

翻译系统的评价

自动化指标:

- ❖ BLEU (Bilingual Evaluation Understudy): 文本生成任务的常用指标. 有一个机器翻译结果 s , 一个或多个人工翻译例句集合 Ref ,
 - BLEU- n = s 中的 n 元语法在至少一个人工句中出现的个数 c_1 / s 中所有的 n 元语法个数。
 - 基于precision的评价标准, 从预测结果的角度来看待。
 - 需要给过短的机器翻译句一个惩罚因子; n 不宜太小

例子:

s = 'today is a nice day' 2-gram=? { today is ✓
is a
a nice ✓
nice day ✓

Ref = {'today is fine', 'it is such a nice day today'}

BLEU-2 = 3/4

问题: BLEU-1, s = 'today today today today today'

翻译系统的评价

自动化指标:

- ❖ Rouge (Recall-Oriented Understudy for Gisting Evaluation)

$$\text{Rouge -n} = \max_{r \in \text{Ref}} \frac{\text{count}(\text{common } n\text{-gram}) \text{ of } r \text{ and } s}{\text{count}(n\text{-gram}) \text{ in } r}$$

- ✓ 基于recall，从人工参考句的角度来看待

$s = \text{'today is a nice day'}$

$\text{Ref} = \{\text{'today is fine'}, \text{'it is such a nice day today'}\}$

$\text{Rouge-2} = \max(1/2, 2/6)=0.5$

- ✓ ROUGE-L: 基于最大公共子序列lcs

翻译系统的评价

自动化指标:

- ✓ **Meteor**: 基于BLEU和ROUGE的调和平均值, ROUGE权重更大; 对于同义词有一个自动匹配, 可以包容同义词
- ✓ **CIDEr (Consensus-based Image Description Evaluation)**: 使用tf-idf计算了不同n元语法的权重, 减少了常见n元语法的权重。然后通过余弦距离计算机器翻译和参考句的向量相似度
- ✓ **困惑度 (Perplexity, PPL)**: 交叉熵的指数形式。也可以作为模型训练时候的loss
- ✓ 以上标准也可以用于评估语言模型、任意的文本生成任务

翻译系统的评价

自动评价的问题？

可能有多种合理的结果，并且和参考句不吻合
例如，汉译英：我妈妈每天花一小时打扫，

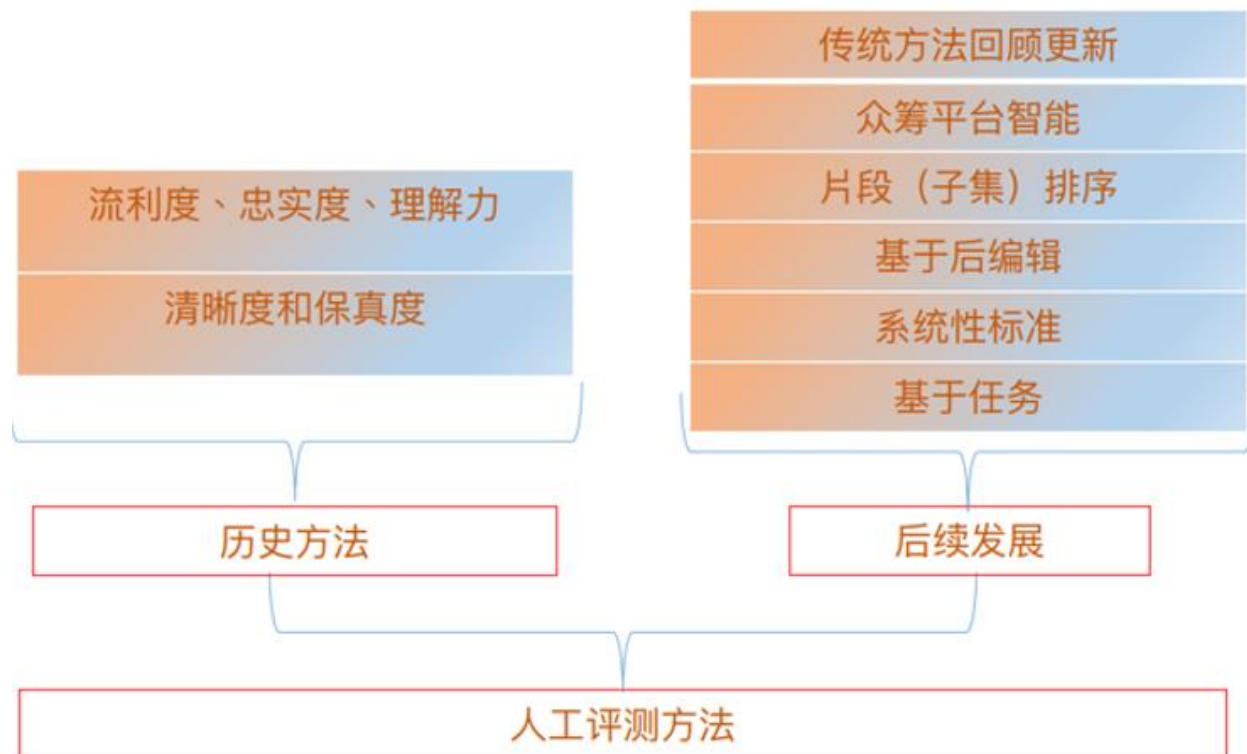
参考翻译：It takes one hour for my mom to clean up everyday

百度翻译：My mother spends an hour cleaning every day

其他文本生成任务里
可能更严重！

❖ 人工评价标准

- 评价人员的培训
- 多人评价、一致性计算
- 标准的合理性



神经机器翻译的优势

- 翻译效果更好

- 更流利：由语言模型带来，和上文具有良好的承接关系
- 更好地利用输入：编码输入句，传给解码阶段
- 更好地泛化短语，更好地根据前面已经学到的短语翻译现在看到的短语

- 神经机器翻译是一个端到端优化的模型

- 编码器和解码器作为一个整体进行前向传播、反向更新，训练更容易

- 更少的人类劳动

- 神经机器翻译把学习都交给语料和模型，不需要特征工程，简洁便利
- 神经机器翻译学习的方法对所有的语言都是通用的：只要成对的平行语料

神经机器翻译的优势

神经机器翻译是深度学习在NLP中的一项巨大成功

- 2014: 第一篇seq2seq论文发表
- 2016: 谷歌翻译系统从统计模型迁移到了神经翻译模型

宝玉早已看见多了一个姊妹，便料定是林姑妈之女，忙来作揖。厮见毕归坐，细看形容，与众各别：两弯似蹙非蹙罥烟眉，一双似喜非喜含情目。态生两靥之愁，娇袭一身之病。泪光点点，娇喘微微。闲静时如姣花照水，行动处似弱柳扶风。心较比干多一窍，病如西子胜三分。宝玉看罢，因笑道：“这个妹妹我曾见过的。”

×

Bǎoyù zǎoyǐ kànjiàn duōle yīgè zǐmèi, biàn liàodìng shì lín gūmā zhī nǚ, máng lái zuòyī. Sī jiàn bì guī zuò, xì kàn xíngróng, yǔ zhòng gè bié: Liǎng wān shì cù fēi cù juàn yān méi, yīshuāng shì xǐ fēi xǐ

晴二五

Baoyu had already seen one more sister, so she guessed that it must be Aunt Lin's daughter, so she hurriedly came to bow to her. After seeing him, he returned to his seat, looked at the description carefully, and found that he was different from the others: two eyebrows that seemed to be frowning but not frowning, and a pair of eyes that seemed to be happy but not happy. Two dimples are born in the state of worry, and the disease of being delicate is attacked all over the body. Tears dripped, panting slightly. When you are quiet, you are like a beautiful flower shining in the water, and when you are in action, you are like a weak willow supporting the wind. The heart has one more opening than the dry, and the disease is like Xizi's three points. After Baoyu looked at it, he said with a smile, "I have seen this sister before."

☆

神经机器翻译的劣势

与统计机器翻译相比:

(以下讨论不仅局限于**MT**，可能存在于所有基于神经网络的任务模型中)

❖ 神经机器翻译的可解释性较差

- 神经网络是一个“黑盒模型”，翻译过程不需要知道内部具体如何运作，**e.g.** 中间的神经元怎么计算的，参数是多少，为什么能得到这个输出？因此调试更为困难。

❖ 神经机器翻译的可控性较差

- 难以
- 可能



Open in Google Translate

Feedback

这是一段赞美真主安拉至高无上的话语。而“is”不由得使人联想起近年来在中东作乱的恐怖组织——“ISIS”（伊斯兰国）。

性

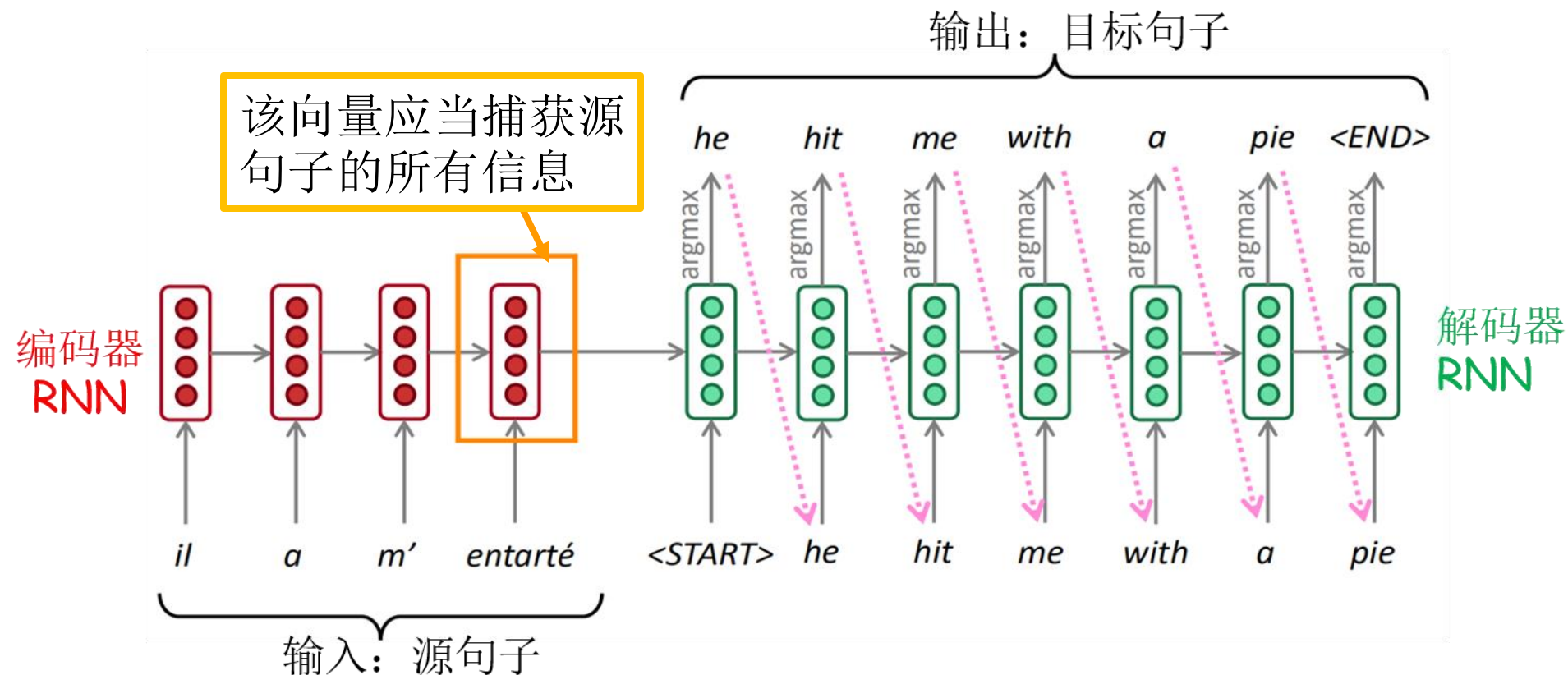
神经机器翻译的难点

- 未登陆词(**Out-of-vocabulary words**): 词表中没有包含的词语, 包括生僻词、新词、网络词。可以考虑进行拆解, 如拆为字母、字、连字符拆分。另外, 还有产生时希望的目标词不在词表中。
- 领域的不匹配: 训练语料和测试语料在内容、用词、风格等方面的不一致性
- 上下文的保持困难: 前文和后文的翻译难以保持一致性
- 低资源问题: 语料较匮乏的语言, 训练效果较差

神经机器翻译的热点

- 多模态：借助其他模态信息(视觉、听觉)帮助消歧
- 领域特定的翻译：专业性更强
- 半监督/无监督的翻译：借助预先预训好的模型；进行数据增强（利用已有数据，生成新的训练数据）

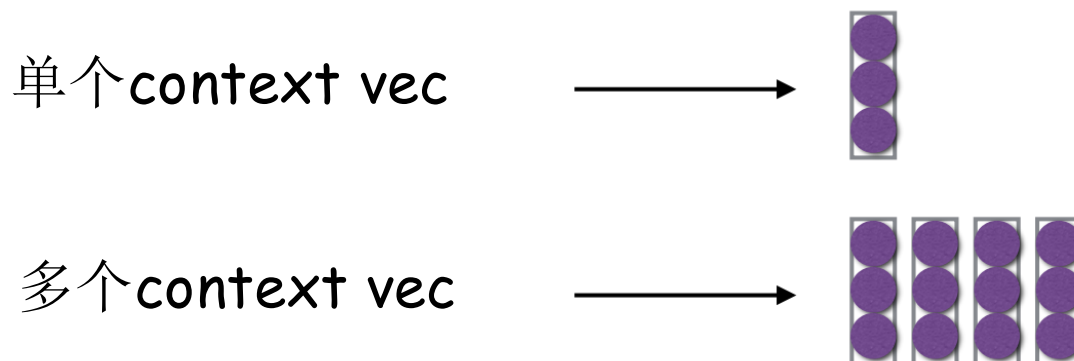
回顾Seq2seq



- **问题：**输入序列经过**encoder**表示为一个低维向量 (**context vector**)，传给**decoder**去产生序列，该向量难以表示输入文本的全部信息 → 瓶颈问题 (**bottleneck**)

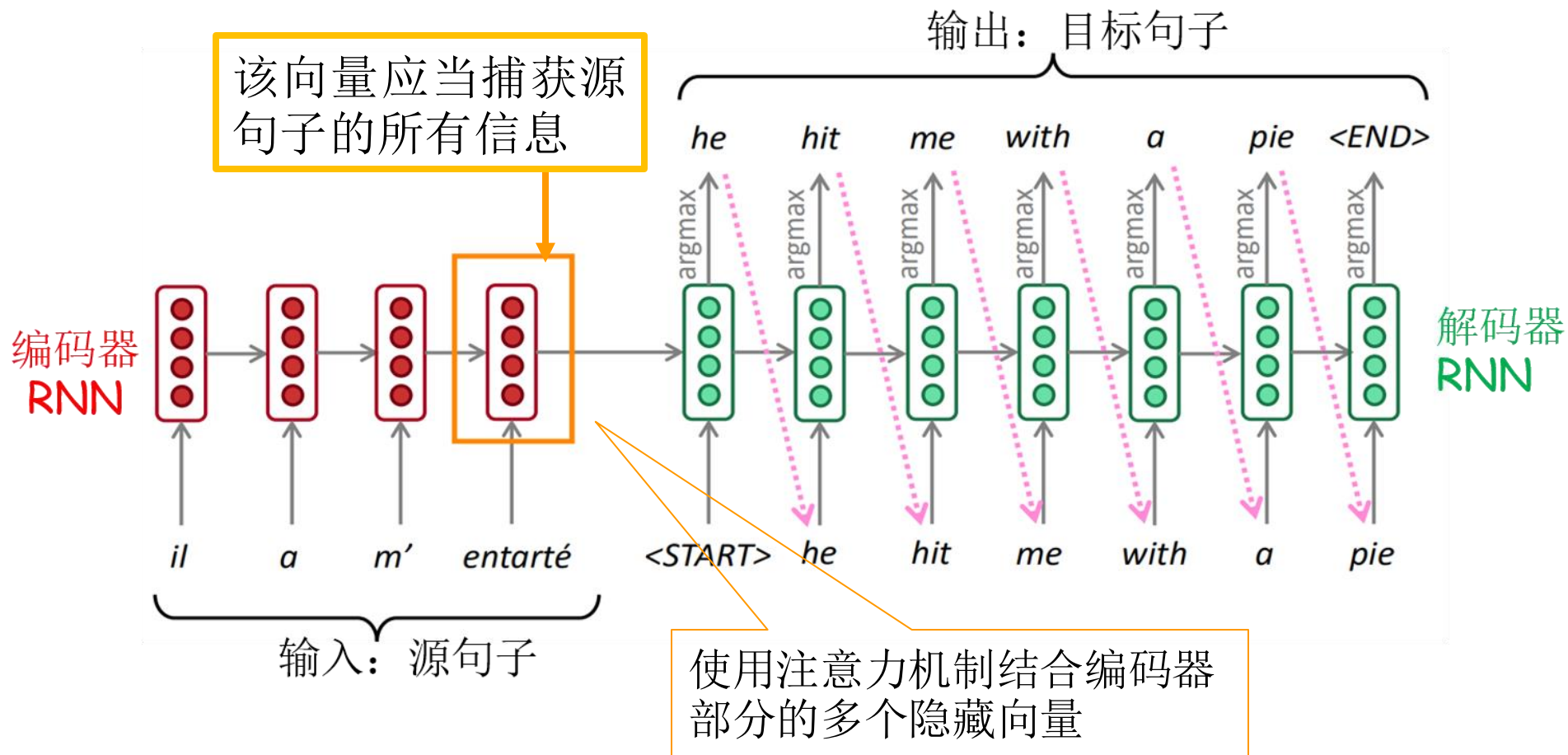
瓶颈问题

- 问题： 基础Encoder-decoder: 序列→单个向量→序列
- 思路： 单个向量换成k个向量，k代表输入文本的长度(token数量)，即每个token的隐藏向量



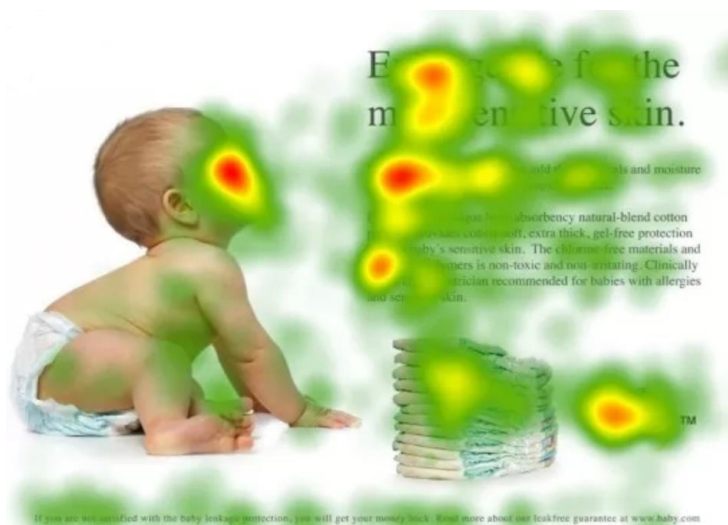
瓶颈问题

使用多个向量一起来表示整个输入句：实现方式即为注意力机制(attention)



注意力机制

- 注意力机制(Attention mechanism)是一种解决信息瓶颈(information bottleneck) 的有效方法



- 在seq2seq中使用注意力机制的核心思想: 在解码的每一步中, 使用与编码器的直接的联系来关注输入句的特定部分

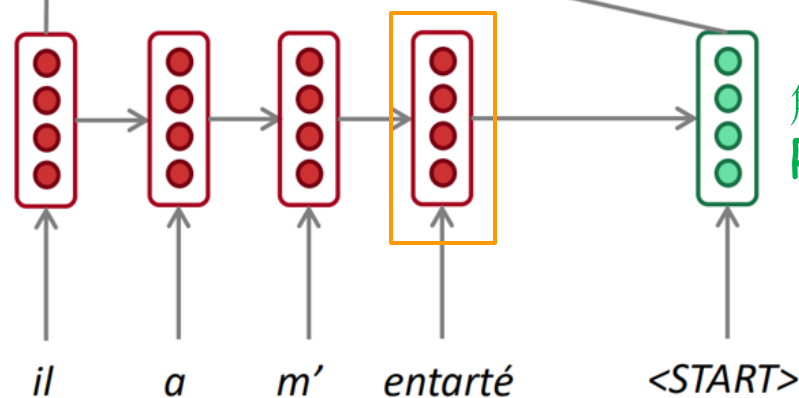
结合注意力的seq2seq

在解码的第一个步骤，使用解码器第一步的隐藏状态（即编码器的**encoding**结果）和编码器的第一个隐藏状态 h_1 进行点积操作，得到一个注意力得分，该得分是一个数值

点乘(dot product)

注意力
分数

编码器
RNN

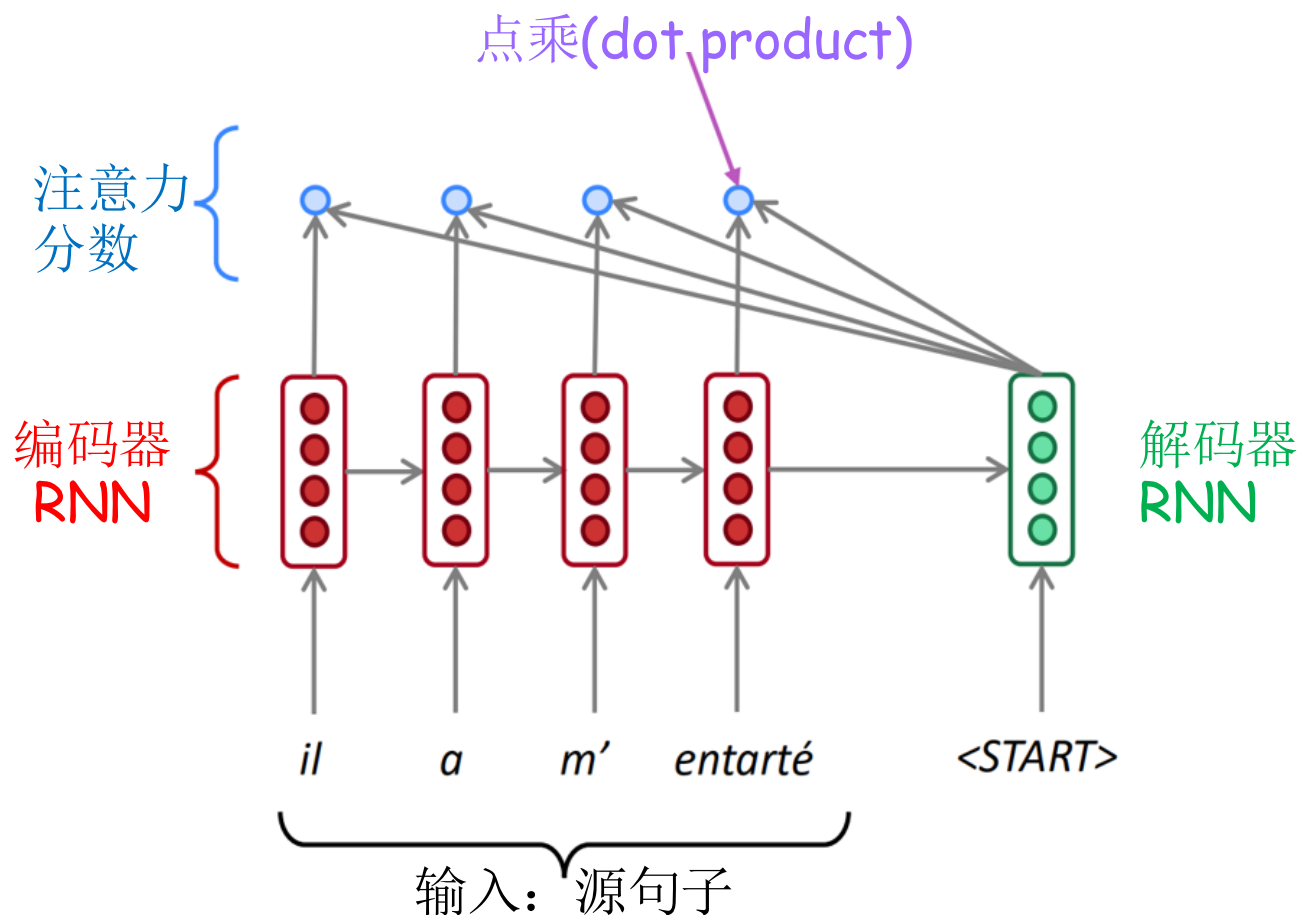


解码器
RNN

点积可以替换成
其他计算方式

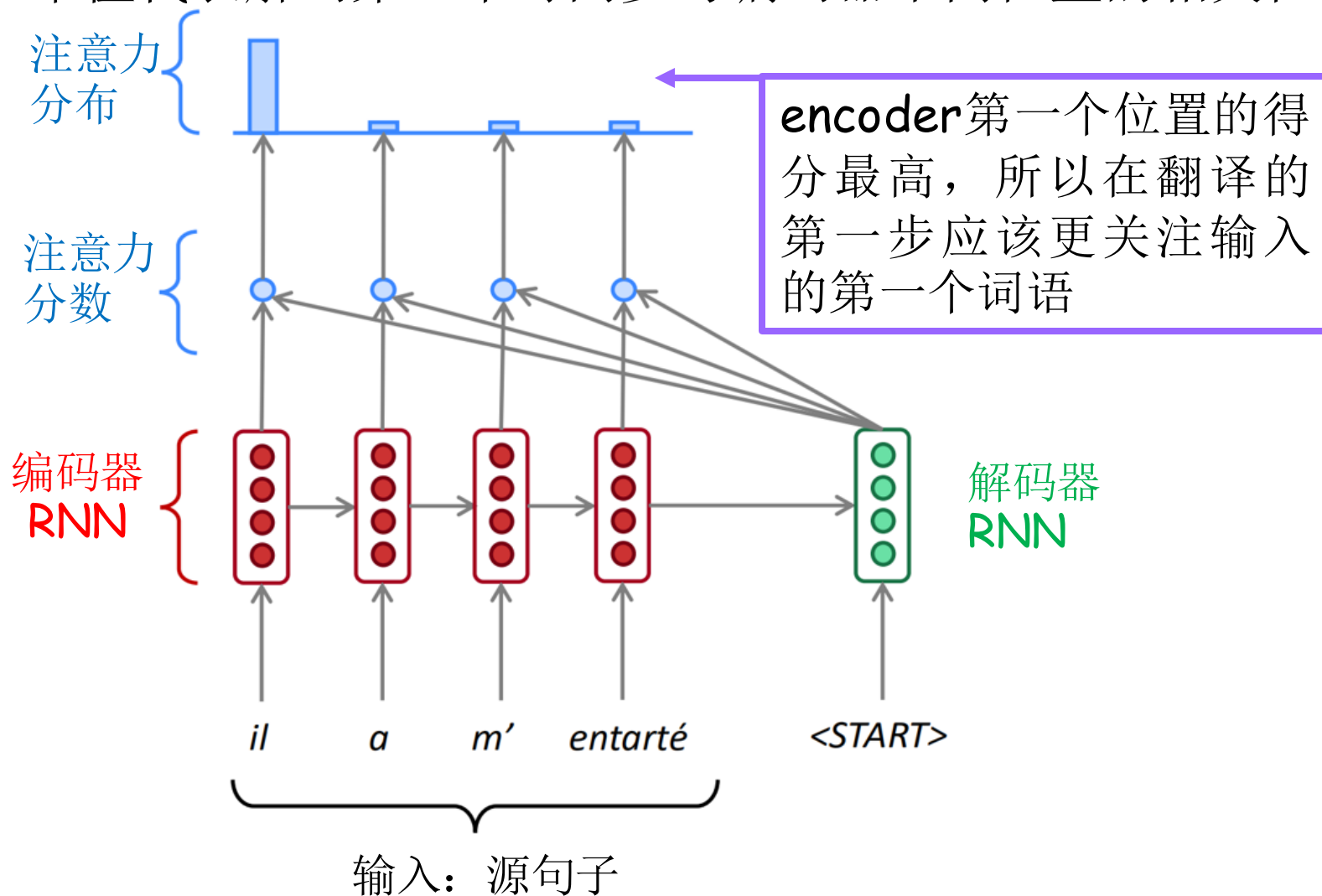
结合注意力的seq2seq

同时，解码器的隐向量和编码器每一个隐向量都计算注意力分数，即在当前的输出时间步计算与每个输入词之间的得分。

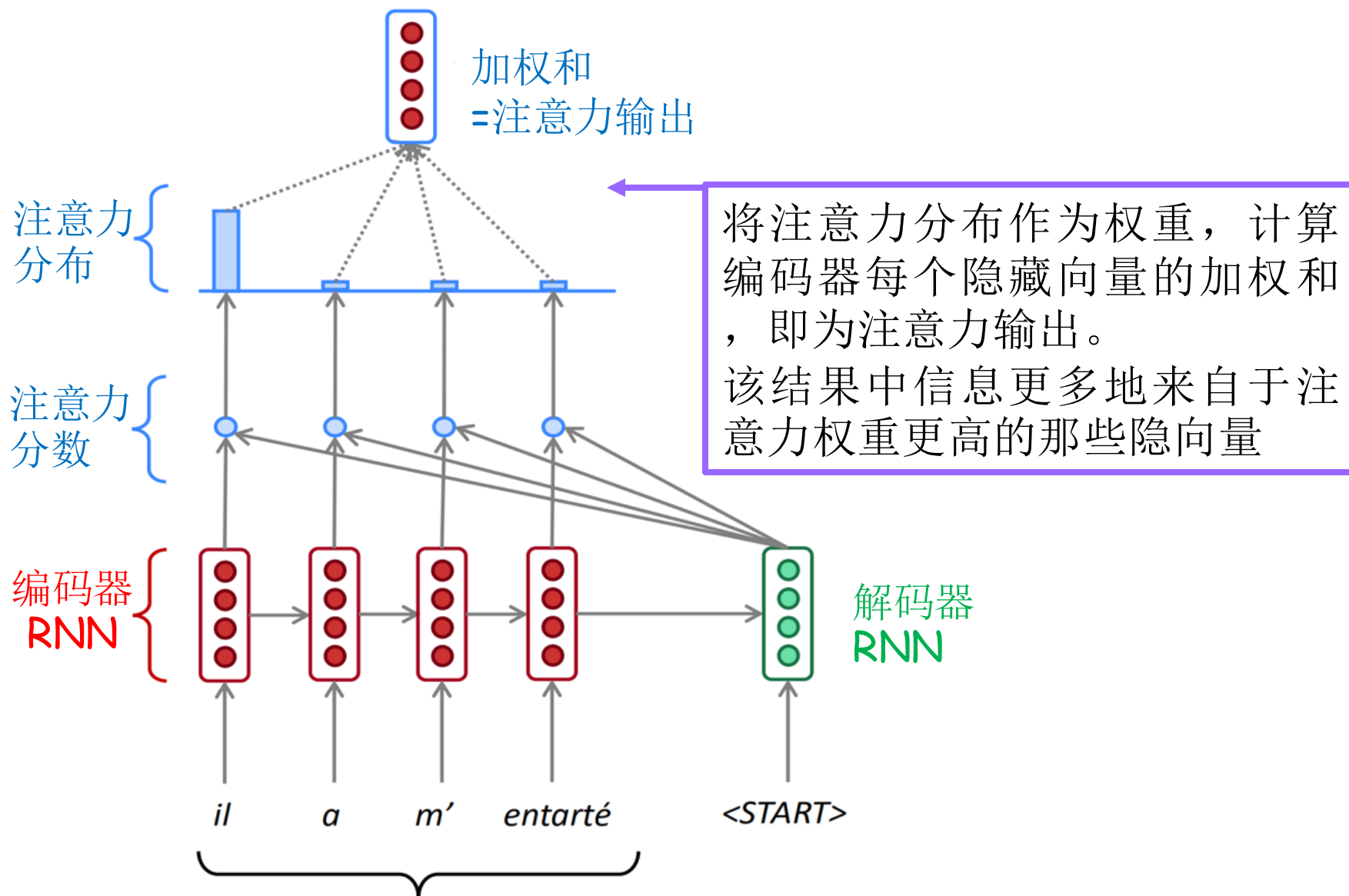


结合注意力的seq2seq

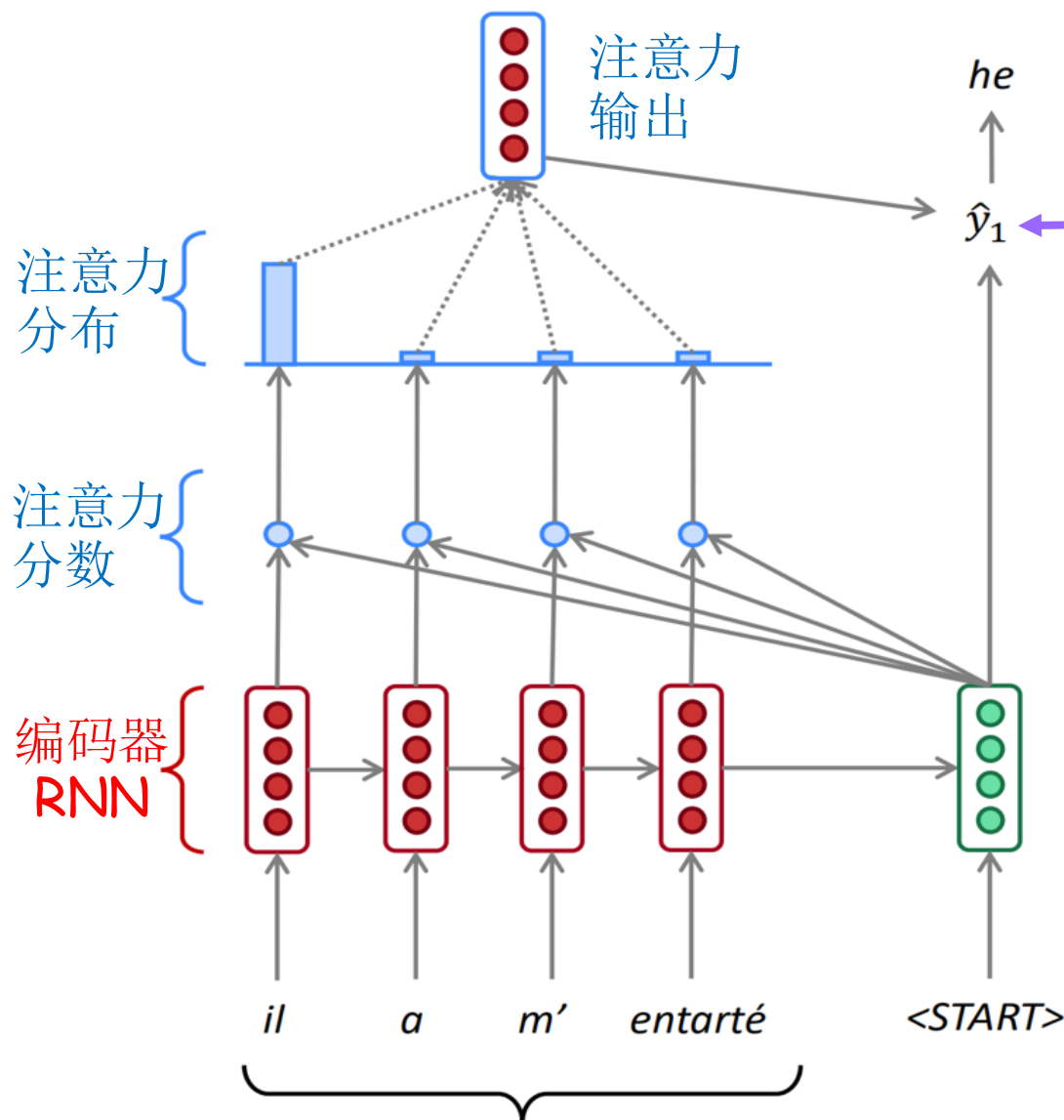
使用**softmax**将当前步骤所有的分数转化为概率分布，具体概率值代表解码第一个时间步与编码器不同位置的相关性程度



结合注意力的seq2seq



结合注意力的seq2seq



将注意力输出和解码器更新后的隐藏状态进行拼接。
基于拼接后的向量计算当前的预测输出 \hat{y}_1

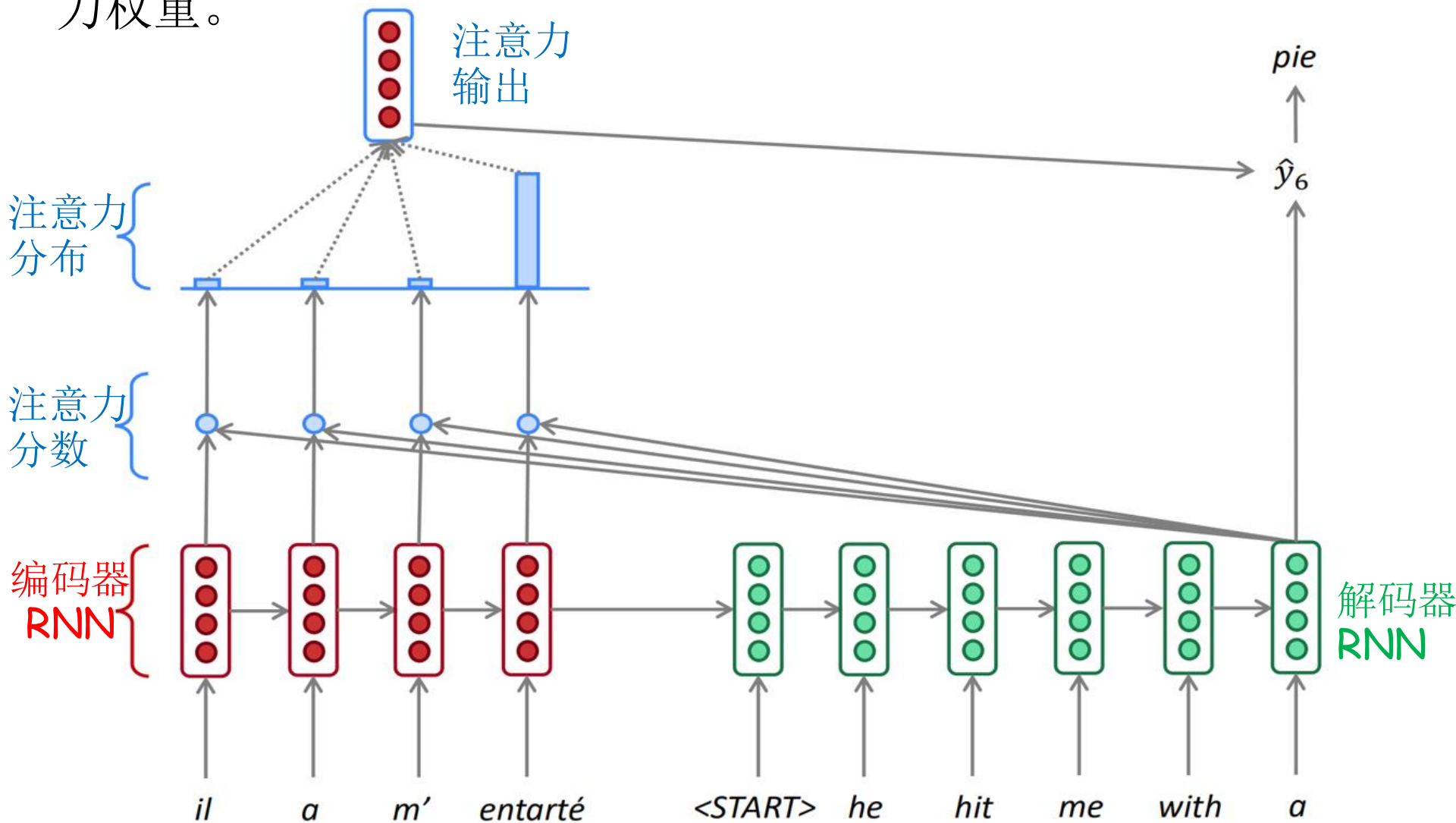
也可以直接使用注意力输出产生 \hat{y}_1

第一步预测的 he 即为第二步的输入，继续进行解码



解码器
RNN

结合注意力的seq2seq

解码的每一步采取同样方式，都要和编码器每个位置计算注意力权重。



注意力计算

- 将编码器序列表示为 $h_1, \dots, h_N \in R^d$ 
- 当前的解码器隐藏状态表示为 $s \in R^d$ 
- 注意力机制的大致流程:
 - f 可以有多种实现方式, 点积是最简单的一种
 - 首先计算 s 和每一步 h 的注意力得分, 函数表示为 f , 得分 $e = [f(sh_1), \dots, f(sh_N)] \in R^N$, 有 N 个得分 (N 代表输入长度)
 - 在 e 上添加 softmax , 得注意力概率分布 $\alpha = \text{softmax}(e) \in R^N$, 有 N 个概率值
 - 使用编码器每个时间步的注意力概率作为权重计算加权和:
 $a = \sum_{i=1}^N \alpha_i h_i \in R^d$, 称为注意力输出(**attention output**)。它是解码器当前步骤与编码器每个位置直接交互的结果, 也可以称为上下文向量(**context vector**)

注意力得分计算

在一些文献中经常用 \mathbf{a} 表示注意力计算函数， \mathbf{a} 指的是alignment，是通过模型训练得到的两个向量相关程度的一种软对齐

- 点乘 (Dot Product)

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$$

- 是解码器隐向量 \mathbf{s} 和编码器第 i 个位置的注意力分数，是一个数值
 - 不需要参数，要求两个隐向量维数相同。
 - 问题**：当向量维数较大时，向量点积结果会很大，即 \mathbf{e}_i 很大， softmax 是S形函数，此时 $\text{softmax}(\mathbf{e})$ 对于 \mathbf{e}_i 的梯度接近于0

- 缩放点乘(Scaled Dot Product)

- 使用向量的维度大小来对原始的点积进行缩放，防止 \mathbf{e} 过大

$$\mathbf{e}_i = \frac{\mathbf{s}^T \mathbf{h}_i}{\sqrt{|\mathbf{h}_i|}} \in \mathbb{R}$$

注意力得分计算

- 双线性计算

$$e_i = s^T W h_i \quad \mathbf{s} \text{ 和 } \mathbf{h}_i \text{ 的维数可以不同}$$

- 拼接计算

$$e_i = v^T \tanh(W[h_i: s])$$

- 加法计算

$$e_i = v^T \tanh(W_1 h_i + W_2 s)$$

本质一个前馈神经网络，所以叫做多层感知机注意力 (MLP attention)。需要保证两个相加乘积的维数相同

多层感知机注意力

- 早期工作。又称为 **Bahdanau Attention (2014)**
- 最初解决机器翻译任务
- 解码依据：前一个预测词 y_{i-1} , 当前步骤处理后的隐向量 s_i , 当前步骤的上下文向量 c_i

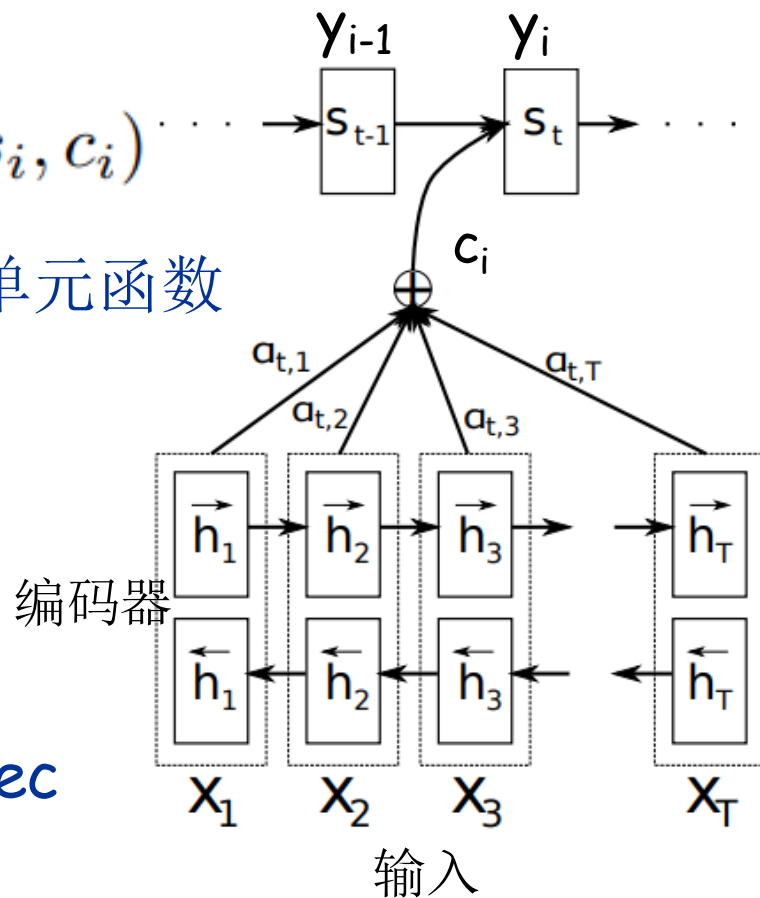
$$p(y_i | y_1, \dots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i) \cdots \rightarrow S_{t-1} \rightarrow S_t \rightarrow \cdots$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \quad \mathbf{f}: \text{循环单元函数}$$

$$\begin{aligned} e_{ij} &= a(s_{i-1}, h_j) \quad \text{加性注意力} \\ &= v_a^\top \tanh(W_a s_{i-1} + U_a h_j) \end{aligned}$$

$$\alpha_{ij} = \text{softmax}(e_{ij})$$

$$c_i = \sum \alpha_{ij} h_j \quad \text{加权和 context vec}$$



示例代码（1） 多层感知机注意力

```
class BahdanauAttention(nn.Module):
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, 1)

    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)

        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights
```

$$e_{ij} = a(s_{i-1}, h_j) \\ = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

$$\alpha_{ij} = \text{softmax}(e_{ij})$$

$$c_i = \sum \alpha_{ij} h_j$$

```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        super(AttnDecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.gru = nn.GRU(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)
```

```
def forward_step(self, input, hidden, encoder_outputs):
    embedded = self.dropout(self.embedding(input))

    query = hidden.permute(1, 0, 2)
    context, attn_weights = self.attention(query, encoder_outputs)
    input_gru = torch.cat((embedded, context), dim=2)

    output, hidden = self.gru(input_gru, hidden)
    output = self.out(output)

    return output, hidden, attn_weights
```

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

示例代码 (2)

```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)  # 线性层: Wh
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

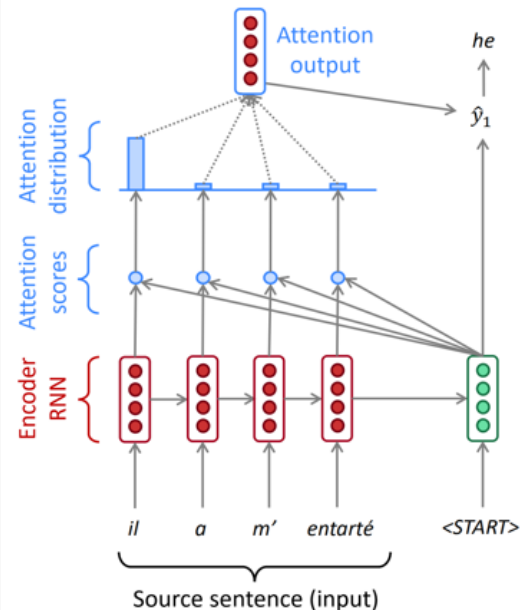
        attn_weights = F.softmax(
            self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                                 encoder_outputs.unsqueeze(0))

        output = torch.cat((embedded[0], attn_applied[0]), 1)
        output = self.attn_combine(output).unsqueeze(0)

        output = F.relu(output)
        output, hidden = self.gru(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```



注意力计算

权重乘→注意力输出

拼接[注意力输出, 解码隐向量]

更新h, 计算预测词

示例代码（2）

```
def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer,
criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
```

编码器

解码器初始化

示例代码（2）

接上页

if use_teacher_forcing: 训练模式

Teacher forcing: Feed the target as the next input

for di in range(target_length):

decoder_output, decoder_hidden, decoder_attention = decoder(
decoder_input, decoder_hidden, encoder_outputs)

loss += criterion(decoder_output, target_tensor[di])

decoder_input = target_tensor[di] *# Teacher forcing*

else: 推理模式

Without teacher forcing: use its own predictions as the next input

for di in range(target_length):

decoder_output, decoder_hidden, decoder_attention = decoder(
decoder_input, decoder_hidden, encoder_outputs)

topv, topi = decoder_output.topk(1) 贪心策略

decoder_input = topi.squeeze().detach() *# detach from history as input*

loss += criterion(decoder_output, target_tensor[di])

if decoder_input.item() == EOS_token: 结束标记
break

loss.backward()

encoder_optimizer.step()

decoder_optimizer.step()

return loss.item() / target_length

注意力的优点

- 显著提升了神经机器翻译的效果
 - 能够在**decoder**阶段关注某些特定的输入词
- 解决了瓶颈问题
 - **decoder**可以直接地和**encoder**每个隐藏状态建立联系，取代单个向量
- 缓解梯度消失问题
 - 因为**decoder**直接和**encoder**隐藏状态联系，不需要信息一步一步进行传输，相当于经历了一个不同状态间的直接的通道，能够减少梯度消失

注意力的优点

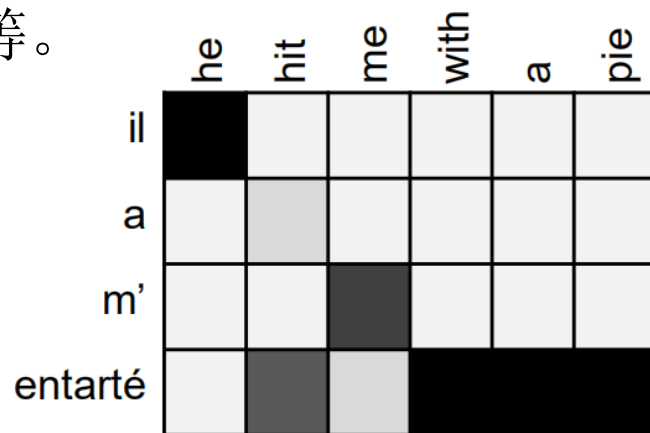
- 提供了一定的可解释性
- 我们可以对**attention** 分布做可视化，就可以看到**decoder**在解码的每一步都重点关注了输入的哪些部分

❖ **Attention**是一种软对齐, 输出和输入词的关系并非选/不选，而是有一个参照多少的程度。

※ 在统计机器翻译里是**hard alignment**, 需要去人为地去学习、定义概念、设计计算模型等。

※ 在结合了**attention**的**seq2seq**里，通过训练就可以得到**attention**矩阵，它是模型训练的一个副产物，并非人为设计，也没有关于**alignment**的标注样本。

attention矩阵完全就是模型自己根据平行语料训练, 然后导出来的一个网络参数。



注意力的意义

- 注意力机制的通用定义：给定一组向量**values**，和一个向量**query**，去基于**query**计算这一组**value**的加权和。
- 注意力机制可以不基于**seq2seq**，如直接结合**CNN**、**RNN**
- 基于**RNN**的**seq2seq**可以看成一种**vanilla seq2seq**，结合注意力机制是优化思路之一
- 由于注意力机制的强大，自身就开创了一种新结构
- **Attention**的加权和理解：对一组**value**中包含的信息的**选择性总结**，选择性体现在一组**value**里的值有不同的选择权重，**query**用来决定要关注哪些**value**
- **Attention**是一种获取任意一组向量(即一组**values**)的固定大小的表示的方法，**任意**体现在一组**values**个数不固定，实际是输入句的长度。这种表示方法依赖于某个其他表示(即**query**)

实验作业

Task1: 选择除机器翻译外的一项生成式任务（越具体越好，如“文学作品的机器翻译”），分析评价适合的指标，特别是任务特定的评价指标

Task2: 机器翻译。采用基于RNN(GRU...)的seq2seq架构，decoder端使用缩放点积注意力和**beam search**解码

Data: WMT 2014 英-德翻译（输入英语，输出德语！）

test.parquet

{ } test.parquet.as.json 1 ×

```
1  {"translation":{"de":"Gutach: Noch mehr Sicherheit für Fußgänger","en":"Gutach: Increased safety for pedestrians"}}
2  {"translation":{"de":"Sie stehen keine 100 Meter voneinander entfernt: Am Dienstag ist in Gutach die neue B 33-Fußgängerampel am Dorfparkplatz in Betrieb genommen worden – in Sichtweite der älteren Rathausampel.","en":"They are not even 100 metres apart: On Tuesday, the new B 33 pedestrian lights in Dorfparkplatz in Gutach became operational – within view of the existing Town Hall traffic lights."}}
3  {"translation":{"de":"Zwei Anlagen so nah beieinander (需要处理输入-输出语言!)","en":"Two sets of lights so close to one another: intentional or just a silly error?"}}
4  {"translation":{"de":"Diese Frage hat Gutachs Bürgermeister gestern klar beantwortet.","en":"Yesterday, Gutacht's Mayor gave a clear answer to this question."}}
```


实验作业

Metrics: BLEU, Human evaluation

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

(来源: Attention is all you need)

Reference:

1. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine
2. Effective Approaches to **Attention**-based Neural Machine Translation
3. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
4. <https://zhuanlan.zhihu.com/p/677231103>

局部注意力

- 局部注意力：解码每一个步骤，只需要关注编码器某些位置(局部位置)的隐藏状态，即图中用虚线部分。相当于**soft** 和**hard attention**的折中。减少计算开销

- 局部位置的选择：在解码第 t 步时，定义在输入中的一个位置 p_t ，在 p_t 周围选择窗口范围

- p_t 的选择：

(1) 单调对齐

$$p_t = t$$

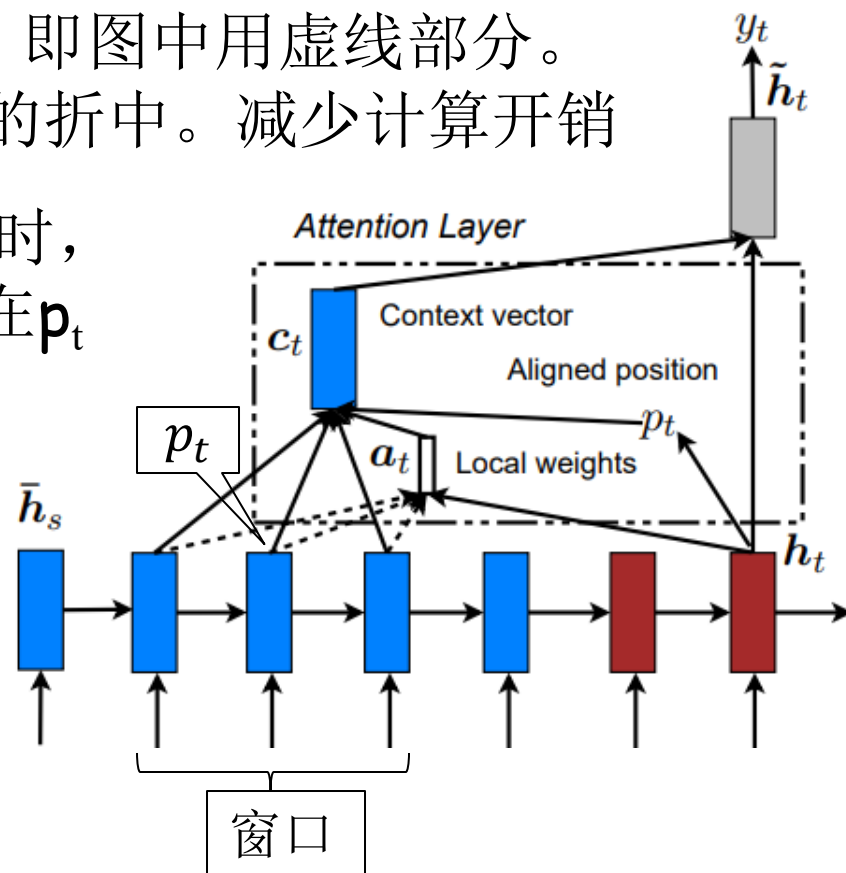
$$\alpha_t(s) = \text{softmax}(\text{score}(h_t, \bar{h}_s))$$

(2) 预测对齐

$$p_t = S \cdot \text{sigmoid}(v_p^T \tanh(W_p h_t)) \quad S: \text{句长}$$

$$\alpha_t(s) = \text{softmax}(\text{score}(h_t, \bar{h}_s)) \exp\left(-\frac{(s-p_t)^2}{2\sigma^2}\right)$$

高斯分布，希望 p_t 接近 t



局部注意力

局部注意力本质是一个局部窗口内的特征聚合，每个位置的权重是注意力相关度，是一个动态计算的局部特征计算模块。

1) 稀疏连接： 解码器隐向量只和局部的编码器隐向量产生交互，类似**CNN** 只和窗口大小内的神经元有连接。

2) 动态权重： 每一个连接的权重都是根据样本特征使用某种方式，动态地计算得到的。(和**CNN**不同)