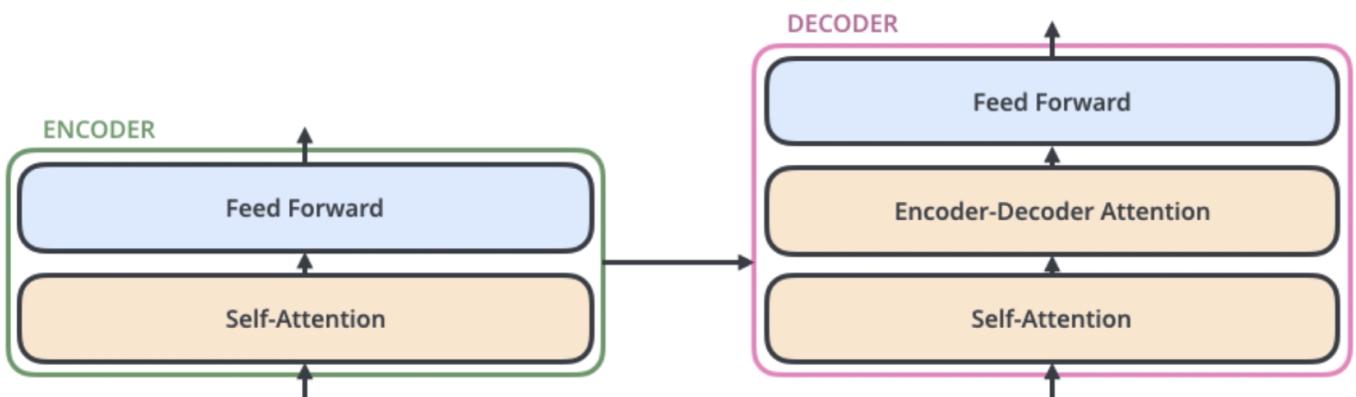
transformer也包含一个encoders和decoders的部分



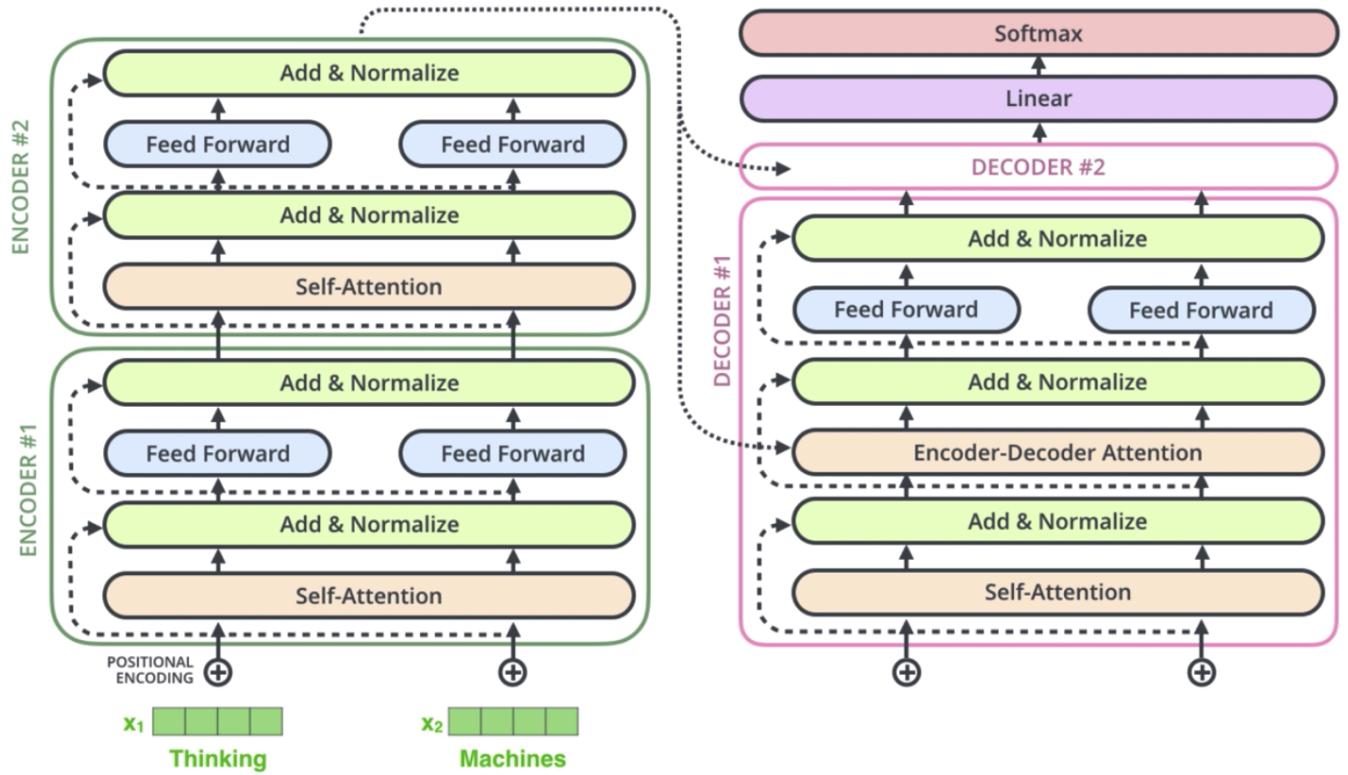
encoders部分包含多层encoder, decoders部分包含多层decoder, 其中一个encoder就是一个rnn, 一个decoder也是一个rnn。



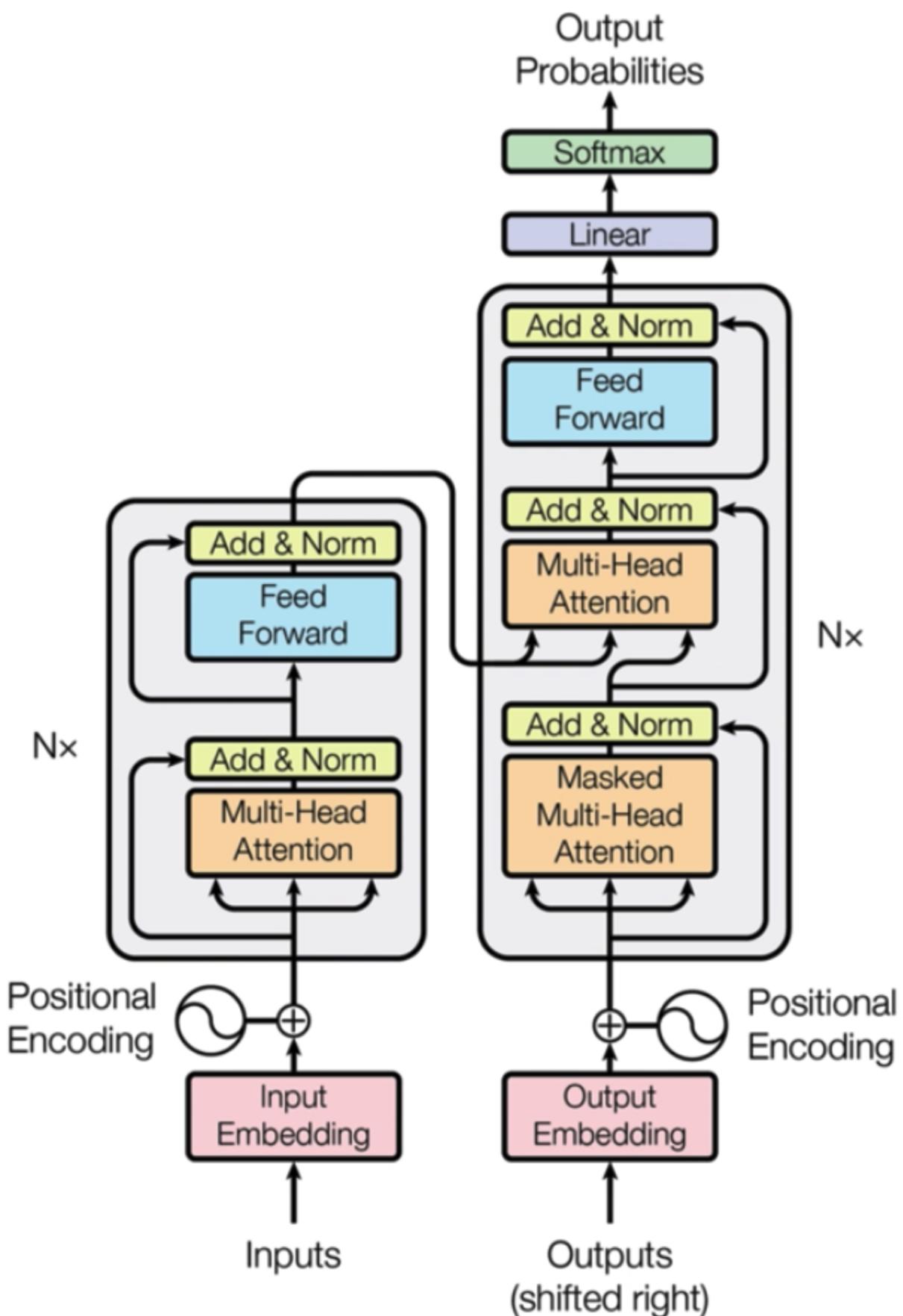
将每个encoder和decoder提取出来，encoder中包含一个multi-head-self-attention和一个dnn，decoder中也包含一个一个multi-head-self-attention和一个dnn，另外包含一个对encoder计算的attention

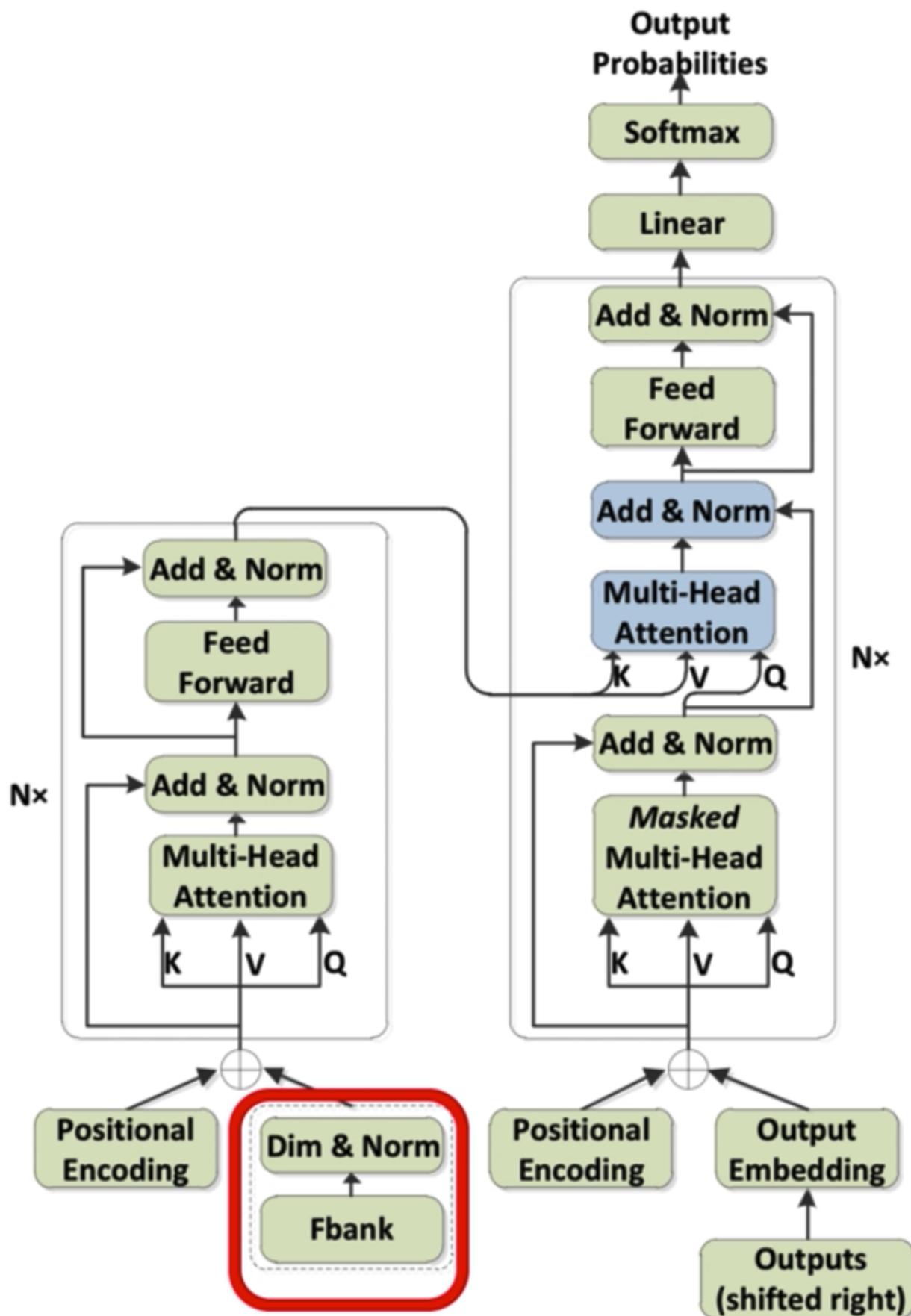


进一步细化，encoder部分包含一个self-attention和一个feed forward，另外将self-attention的输入和输出加载一起，做一个残差连接，在做一个归一化，对feed forward也做同样的操作。



总结，transformer是一个encoder-decoder框架，encoder和decoder都是由多层组成，每一层里面encoder有一个self-attention和一个残差连接，一个feed forward和一个残差连接，decoder类似。输入的时候加一个positional encoding，给输入提供位置信息。





参考论文：

Speech–Transformer: A No–Recurrence Sequence–to–Sequence Model for Speech Recognition
The Speechtransformer for Large–scale Mandarin Chinese Speech Recognition

ctc

- 参考论文

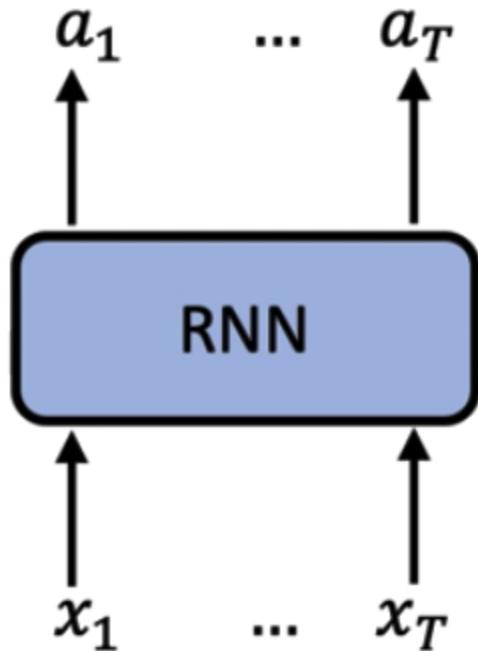
Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks 2016

Towards end-to-end speech recognition with recurrent neural networks 2014

给定语音特征序列 $X = x_1, x_2, \dots, x_T$ 和文本序列 $Y = y_1, y_2, \dots, y_U$ ，T和U一般是不等长的，并且T一般比U要长。ctc通过一个rnn或self–attention等序列模型，将语音特征序列映射到文本序列。

- 问题

使用rnn时，其输出序列A与输入序列X等长， $A = a_1, \dots, a_T$ ，但是文本序列Y一般与特征序列X不等长，也就是 $U! = T$ ，而且通常 $U <= T$ ，该如何用单个rnn将特征序列X映射到文本序列？



- 解决方案

在输出集合中引入表示空的符号blank(ϵ)，对包含 ϵ 的rnn输出序列进行处理，将其变成不包含 ϵ 的正常文本序列

- 合并重复字符
- 移除 ϵ 符号

h h e ε ε | | | ε | | o

First, merge repeat characters.

h e ε | ε | o

Then, remove any ϵ tokens.

h e | | | o

The remaining characters are the output.

h e l l o

- ctc对齐

同一个Y对应多个对齐，对齐具有单调性。引入 ϵ 后，不需要预先知道特征序列X的对齐也可以得到输出序列。

有效的对齐和无效的对齐：

Valid Alignments

$\epsilon \ C \ C \ \epsilon \ a \ t$

Invalid Alignments

C ϵ C ϵ a t

corresponds to
 $Y = [c, c, a, t]$

c c a a t t

c c a a t _

has length 5

c a ϵ ϵ ϵ t

c ϵ ϵ ϵ | t t

missing the 'a'

• ctc定义

- 已知训练集D，包含多对语音特征序列X和文本序列Y
- 解码可以表示成 $Y^* = argmax_y P(Y|X)$
- 训练时最小化的loss可以表示成 $L = \sum_{(X,Y) \in D} -argmax_y P(Y|X)$
- 假设rnn的输出为 $A = a_1, \dots, a_T$,其中包含 ϵ ，合并重复字符和移除 ϵ 后可得到Y，把每个时刻已知X时的输出连乘在一起 $P(A|X) = \prod_{i=1}^T P(a_i|X)$, 公式成立的前提是ctc rnn的输出彼此无关

- 令 $A(Y)$ 表示一个集合，其中每个元素A是一个序列，在合并A的重复字符和移除 ϵ 后可变为Y，也就是 $A(Y)$ 表示Y对应的对齐序列集合

$$P(Y|X) = \sum_{A \in A(Y)} P(A|X) = \sum_{A \in A(Y)} \prod_{i=1}^T P(a_i|X)$$

- ctc训练

- 已知训练集D，包含多对语音特征序列X和文本序列Y，

$$CTCLoss = L = \sum_{(X,Y) \in D} -\log P(Y|X), \text{ 其中}$$

$$P(Y|X) = \sum_{A \in A(Y)} P(A|X) = \sum_{A \in A(Y)} \prod_{i=1}^T P(a_i|X)$$

- 每个A相当于一条可以得到Y的路径，

$$P(Y|X) = \sum_{A \in A(Y)} P(A|X) = \sum_{A \in A(Y)} \prod_{i=1}^T P(a_i|X) \text{ 相当于对多条可以得到Y的路径进行求和}$$

- 多条路径可能存在相同的前缀，因此 $P(A|X) = \prod_{i=1}^T P(a_i|X)$ 中存在重复计算
- 可以通过前向后向算法，高效的计算L，求出L关于rnn输出的梯度，从而使用SGD更新rnn参数

- ctc解码

- 已知语音特征序列X和rnn模型
- 解码可以表示成 $Y^* = argmax_Y P(Y|X)$

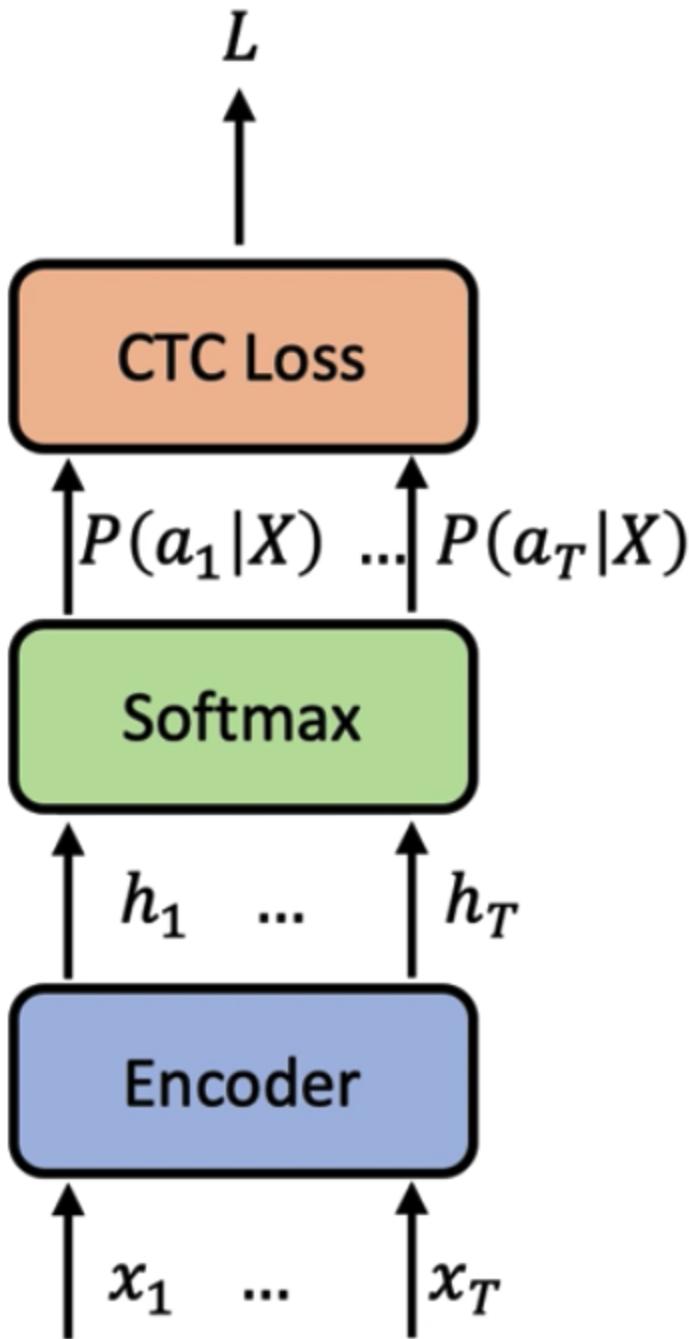
- greedy decode 取每一时刻概率最大的输出 $A^* = argmax_A \prod_{i=1}^T P(a_i|X)$

- 合并 A^* 中的重复和移除 ϵ 后即可得到 Y^*

- beam search参见论文
 - Sequence Modeling with CTC 2017

- 总结

- encoder将输入特征序列转换成隐状态表示，encoder除了可以是rnn(lstm)外，可以是任何可能对时序建模的nn
- softmax将隐状态变成概率，即计算 $P(a_1|X) \dots P(a_T|X)$ ，此时softmax的输出和encoder的输入是等长的
- CTC Loss将和encoder输入等长的概率转换成和输入不等长的Y，ctc loss利用前向后向公式计算Loss并求出关于softmax输出的梯度，用于网络更新

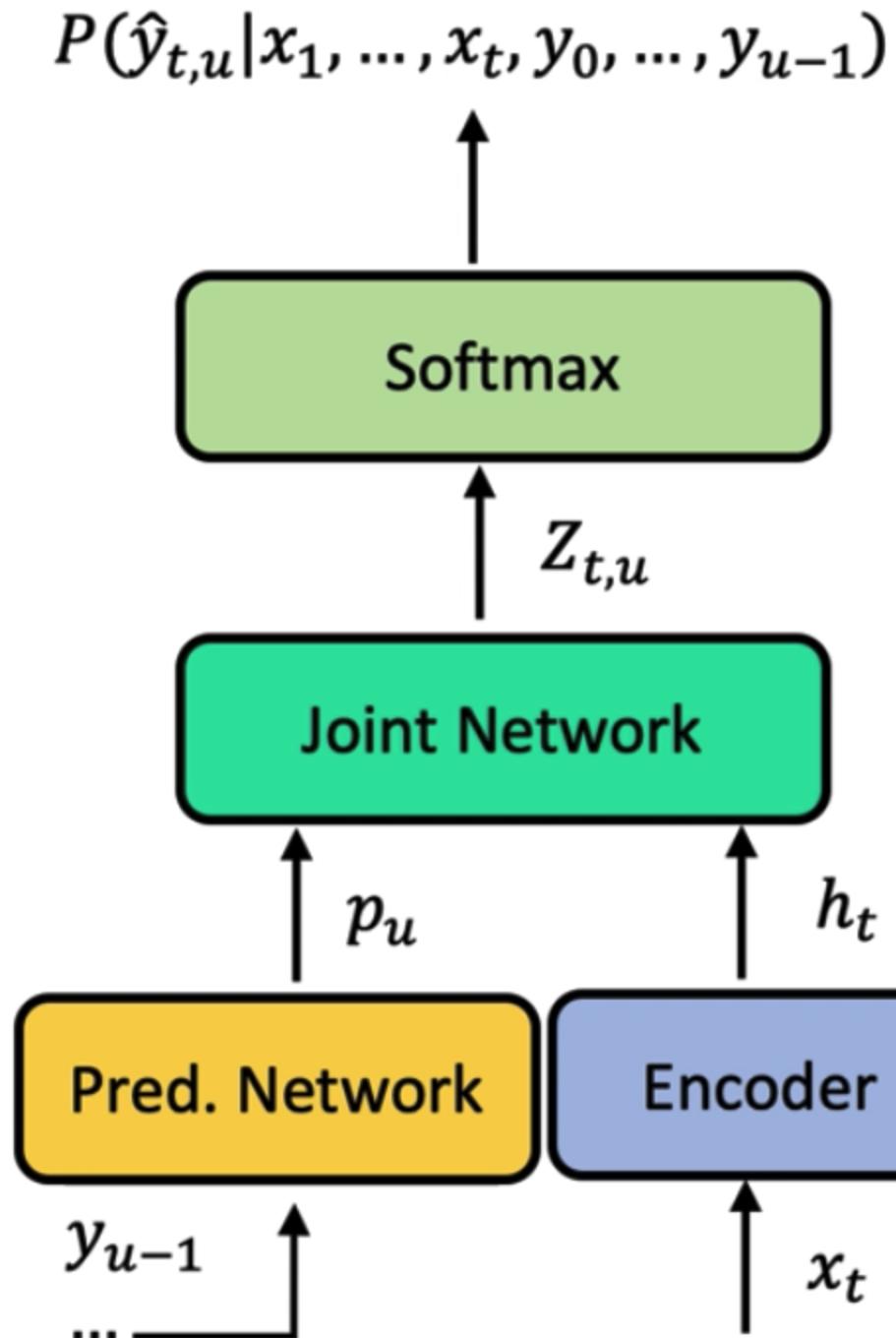


- 优点是引入blank解决输入输出不等长的问题，不需要进行对齐，给定语音特征X和文本Y即可直接进行网络训练，对齐具有单调性
- 缺点是假设不同时刻的输出彼此独立，但实际输出有依赖关系，取得好的性能需要引入额外的语言模型，用于对输出之间的依赖进行建模
- connectionist temporal classification指的就是用rnn进行sequence labelling

rnn transducer

- encoder

- 和ctc一样，可以是任何rnn类的网络
 - 将声学特征 $X = x_1, x_2, \dots, x_T$ 映射到隐向量 $h^{enc} = h_1, \dots, h_T$
 - 作用相当于声学模型一样，提取输入特征信息
- prediction network
 - 对输出的依赖关系进行建模，可以是任何单向rnn类网络
 - 把前一时刻的rnn-t输出 y_{u-1} 作为输入，预测下一个输出 p_u ， p_u 可以视为label，是对语言的依赖关系进行建模
 - 作用相当于语言模型一样
- joint network
 - 用mlp把encoder和prediction network的输出结合在一起，预测最终的网络输出
 - 把声学信息和语言信息集合在一起
- softmax
 - 计算输出概率
 - 训练是计算rnn-t loss，和ctc loss类似，用前向后向算法进行计算



- 优点

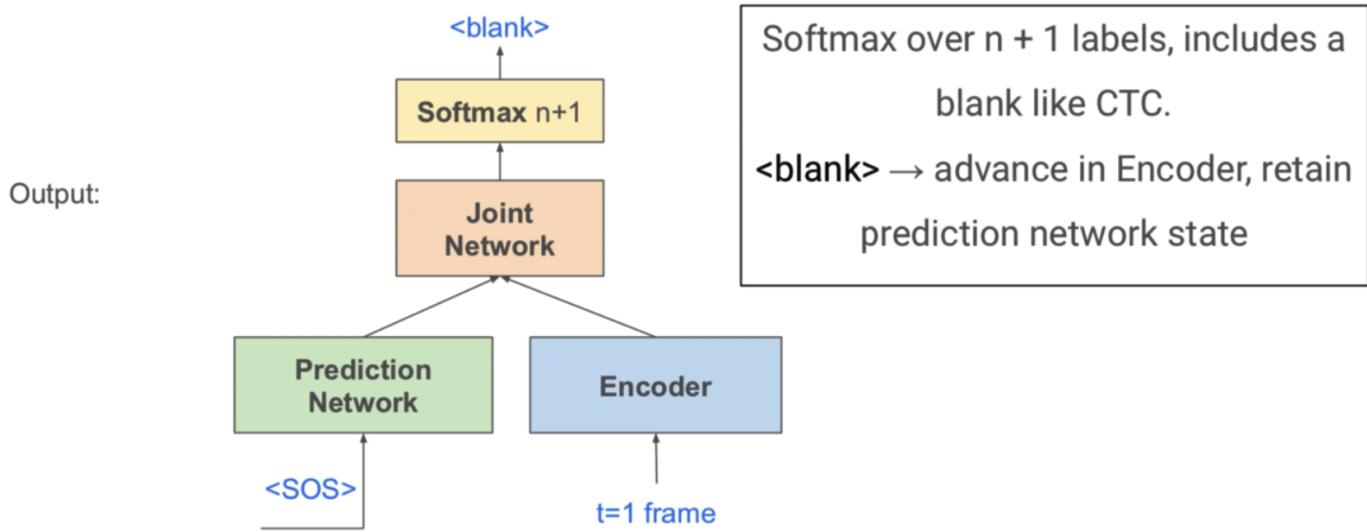
- 使用单向encoder时可以进行流式语音识别，也就是拿到一帧就可以解码一帧
- las、transformer等必须等待一句话说完才可以解码
- 论文 Streaming end-to-end speech recognition for mobile devices

- 流式识别解释

首先假设softmax在 $n+1$ 个labels上计算一个概率，其中包括blank，如果最终网络的输出为blank，那么encoder的输入就往前挪一帧，并保持prediction network的状态。

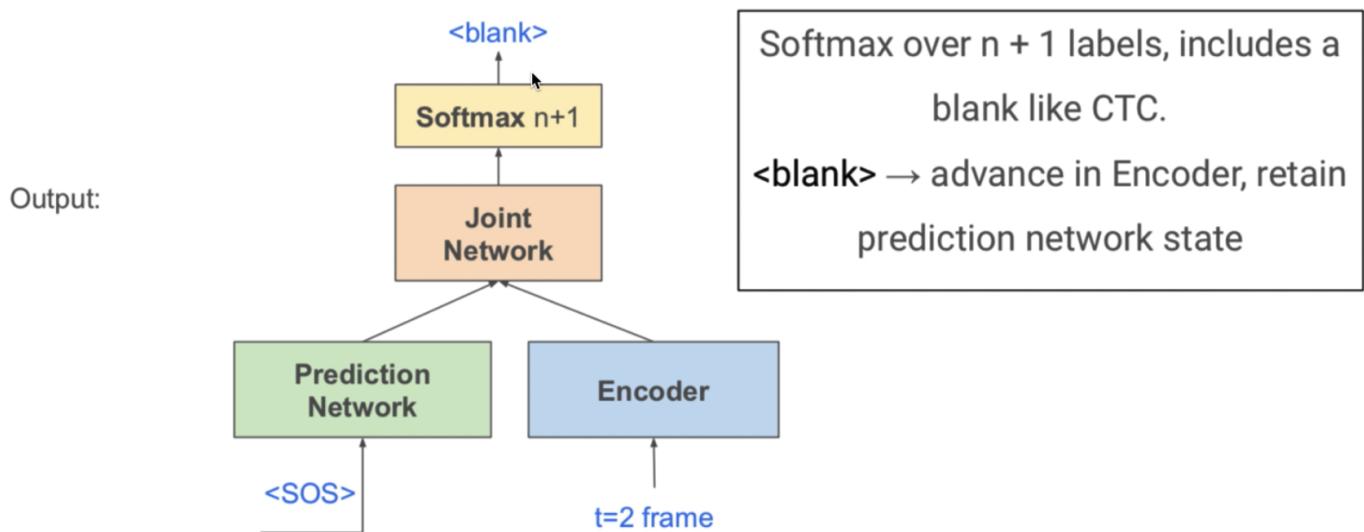
- 第一帧

因为prediction network前一时刻还没有任何输出，我们给它一个<sos>，表示start of sentence。如果此时网络最终的输出为blank，那么encoder输入往前走一帧，保持prediction network输入不变



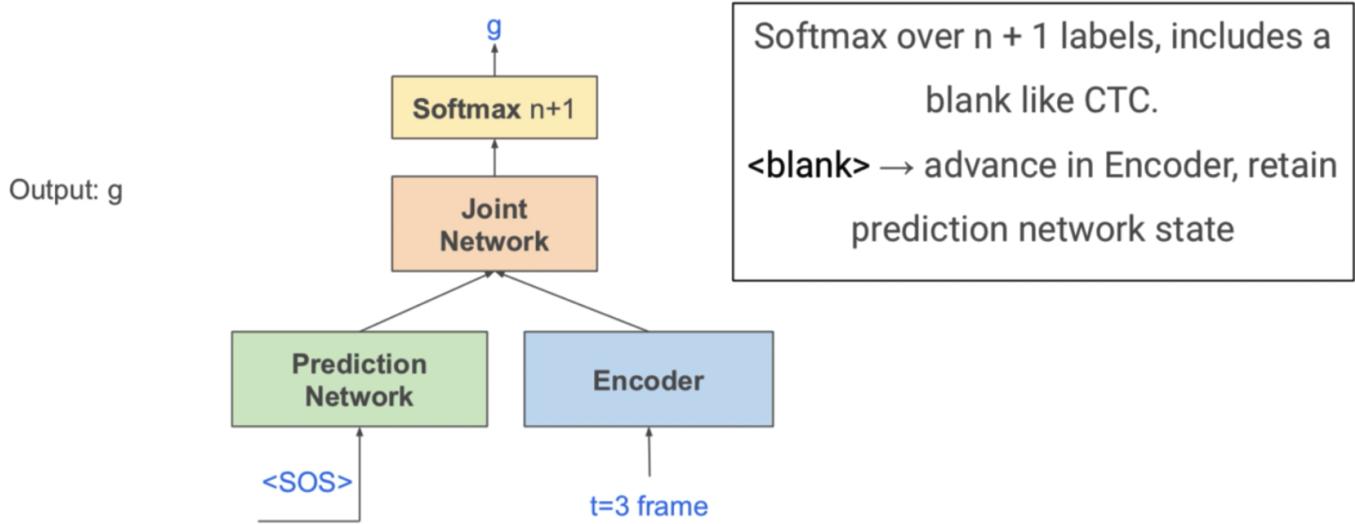
- 第二帧

输出还是blank，prediction network输入保持不变，encoder输入往前走一帧



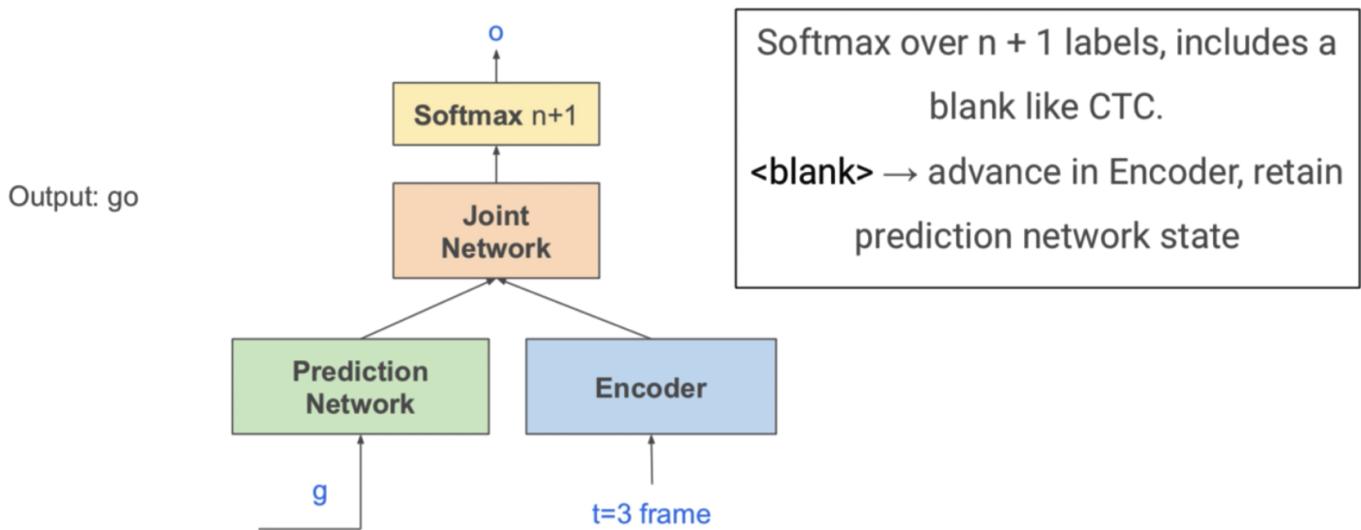
- 第三帧

此时网络输出为g，下一时刻将把g作为prediction network的输入



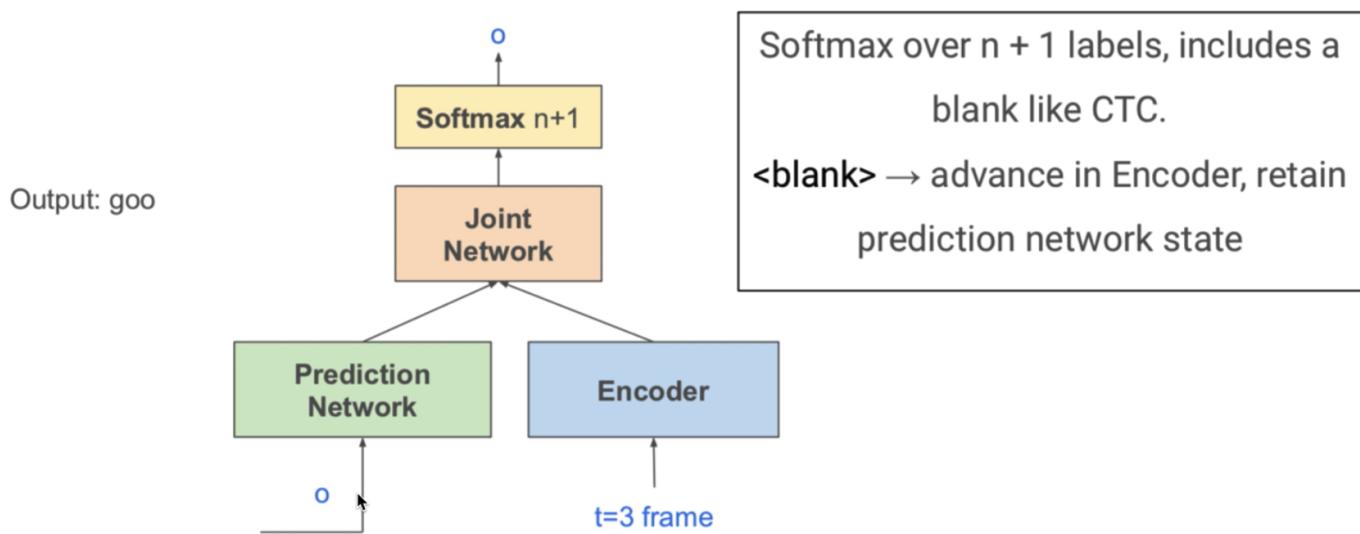
- 第四帧

g作为prediction network的输入，encoder的输入可以更新也可以不更新，但是一般会选择更新。因为一般来说一帧对应一个输出，少数情况下这一帧切换在两个音的分界处。此时输出为o



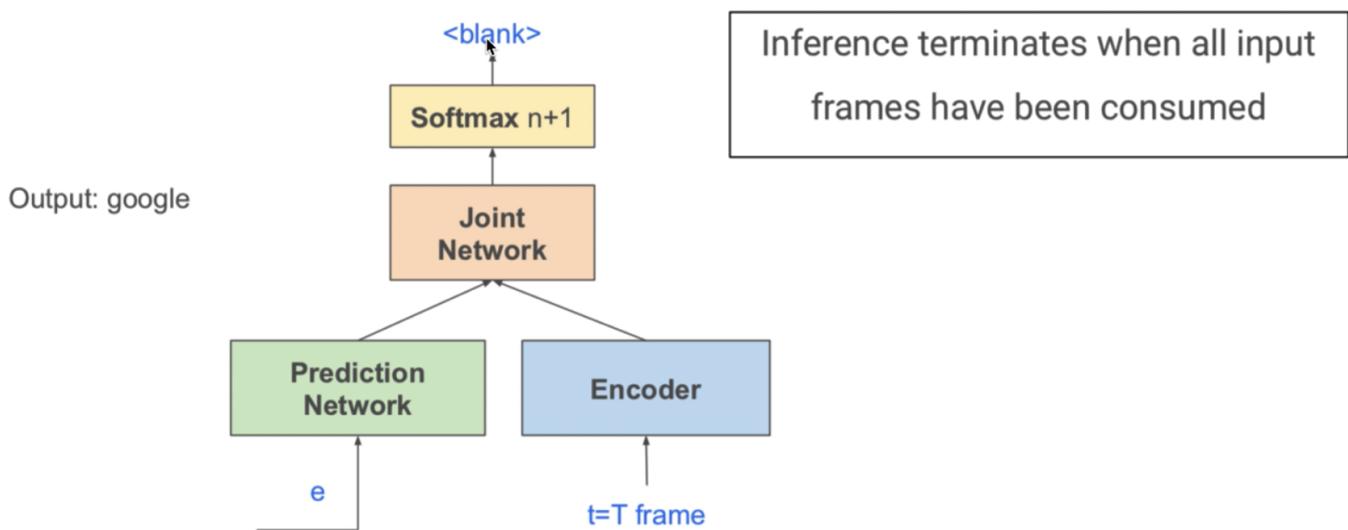
- 第五帧

o作为prediction network的输入，encoder的输入没变，继续预测输出o



- 最后一帧

依次类推，直到最后时刻T，预测输出blank



- 总结

流式体现在，来一帧就可以解码一次，当帧完了，解码就完成了。

开源实现

- espnet [连接](#)
- speech transformer [链接](#)
- las [链接](#)

总结

1. end2end识别动机是简化asr的流程
2. las的优点是没有输出独立性假设，缺点是无法支持流式识别，rnn难以并行训练
3. speech transformer的优点是self–attention易并行，训练速度快，效果好，缺点是有时对超参数很敏感
4. ctc优点是输入和输出具有单调关系，缺点是输出独立性假设，好性能需要LM
5. rnn–transducer优点是天然支持流式识别，缺点是Loss计算不易实现，一般使用开源库