自然语言处理
Natural Language Processing

Chapter 6
序列到序列模型

Outline

- 机器翻译任务

⇩ 典型任务

- 序列到序列

⇩ 提升思路

- 注意力机制
- Case Study

机器翻译任务

- 机器翻译(Machine translation, MT)的目标是将一种语言(源语言, source language)的句子x自动翻译为另一种语言(目标语言, target language)的句子y
- 机器翻译是一种典型的异步多对多任务，是一种序列到序列的任务

E.g. x: There is no doubt at all that we will win

⇩

y: 毫无疑问我们会赢

三个发展阶段:

早期(基于规则)、中期(基于统计)、目前主流(基于神经网络)

早期阶段

- 基于规则的机器翻译
- 1950s-1990s
- 背景：冷战时期，西方希望破译苏联人的俄语
- 初步工作：1954年IBM实现史上首例机器翻译，早于人工智能一词的提出(1956年)。当时的翻译系统叫做“electronic brain”
- “一位不懂俄语的女孩在IBM的卡片上打出了俄语信息。‘电脑’以每秒2.5行的惊人速度，在自动打印机上迅速完成了英语的翻译。”——IBM报道
- 方法：使用俄语和英语的双语词典，这些词典存储在大型磁带上面，对照着词典进行俄语词汇和对应英语词汇的翻译

第二阶段

- 统计机器翻译(Statistical Machine Translation, SMT)
- 1990s-2010s
- 核心思路: 从数据当中学习概率模型
- 假设进行汉英翻译，则目标为：给定汉语句子x，找到一个最好的英语句子$y=argmax_y P(y|x)$, 即给定x的情况下，找到一个y让概率分布P最大。从数据里学到的概率模型就是这个P
- 使用贝叶斯法则进行分解: $argmax_y P(x|y)P(y)$

翻译模型(Translation model)

对词语和短语的翻译方式进行建模。从大规模双语数据(parallel data，平行数据)中学习得到。

语言模型(Language model)

计算目标句，即单词序列出现的概率。关注句子的流利性。从大规模单语数据中学习得到。

统计机器翻译

- 问题1: 如何学习语言模型 P(y)?
  - 给定词语序列 $y^1, y^2, \ldots, y^{t-1}$, 计算下一个词$y^t$出现的概率:

$$P(y^t \mid y^{t-1}, \ldots, y^1)$$

其中y^t是词典中的任意词语. 则直到位置T，序列y的概率：

$$P(y^1, \ldots, y^T) = P(y^1) \times P(y^2|y^1) \times \cdots \times P(y^T|y^{T-1}, \ldots, y^1)$$

$$= \prod_{t=1}^{T} P(y^t \mid y^{t-1}, \ldots, y^1) \rightarrow \text{似然} \frac{\text{count}(y^1, \ldots, y^t)}{count(y^1, \ldots, y^{t-1})}$$

计算量巨大

- 举例：手机输入法；搜索引擎
- 下一个词概率计算的简化:
  - 基于前n-1个词 $y^{t-1}, \ldots, y^{t-n+1}$: n元语法模型 (n-gram)

E.g. I'm angry because he broke the _ → y^t

n=1: 只考虑y^t概率
n=2: the (y^{t-1}) 一阶马尔科夫链
n=3: broke the (y^{t-2} y^{t-1})

平滑(smoothing)

- 问题: 计算P(y)时出现频次=0怎么办?
- **稀疏性问题**
- e.g. I felt angry because he broke the _. 答案: oath

$$p(oath|broke\ the) = \frac{count(broke\ the\ oath)}{count(broke\ the)}$$

分子可能=0， 分母也可能=0

- 直接方案: 增加训练数据。但自然语言中大部分的词都是低频词，边际效益会随着数据集规模的增加而递减。

平滑(smoothing)

- 目的：调整最大似然估计的概率值，使零概率增值，非零概率下调，"劫富济贫"，消除零概率，改进模型的整体正确率

(1) 加法平滑(拉普拉斯平滑)

$$\frac{\text{count}(y^{1-n+1},\ldots,y^{t})+\delta}{count(y^{1-n+1},\ldots,y^{t-1})+\delta|V|}$$

δ 通常很小，可以是一个常数，也可以是一个参数

特殊情况：$\delta = 1$，称为加一平滑

平滑(smoothing)

(2) 减值平滑

修改训练样本中事件的实际计数，使样本中(实际出现的)不同事件的概率之和小于1，剩余的概率量分配给未见概率。

❖ 古德-图灵平滑(Good-Turing)

假设一共有N个n元语法，n_r是出现了r次的n元语法的类别数，则

$$N = \sum_{r=1}^{\infty} n_r \cdot r = \sum_{r=0}^{\infty} n_{r+1} \cdot (r+1)$$

减小r，替换为$r^* = (r+1)\frac{n_{r+1}}{n_r}$

则样本中出现r次的事件概率 = $p_r = \frac{r^*}{N}$，所有实际出现的事件概率和 = $\sum_{r>0} n_r \cdot p_r$ <1

平滑(smoothing)

❖ 退回(back off). 例如3元语法退回2元语法.

当某一事件在样本中出现的频率 > 阈值K (通常0 或1)时，运用最大似然估计的减值法来估计其概率，否则，使用 n-1元语法的概率替代

假设一共有N个n元语法，对于出现了r次的n元语法，待预测词 y^t，前面n-1个词序列是y_{n-1}^t，将概率p修改为：

$$p_r = \begin{cases} d_r \cdot p, & r \geq K \quad d_r < 1，\text{p减值(留出概率值)} \\ \alpha(y_{n-1}^t)\dfrac{c(y_{n-1}^t)}{N}, & r < K \quad \text{分子=n-1元语法频次} \end{cases}$$

n-1元语法概率，受归一化因子作用

平滑(smoothing)

❖ 绝对减值法

从样本中每个实际出现的n元语法计数中减去同样的量，剩余的概率量由n_0个未见的n元语法均分。

❖ 线性减值法

从样本中每个实际出现的n元语法计数r中减去和r成正比的量，剩余的概率量被n_0个未见的n元语法均分。

平滑(smoothing)

(3) 删除插值法(Deleted interpolation)

用低阶语法估计高阶语法，即当 3-gram 的值不能从样本中准确估计时，用 2-gram 来替代；当 2-gram 的值不能从样本中准确估计时，用 1-gram 的值来代替。插值公式：

$$P'(w_3|w_1w_2) = \lambda_3 P(w_3|w_1w_2) + \lambda_2 P(w_3|w_2) + \lambda_1 P(w_3)$$

权重和=1

也体现了back off的思想

统计机器翻译

- 问题2: 如何学习翻译模型P(x|y)?
  - 需要大量成对的平行数据 (e.g.成对的英语-汉语翻译)

- 问题: 如何从平行语料中学习翻译模型?
  - 预测给定目标语言句子y, 源语言句子x的条件概率分布。进一步目标分解：P(x, a|y)

a 是词级别的对齐(alignment), e.g. 自然-natural, 语言-language, 处理-processing,

对齐(alignment)

- 对齐(alignment)：翻译句子对中特定单词之间的对应关系.

- Note: 对齐过程中，不一定会每次都找到对应的词语

他 → He

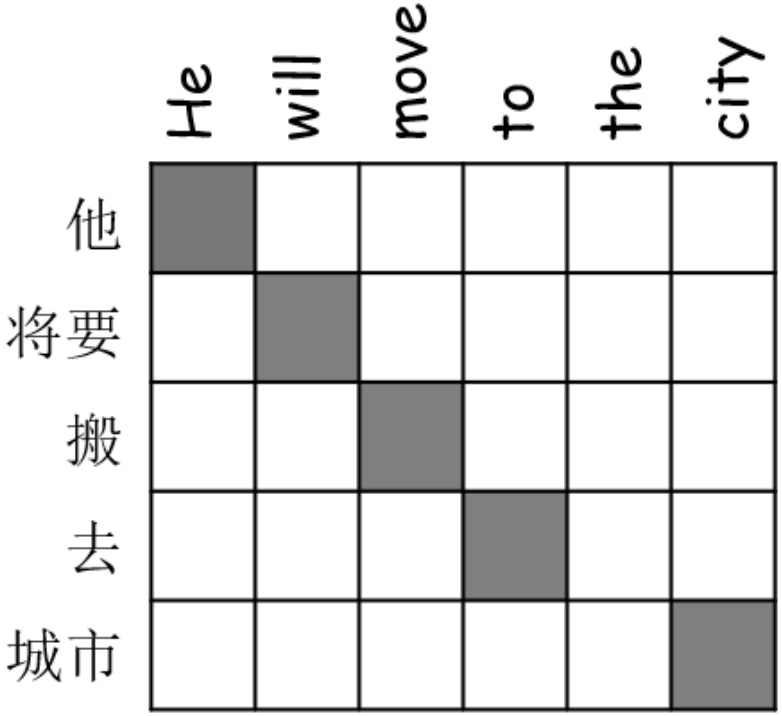
将要 → will

搬 → move

去 → to

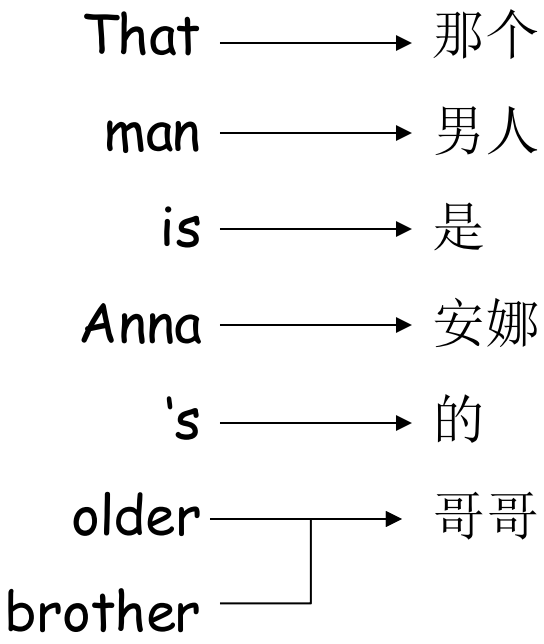
the 'spurious' word 虚假词

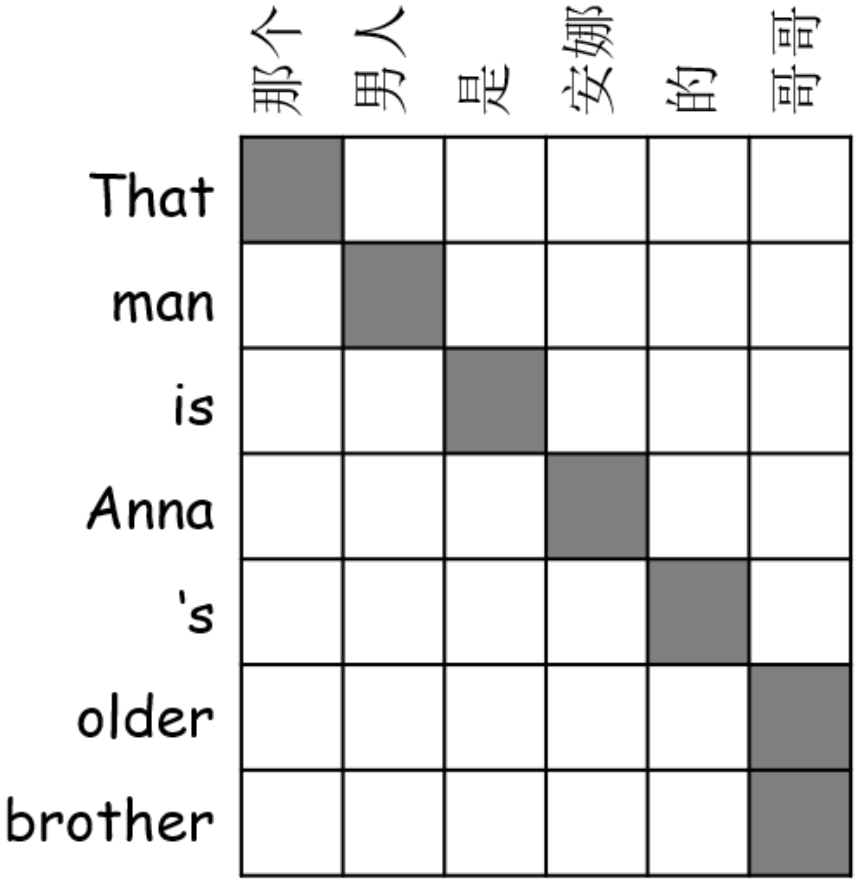
城市 → city



对齐(alignment)

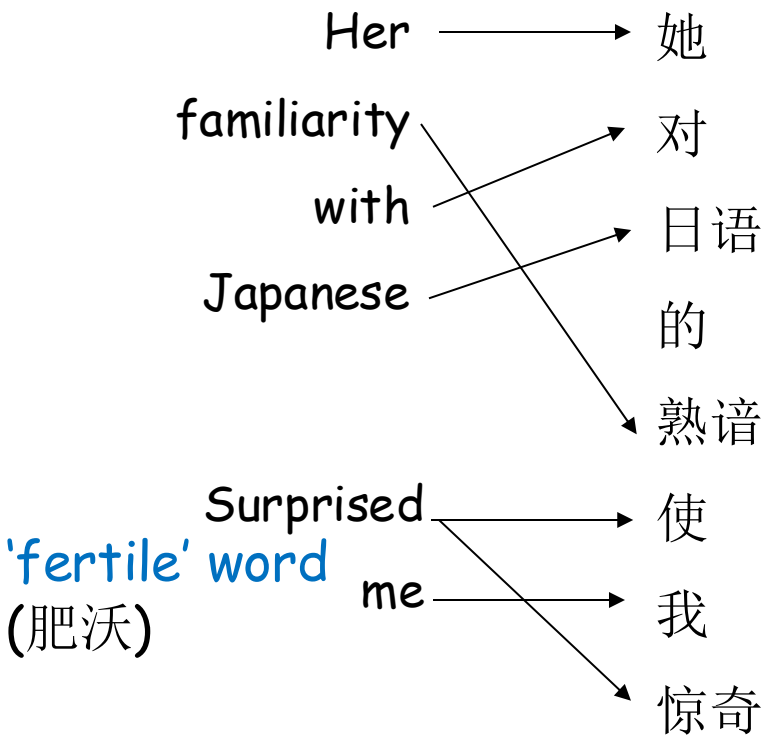
- 对齐(alignment)可能是多对一的

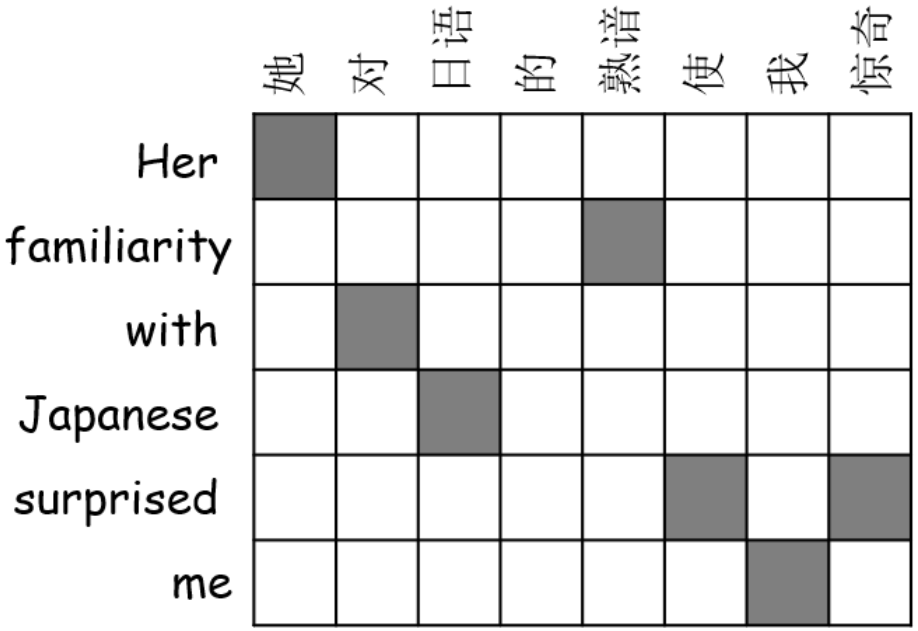




对齐(alignment)

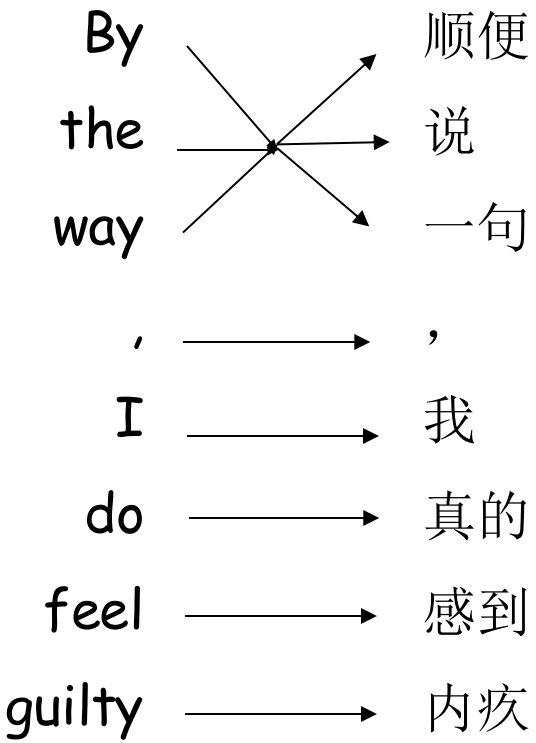
- 对齐(alignment)可能是一对多的

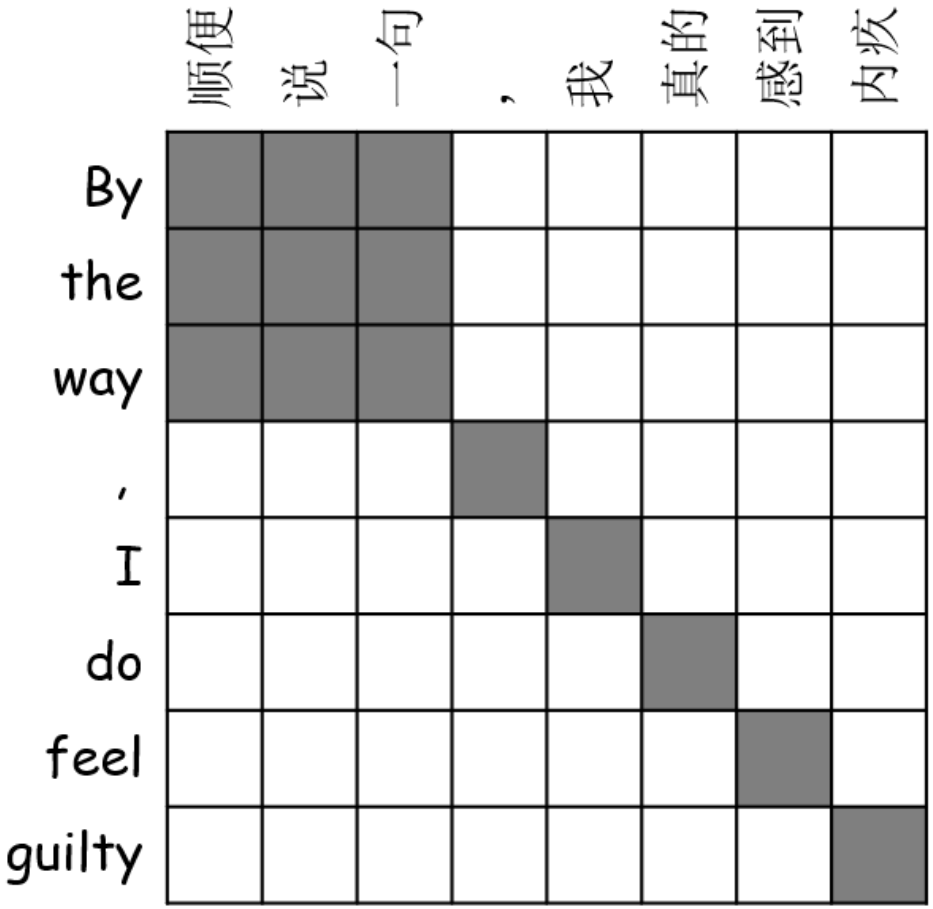




对齐(alignment)

- 对齐(alignment)可能是多对多的(短语级)



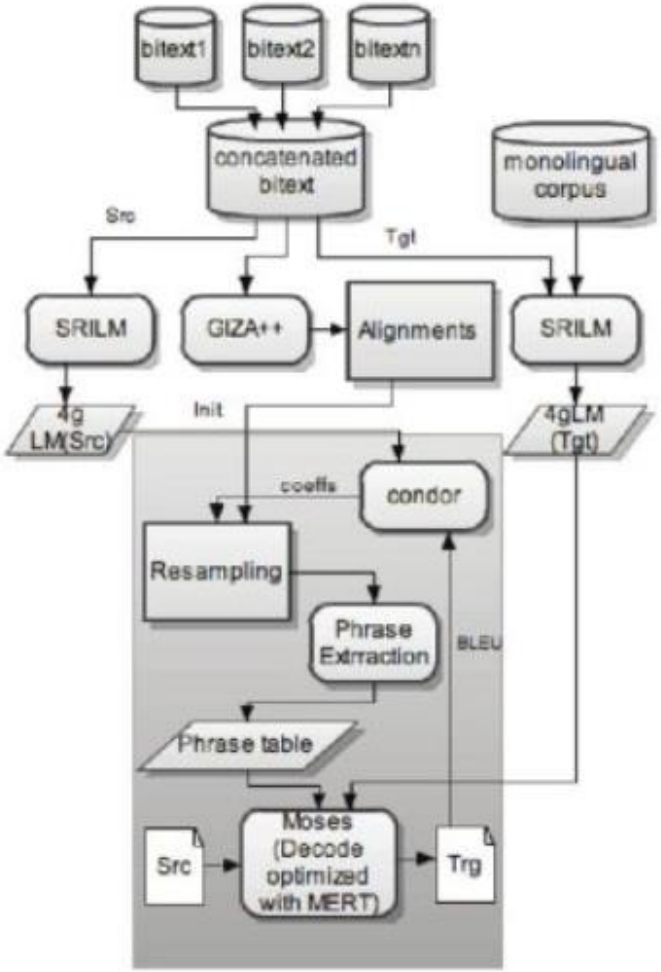


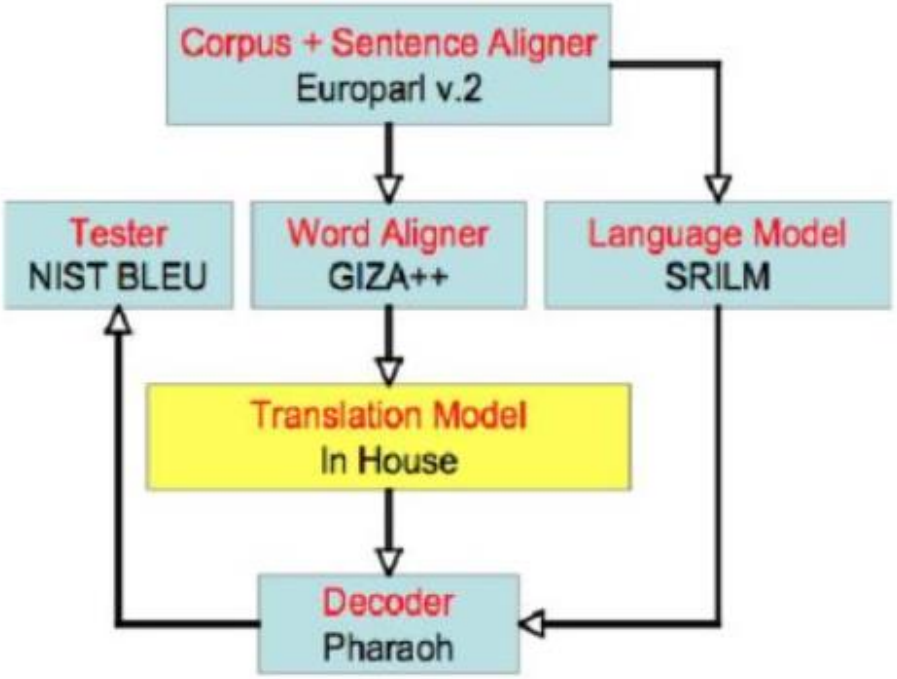
统计机器翻译

- 统计机器翻译是一个庞大、复杂的研究领域
  - 一个翻译系统包含很多需要单独设计的子组件，包含很多细节，比如怎么对齐？怎么平滑？
  - 需要进行大量的特征工程，比如某种语言里特殊的语言现象需要设计特征来刻画，需要收集和维护额外的资源
  - 需要存储翻译过程的中间结果，比如一个词或者短语有几个可能的翻译结果，需要先记录不同的假设，在翻译过程中根据是否符合进行剪枝，最后根据句子整体，研究如何把局部的翻译做整合。
  - 需要大量人力进行维护，需要针对新的语言进行重复的工作

统计机器翻译

机器翻译功能模块的分解（示例）： 双语对齐， 语言模型， 翻译模型， 解码器， 测试。



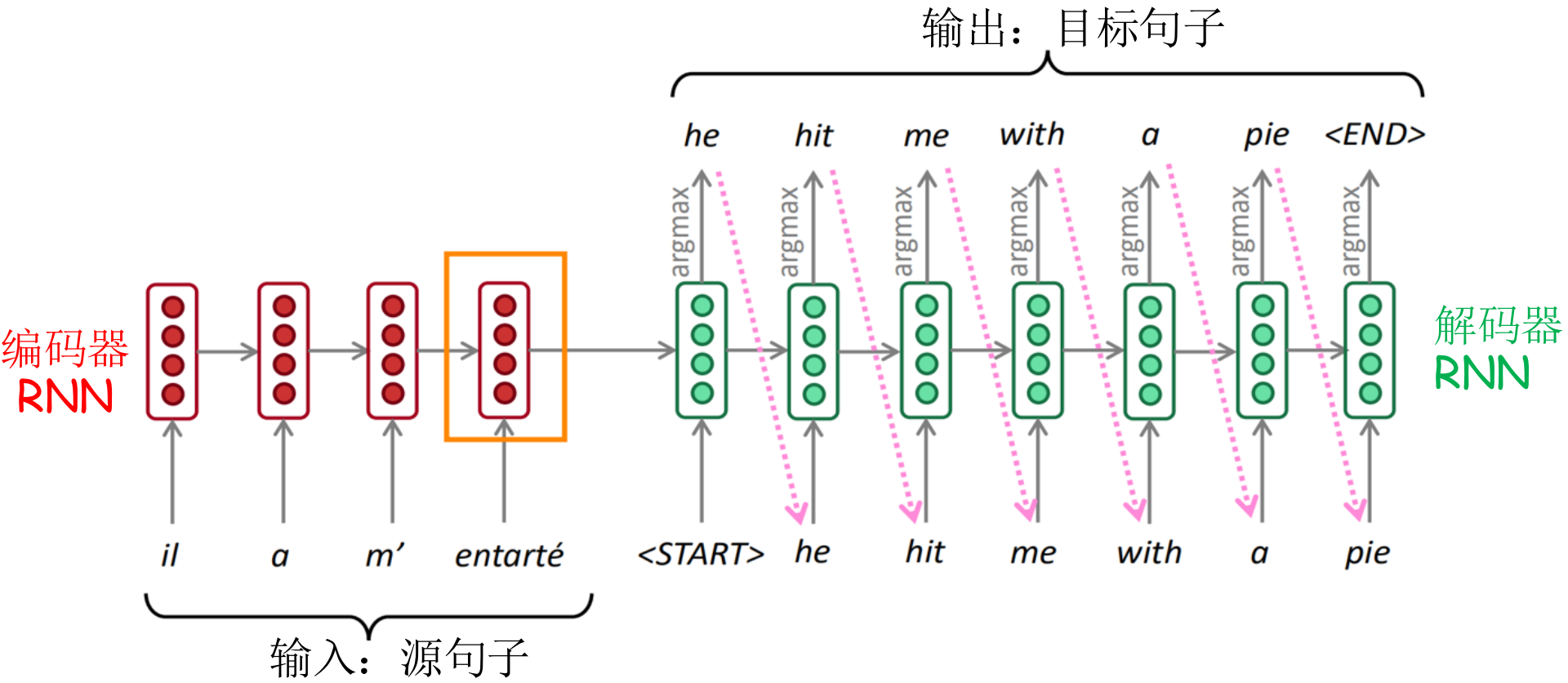


第三阶段

- 神经机器翻译(Neural Machine Translation, NMT)
- 约2014年至今，目前的主流方法
- 只用一个单独的神经网络框架进行翻译

- 这种神经网络结构叫做sequence-to-sequence序列到序列(简称seq2seq)
- 使用两个模型分别处理 源句子序列 和 目标句子序列，处理输入序列的部分叫做encoder编码器，生成输出序列的部分叫做decoder解码器。
- 最基础的seq2seq：使用RNN作为编解码器
- 可以把这种seq2seq结构叫成encoder-decoder结构

神经机器翻译



编码器RNN对于整个输入句子，产生了一个向量(encoding)，传给解码器RNN作为解码依据

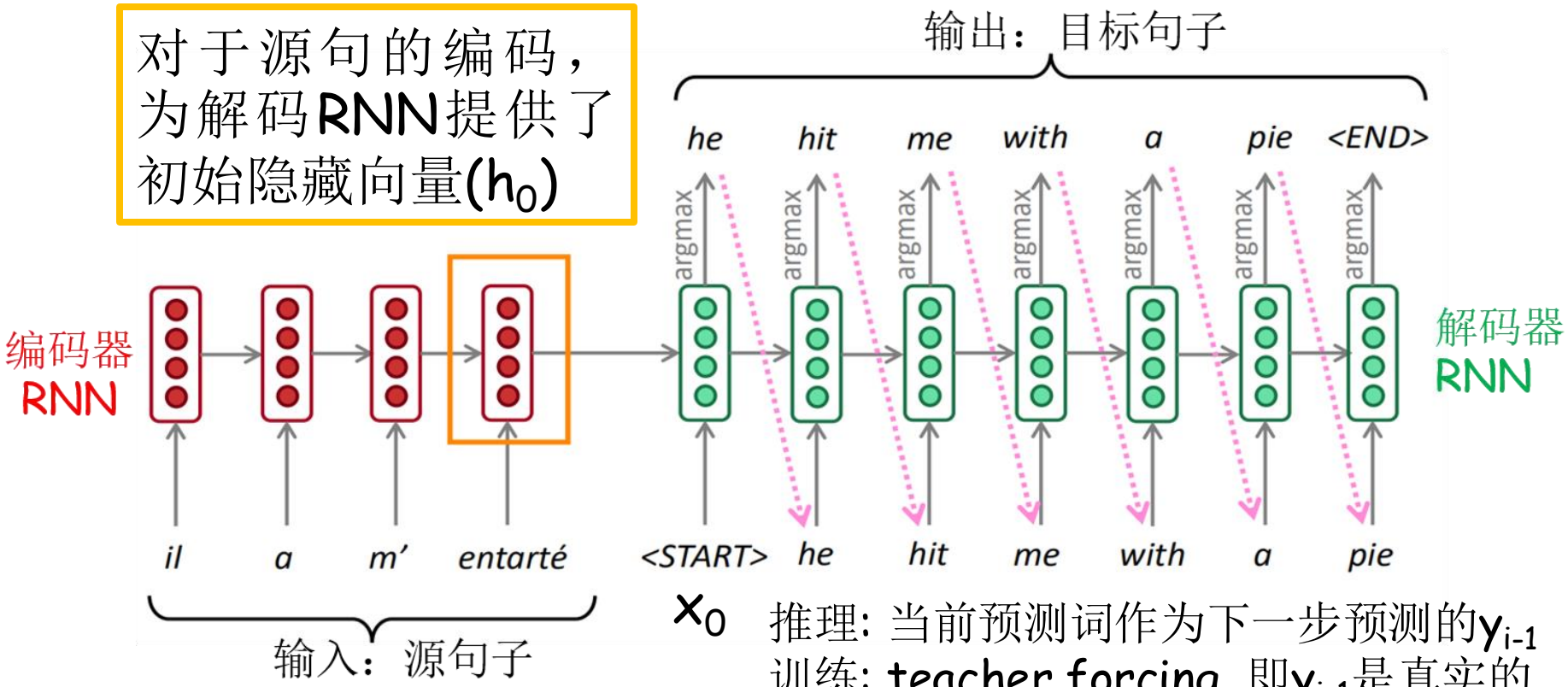
除RNN以外的其他方式:

1) $h = \sum \alpha * w$

2) $h = FNN(w^{1-n+1}, \ldots, w^{t-1}; \theta)$

神经机器翻译

对于源句的编码，为解码RNN提供了初始隐藏向量(h_0)



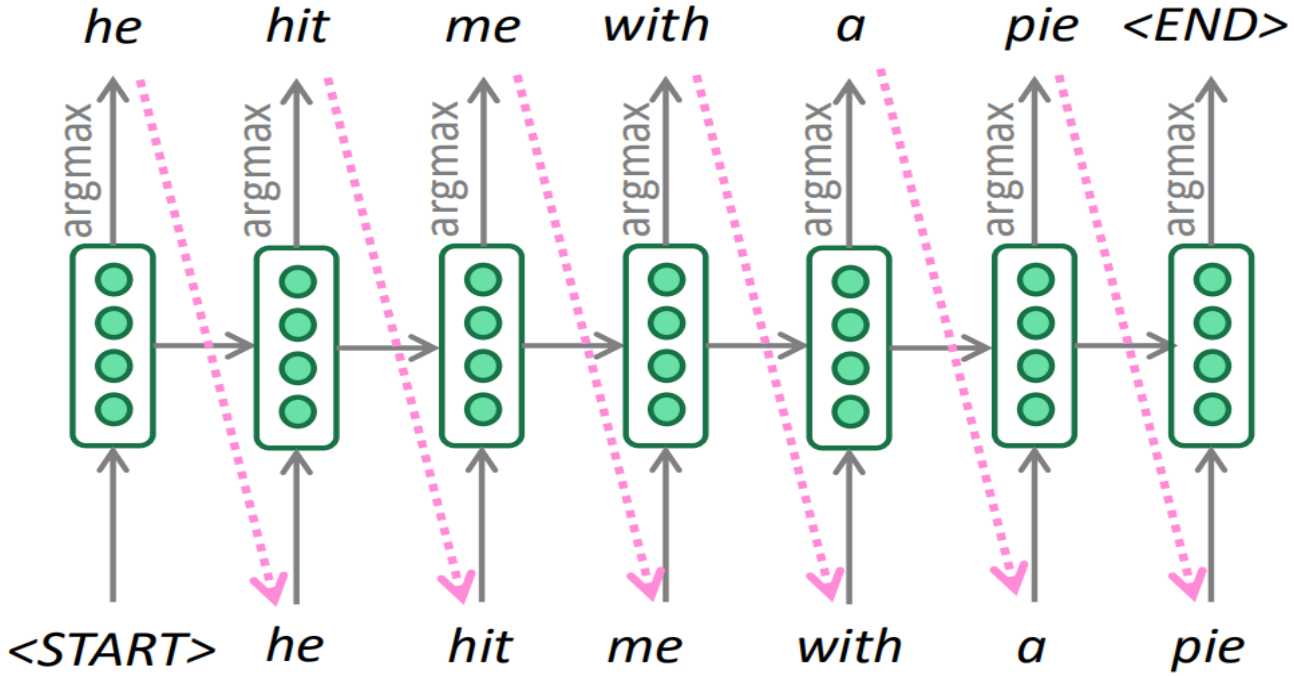
推理：当前预测词作为下一步预测的y_{i-1}
训练：teacher forcing，即y_{i-1}是真实的词语，而非预测的

编码器RNN对于整个输入句子，产生了一个向量(encoding)，传给解码器RNN作为解码依据

解码器RNN是一个语言模型，依据编码器提供的encoding (conditioned on...) 逐词生成目标句子 (**条件**语言模型)

解码策略

- 贪心(Greedy)策略：解码过程中，每一步通过argmax选择输出概率最大的那一个词语，直到生成结束，得到目标句子



- 可能导致的问题：错误传播(error propagation)，基于当前错误的预测词进行下一步预测，一步错、步步错

怎么缓解？

解码策略

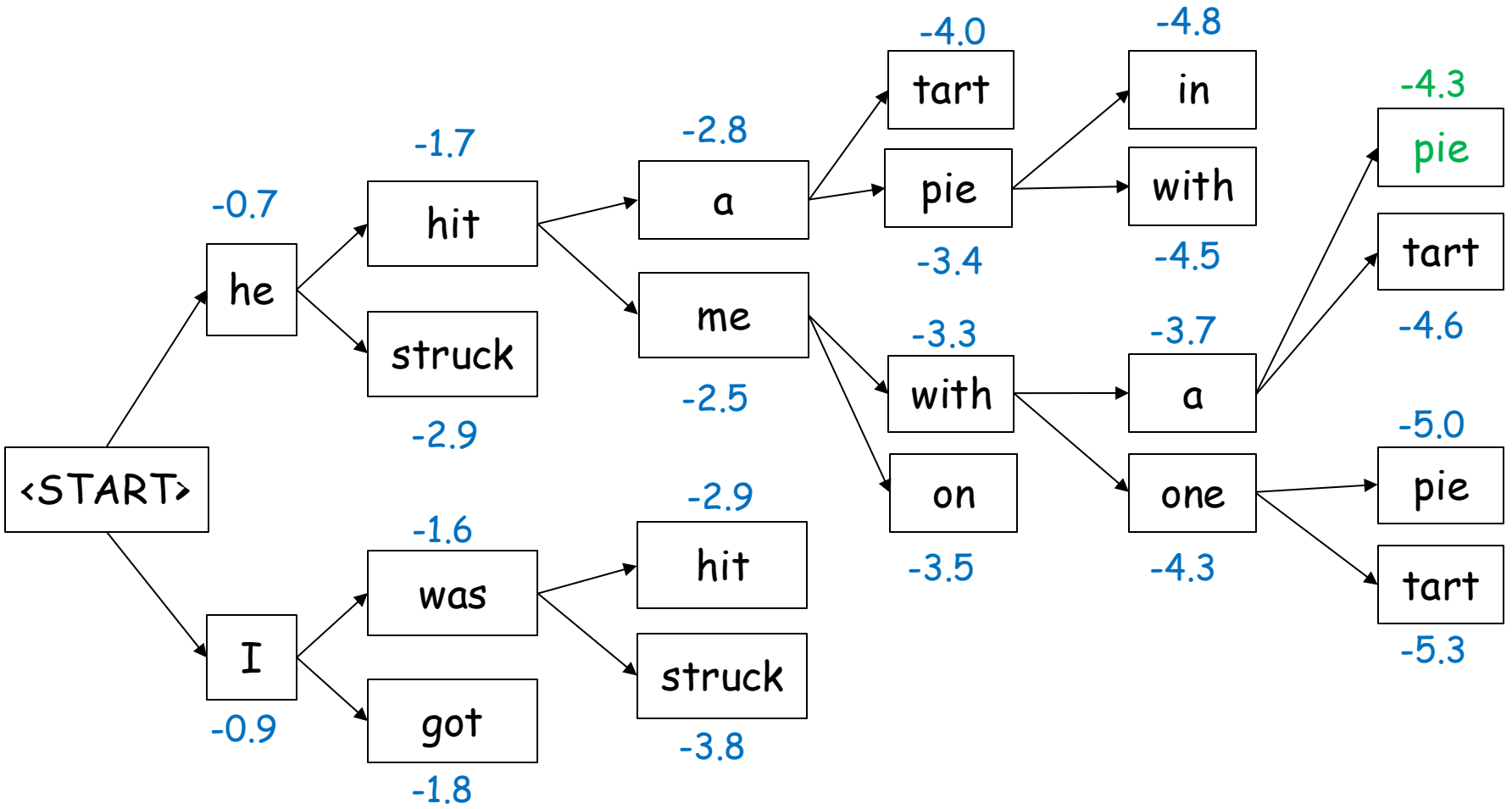
- 束搜索(beam search): 解码的每一步中，保持k个概率最大的部分翻译(partial translations)，可以称为hypotheses。它是翻译过程中直到当前步骤生成的非完整的序列片段。
- k称为beam size
- 一个hypothesis $y_1, \ldots, y_t$的分数是如下的对数概率：

$$score(y_1, \ldots, y_t) = \log P(y_1, \ldots, y_t|x)$$
$$= \log [\, \mathrm{P}(y_1|x) \times \mathrm{P}(y_2|y_1, x) \times \cdots \times P(y_t|y_1, \ldots, y_{t-1}, x)]$$
$$= \sum_{i=1}^{t} \log P(y_i|y_1, \ldots, y_{i-1}, x)$$

- beam search并不能100%保证得到最好的输出
- 但是一种合适的折中策略，比穷举法代价小，比贪心法精度高

束搜索(beam search):

- k = 2, 蓝色数字 = $score(y_1, \dots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$



解码停止策略

- 目标句所在词表中人为设置了一个特殊的字符<END>(或其他形式)，代表终止符。解码过程中，一旦产生了<END>则停止解码。
  - E.g.: <START> he hit me with a pie <END>
- 在束搜索中，不同的hypotheses可能会在不同的时间步产生<END>。假如其中一个hypothesis产生了<END>，其他的hypotheses怎么办呢？
  - 结束掉这一条路径，其他的hypothesis还是照样继续按照beam search的方式解码，直到产生自己的<END>

❖其他策略:

- 达到了预定义的最大时间步T，即已经生成了T个词语
- 已经产生了n 个完整的输出序列 (n 是预定义的)

解码策略

- 通过束搜索，获得了若干个输出序列的候选. 怎么选择最合适的、得分最高的输出序列？
- 1个hypothesis $y_1, \ldots, y_t$的分数是如下的对数概率：

$$score(y_1, \ldots, y_t) = \log P(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P(y_i | y_1, \ldots, y_{i-1}, x)$$

- 问题: 对数概率是一个负数，加法次数越多值越小，因此序列越长，得分往往越低，模型会倾向于生成很短的翻译结果

- 解决方案: 通过序列长度进行得分标准化，减少序列长度带来的偏差影响

$$score(y_1, \ldots, y_t) = \frac{1}{\mathrm{t}} \sum_{i=1}^{t} \log P(y_i | y_1, \ldots, y_{i-1}, x)$$

- $y = \mathrm{argmax}_y \, score$

解码策略

- 在解码过程中添加**随机性**，以文本序列任务为例：

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Beam Search, *b*=32:
"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ..."

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

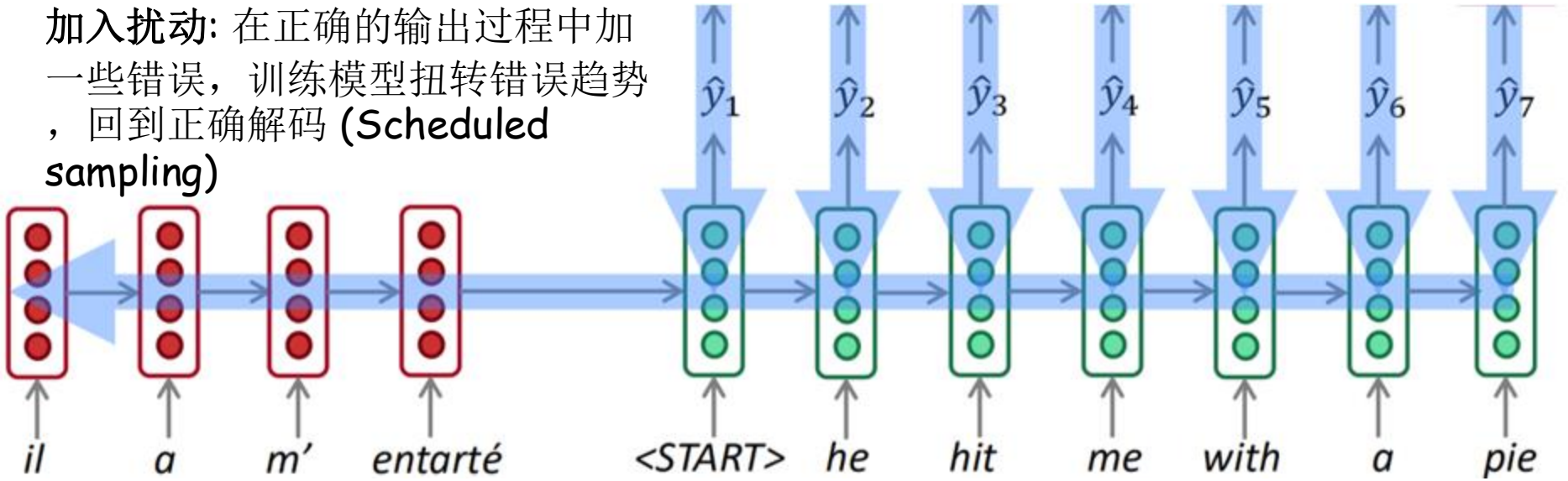
- 一般来说是否添加随机性和seq2seq具体任务有关
- 如果要做的任务答案是比较明确的，如语言转文字，稍微长一点的句子就不至于有歧义，这样的答案就是比较确定的，这种情况适合用beam search。但如果这个任务答案不确定性、多样性较大，需要模型自己加入创意，那么就需要在解码中添加随机性。

神经翻译系统训练

- 训练数据：大量的(源句，目标句)对 组成的语料库
- 计算给定句子x，生成句子y的概率p(y|x)，取负对数作为损失

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t = J_1 + J_2 + J_3 + J_4 + J_5 + J_6 + J_7$$

加入扰动: 在正确的输出过程中加一些错误，训练模型扭转错误趋势，回到正确解码 (Scheduled sampling)

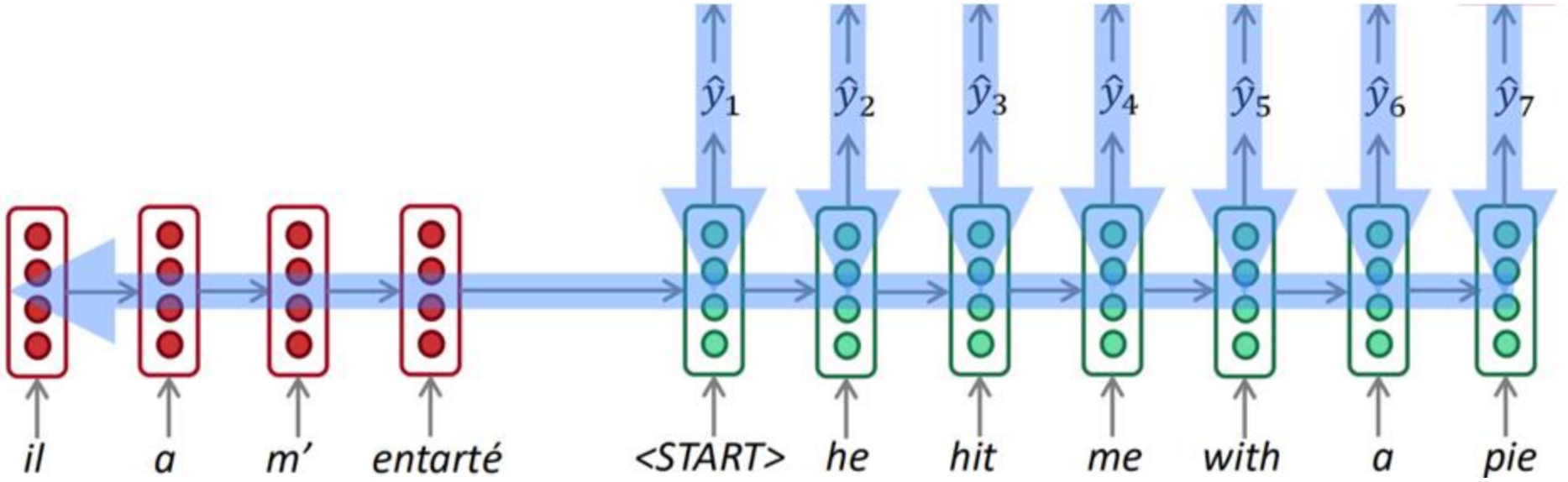


语料中的源句x　　语料中的目标句y

训练模式：teacher forcing，即使用训练数据的标准答案作为每一步骤的输入词

神经翻译系统训练

- Seq2seq是一个整体系统。输入源句，输出目标句，前向传播不需要中间的人为操作。
- 反向传播中，编码器、解码器作为一个整体训练优化。
- 端到端的系统(end-to-end)



序列到序列模型的应用

- 神经机器翻译只是Seq2seq的典型应用场景之一
- 很多NLP任务都可以转化为序列到序列的形式，并使用encoder-decoder架构解决，e.g.：

-文本摘要 (长文章→ 较短的摘要文本)

- 对话生成 (前一句话→ 后一句话)

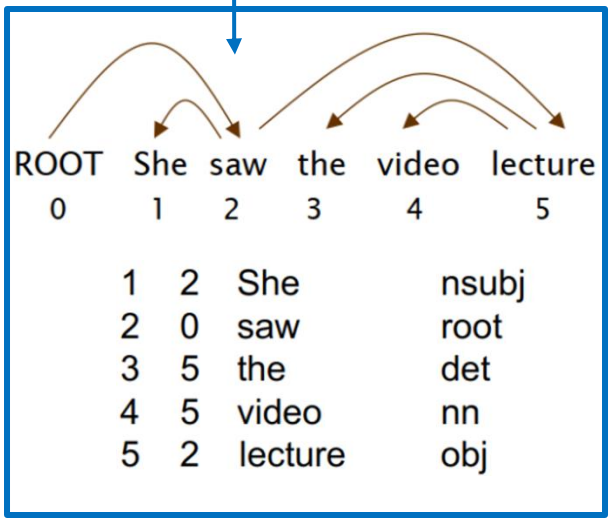
- 依存分析 (输入文本 → 依存分析结构序列)

- 多标签分类(待分类文本 → 标签序列)

- 代码生成 (自然语言→ 代码)

- ……

类别标签：体育0, 财经1, 政治2, 法制3, 社会4, 娱乐5, 房产6…
Input = 吴亦凡判了,坐牢13年, 补税6亿外加驱逐出境
Output = "3 5"



Seq2seq: 早期工作

- Cho et al, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, *ACL*, 2014. PDF

- 基于RNN的编解码器框架

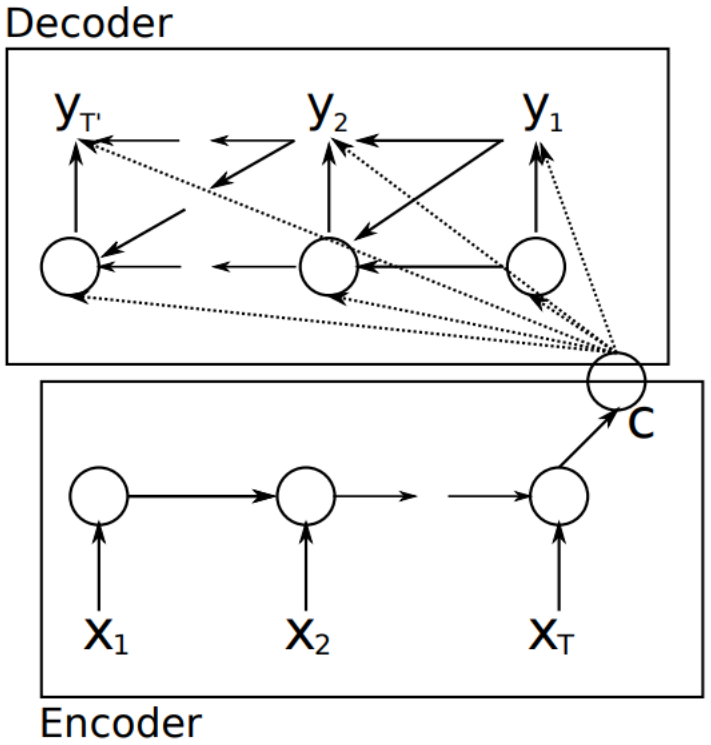
- 编码器:

  $h_t = f(h_{t-1}, x_t)$

- 解码器:

  $h_t = f(h_{t-1}, x_t, c)$

  $P(y_t|y_{t-1},\dots,y_1, c) = g(h_t, x_t, c)$



示例代码

```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

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

就是一个RNN (GRU版)

```python
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden

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

也是一个RNN (GRU版)

解码阶段的初始化

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

decoder_hidden = encoder_hidden
```

类似<start>的开始标记

隐向量h_0=编码器结果

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