端对端语音识别分享

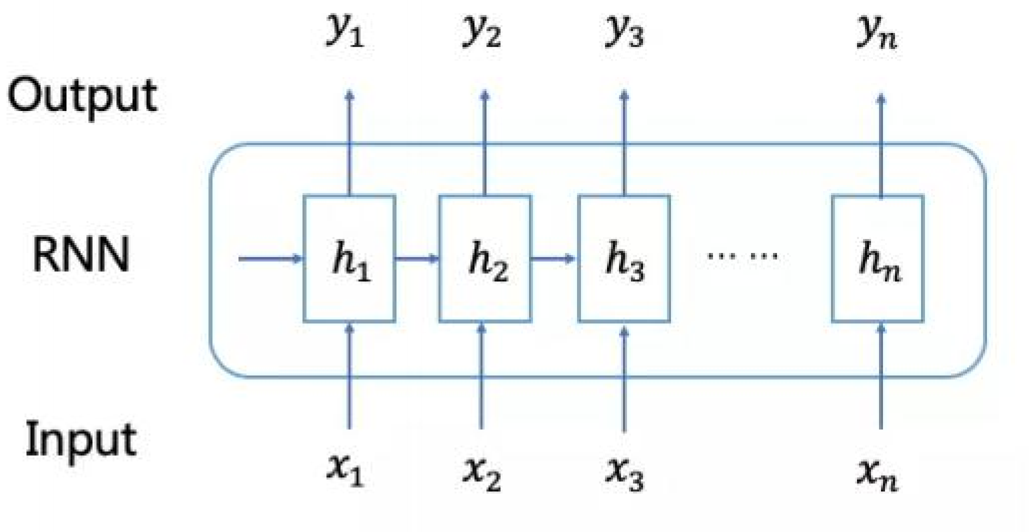
分享人：黄佳恒
2019.12.16

什么模型可以做语音识别？

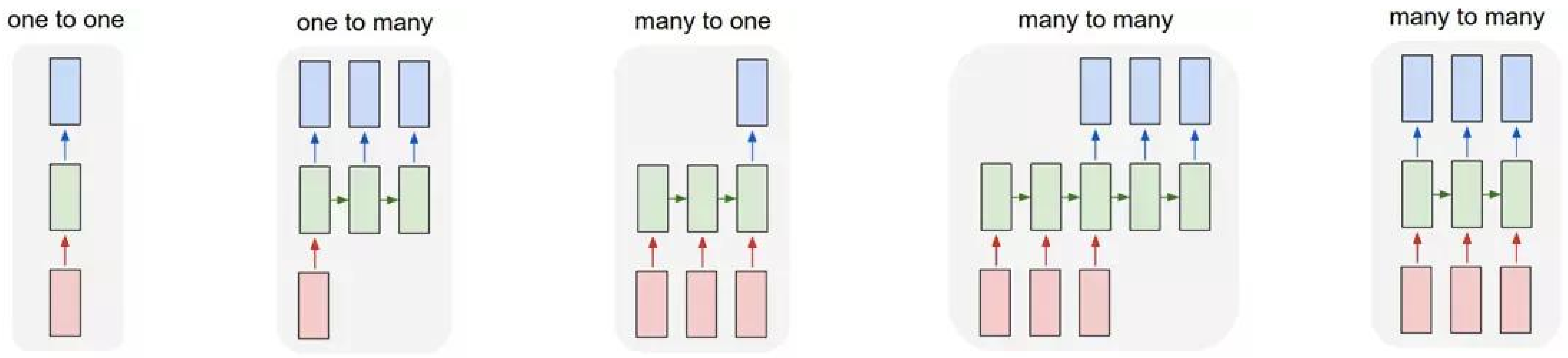
RNN能做什么

1. 文本分类
2. 词性标注

RNN要求：
输入序列和输出序列质检映射关系提前标注好



rnn



RNN不能做什么

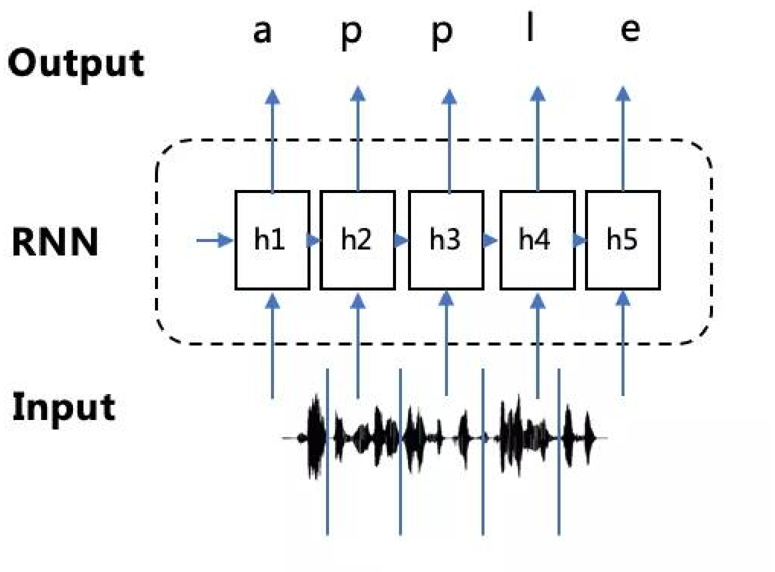
天然难分割：音频、图像

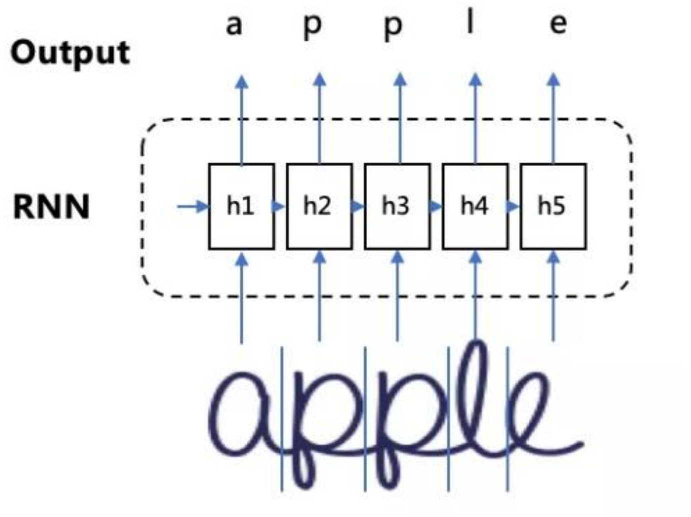
为什么呢？

1）音频的最小单位应该是什么？
2）标注的量级大吗？

说话人语速有快有慢
xua
一个音素持续时间50-100ms之间
一般选取20、30、40、50ms

分割并标注映射关系的数据依赖是不切实际的



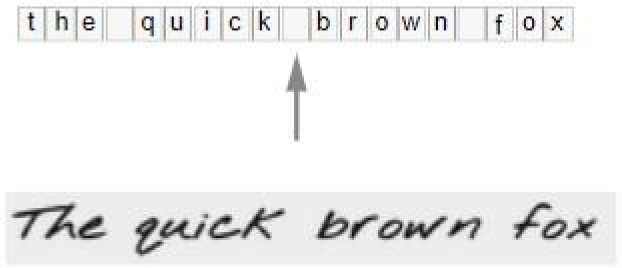


CTC

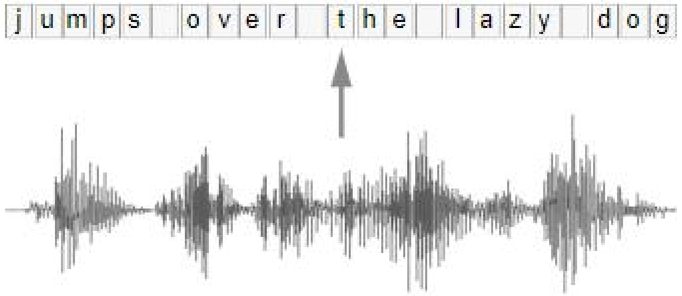
Connectionist Temporal Classication(连接时序分类器)

扩展了RNN的输出层，在输出序列和最终标签之间增加了多对一的空间映射，并在此基础上定义了CTC Loss函数

识别过程：https://src.ailemon.me/blog/2019/20190718-1.gif



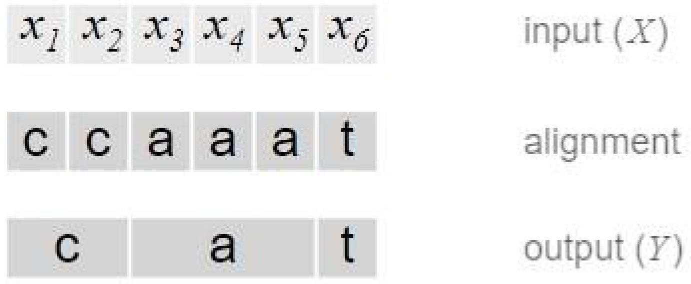
手写识别: 输入可以是 (x, y) 笔划的坐标或图像中的像素



语音识别: 输入可以是语谱图或其他基于频率的特征提取器

对齐

- 这种方法有两个问题。
- 通常，强制每个输入元素与某些输出对齐是没有意义的。 例如，在语音识别中，输入元素可以是没有相应输出的静音段。
- 我们无法生成连续的多个字符的输出。 比如对齐的[h，h，e，l，l，l，o]，合并重复将产生“helo”而不是“hello”。



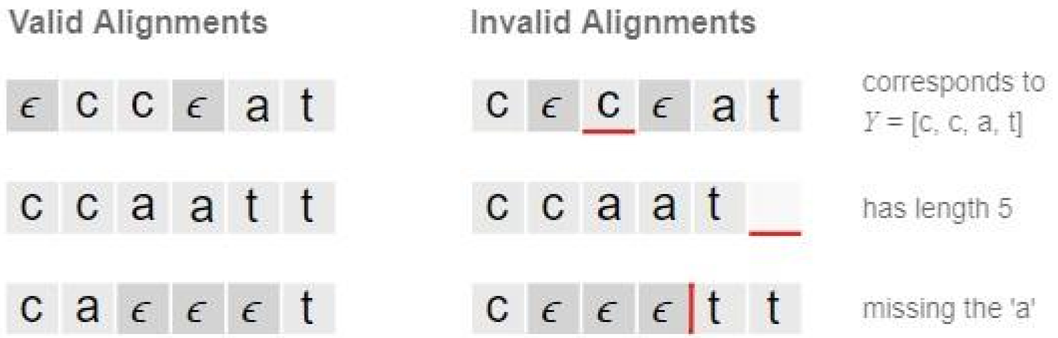
引入空标记

.为了解决这些问题，CTC为允许的输出集引入了一个新的标记，这个标记有时被称为空标记。我们在这里将它称为ϵ。ϵ标记与任何内容都不对应，只会从输出中删除。

.CTC允许的对齐长度与输入的长度相同。 我们允许在合并重复和删除ϵ标记后映射到Y的任何对齐：

.如果Y在一行序列中有两个相同的字符，那么有效的对齐必须在它们之间有一个ϵ。 有了这个规则，我们就可以区分出“hello”的合并对齐和“helo”的合并对齐。

.让我们回到输出长度为6的输出[c，a，t]。 以下是有效和无效对齐的更多示例。



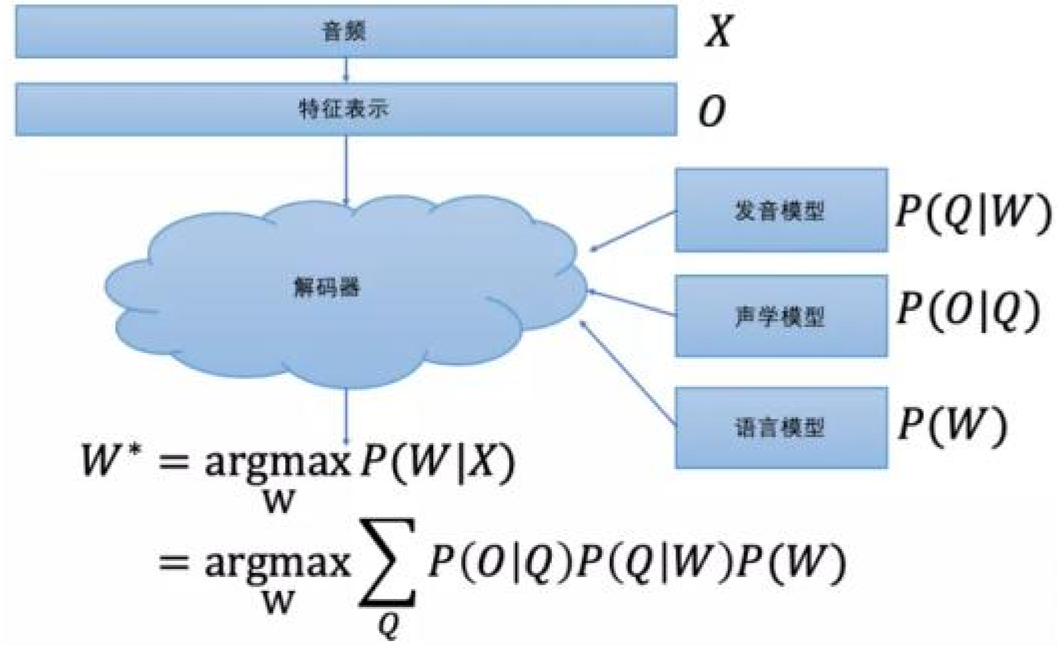
CTC Loss

1. X是音频信号，O是它的特征表示
   一般使用MFCC、Fbank提取特征

2.Q是O对应的发音序列，建模单元，
   一般是音素

3. W是音频信号X对应的文字

贝叶斯分解
引入隐变量（更符合常理）
求积分

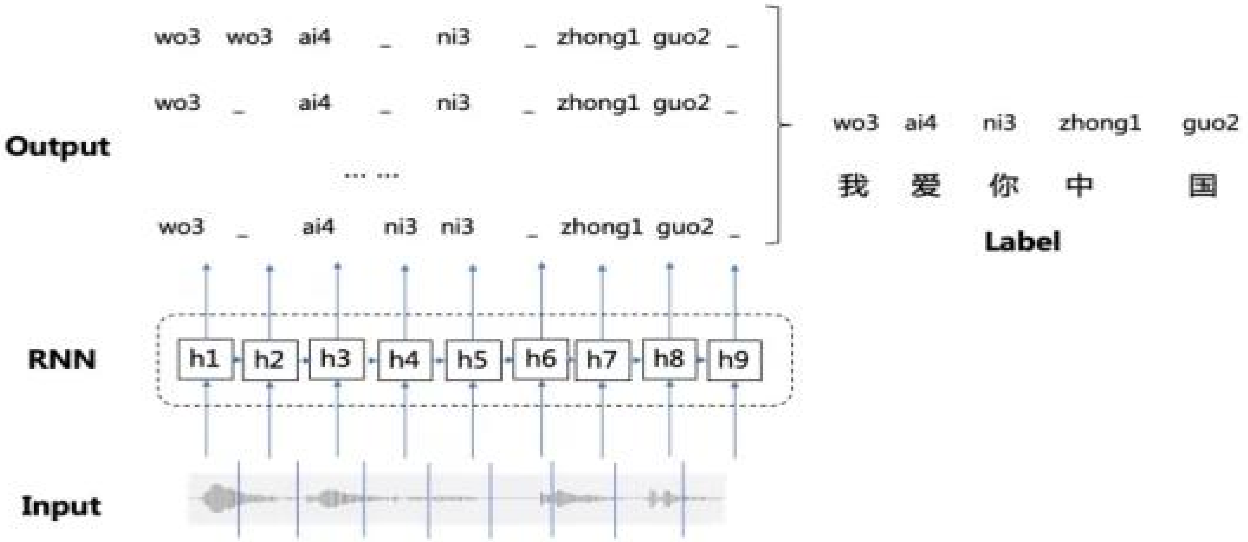


发音模型

声学模型

语言模型

CTC LOSS

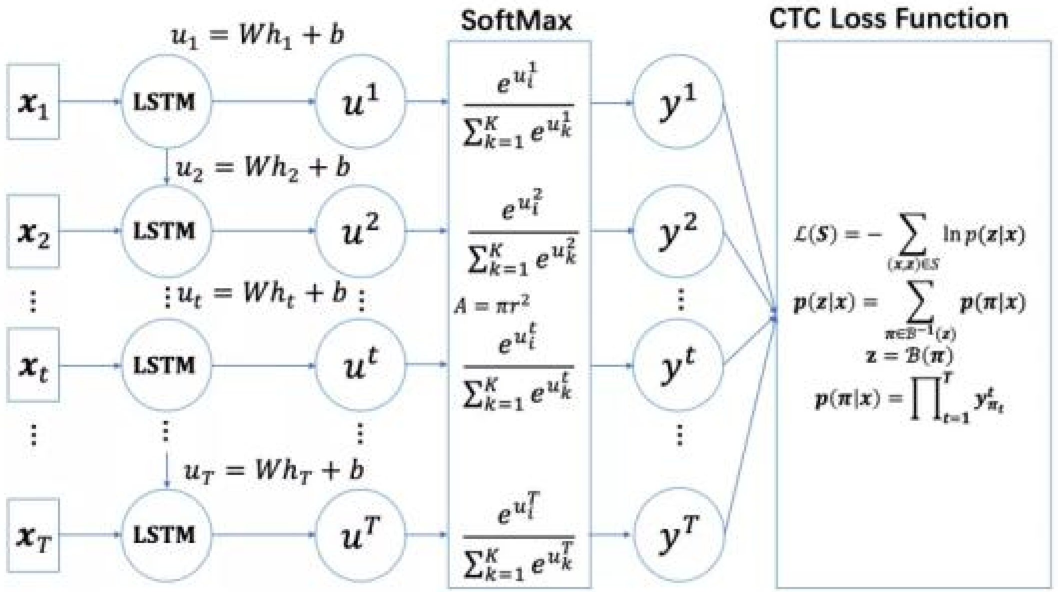


- 训练一个时序分类器：$\boldsymbol{h}(\boldsymbol{x}) = \underset{z \in L^{\mathrm{T}}}{\operatorname{argmax}}\, \boldsymbol{p}(\boldsymbol{z}|\boldsymbol{x})$
- Loss函数$\mathcal{L}(\boldsymbol{S}) = -\prod_{(x,z)\in S} p(\boldsymbol{x}, \boldsymbol{z}) =: -\sum_{(x,z)\in S} \ln p(\boldsymbol{x}, \boldsymbol{z})$
  - $\boldsymbol{S}$ => training data set from a fixed distribution $\mathcal{D}_{\mathrm{X}\times\mathrm{Z}}$
  - Input Space : $\mathbf{X} = (\mathbb{R}^m)^*$ => m维实数向量序列集合
  - Target Space : $\mathbf{Z} = \boldsymbol{L}^*$ => 由有限字符集$\boldsymbol{L}$组成的序列集合
  - $\boldsymbol{x}$序列的长度大于或等于$\boldsymbol{z}$序列的长度

CTC LOSS

- 假设给定输入序列和模型参数，RNN每一时刻的输出之间是条件独立的，则：
  - $\boldsymbol{p}(\boldsymbol{\pi}|\boldsymbol{x}) = \prod_{t=1}^{T} \boldsymbol{y}_{\boldsymbol{\pi}_t}^{t}, \forall \boldsymbol{\pi} \in \boldsymbol{L}'^{\mathrm{T}}$
  - $\boldsymbol{p}(\boldsymbol{z}|\boldsymbol{x}) = \sum_{\boldsymbol{\pi} \in \mathcal{B}^{-1}(\boldsymbol{z})} \boldsymbol{p}(\boldsymbol{\pi}|\boldsymbol{x})$，$\mathcal{B}^{-1}$是$\boldsymbol{z}$全部路径集合的映射函数

- 我们得到CTC的Loss函数，如下：
  - $\mathcal{L}(\boldsymbol{S}) = -\ln \prod_{(\boldsymbol{x},\boldsymbol{z}) \in S} p(\boldsymbol{z}|\boldsymbol{x})$
  - $\quad = -\sum_{(\boldsymbol{x},\boldsymbol{z}) \in S} \ln p(\boldsymbol{z}|\boldsymbol{x})$
  - $\quad = -\sum_{(\boldsymbol{x},\boldsymbol{z}) \in S} \ln \sum_{\boldsymbol{\pi} \in \mathcal{B}^{-1}(\boldsymbol{z})} \boldsymbol{p}(\boldsymbol{\pi}|\boldsymbol{x})$
  - $\quad = -\sum_{(\boldsymbol{x},\boldsymbol{z}) \in S} \ln \sum_{\boldsymbol{\pi} \in \mathcal{B}^{-1}(\boldsymbol{z})} \prod_{t=1}^{T} \boldsymbol{y}_{\boldsymbol{\pi}_t}^{t}$， $\forall \boldsymbol{\pi} \in \boldsymbol{L}'^{\mathrm{T}}$



动态规划

前向、后向到底解决了什么问题？

1. 计算时间复杂度优化
2. 方便求导的计算。因为梯度已经分解成了alpha和beta的函数
   复杂的损失函数分解成了alpha和beta和y(t)
3. 动态规划不是贪婪算法，没有损失最优解

LOSS损失函数计算

暴力计算P(z|x)的复杂度非常高
使用Forward-Backward算法，利用动态规划求解

前向后向为甚什么会减少时间复杂度？

举个例子，斐波那契数列 0,1,1,2,3,5,8,13,…有着一个相当简单的描述方式，它的每个数字都与前两个紧邻的数字相关。如果 F(n) 是第 n 个数字，那么我们会有 F(n) = F(n-1) + F(n-2)。这个在数学上称作*递归方程*或者*递推关系*。为了计算后面的项，它需要前面项的计算结果作为输入。

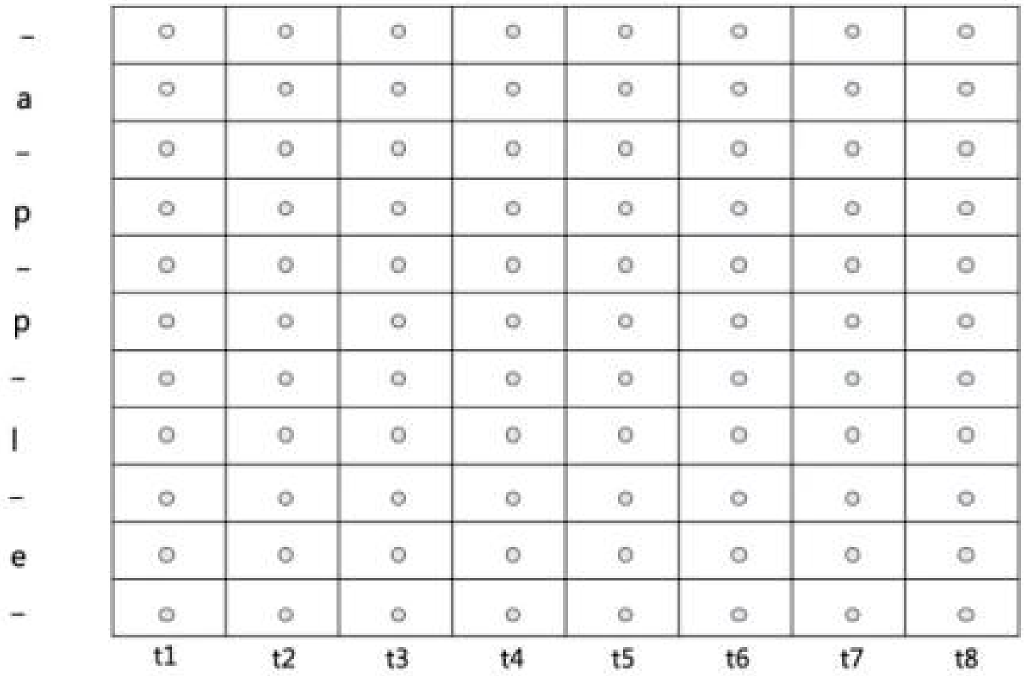
如果计算前N项斐波那契数列的和：

可以使用递归算法和迭代算法

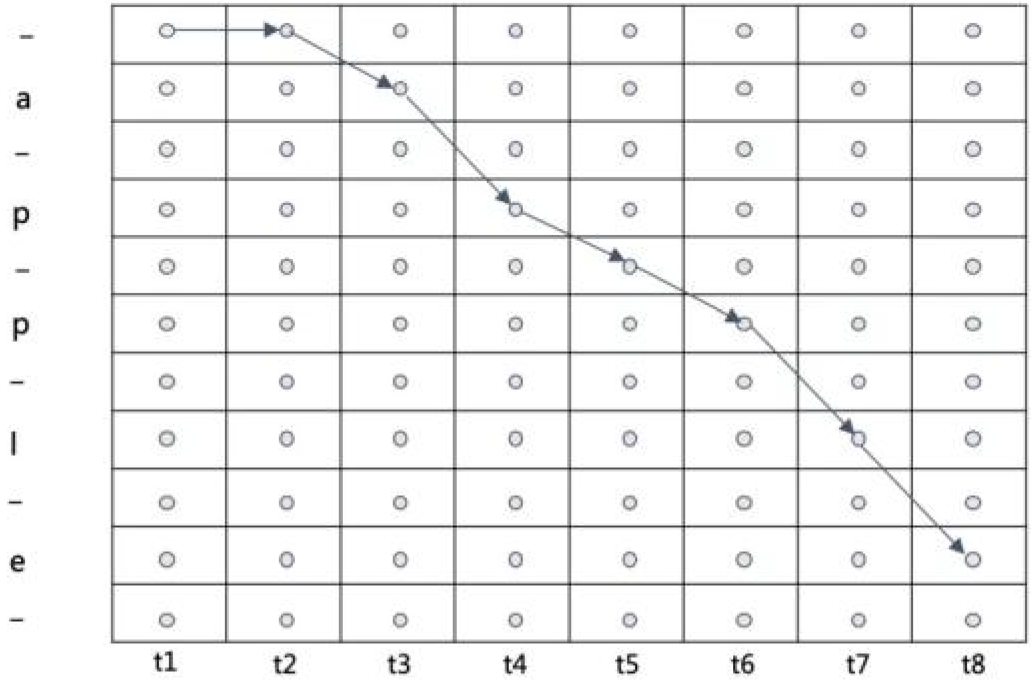
递归算法之所以速度慢，是因为它一遍又一遍地计算了相同的斐波那契数列

合法路径约束

以apple这个标签为例，按照时间序列（T=8）展开，它的路径搜索空间如下图

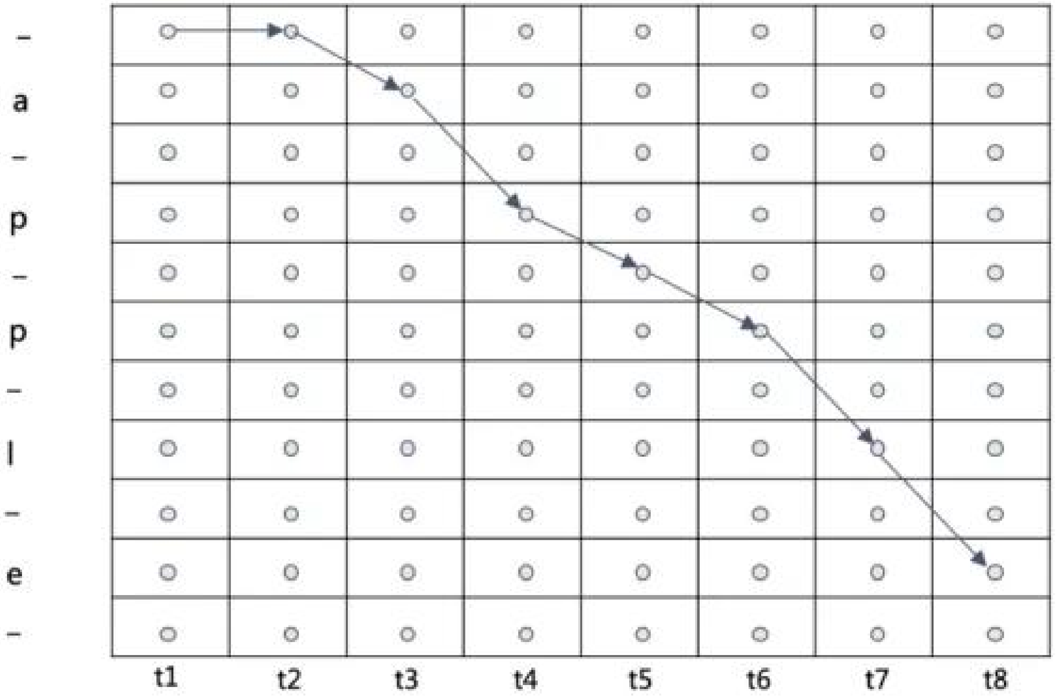


比如，B(” _ _ a p _ p l e”)= “apple” ，对应的path如下：

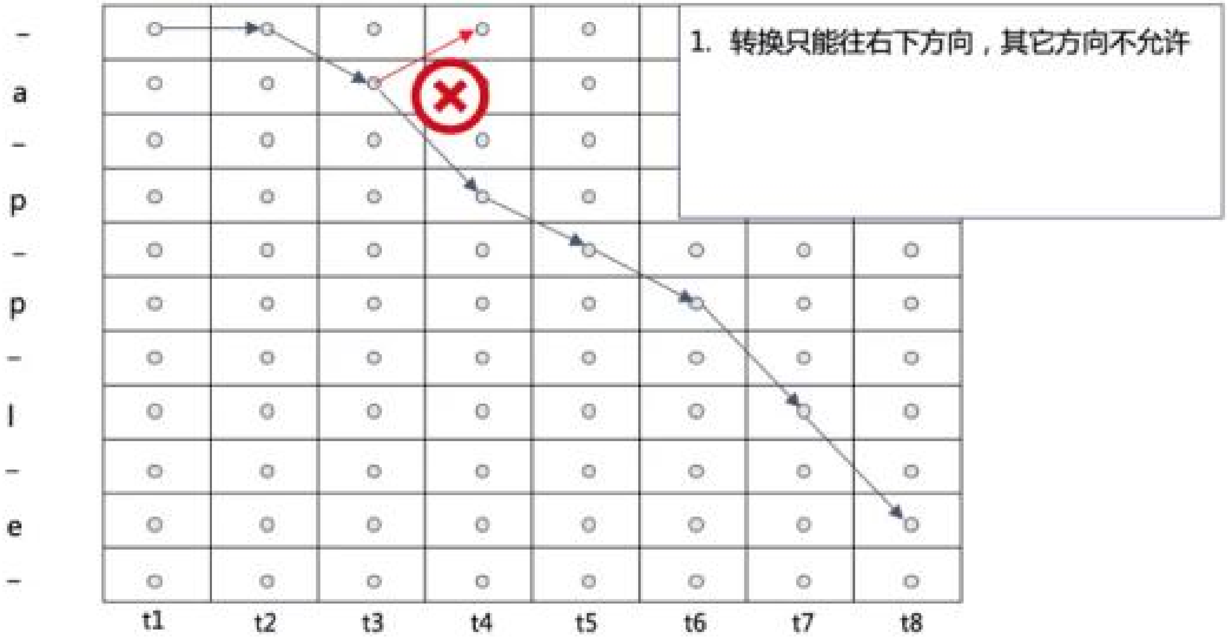


合法路径约束

P(" _ _ a p _ p l e") = $y_-^1 y_-^2 y_a^3 y_p^4 y_-^5 y_p^6 y_l^7 y_e^8$

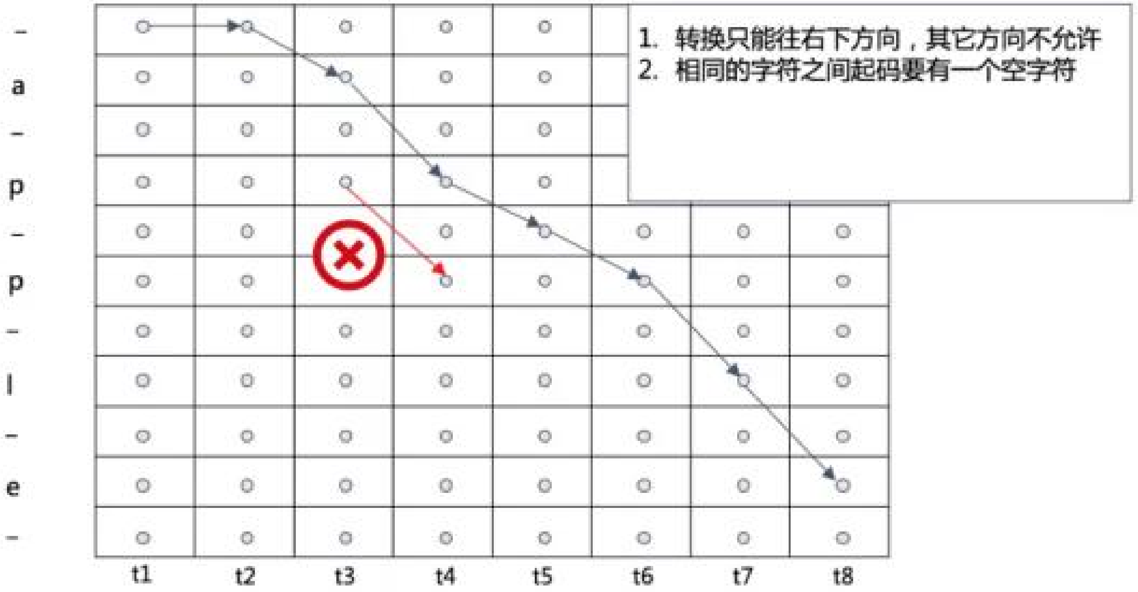


为了让所有的path都能在图中唯一、合法的表示，结点转换有如下约束：

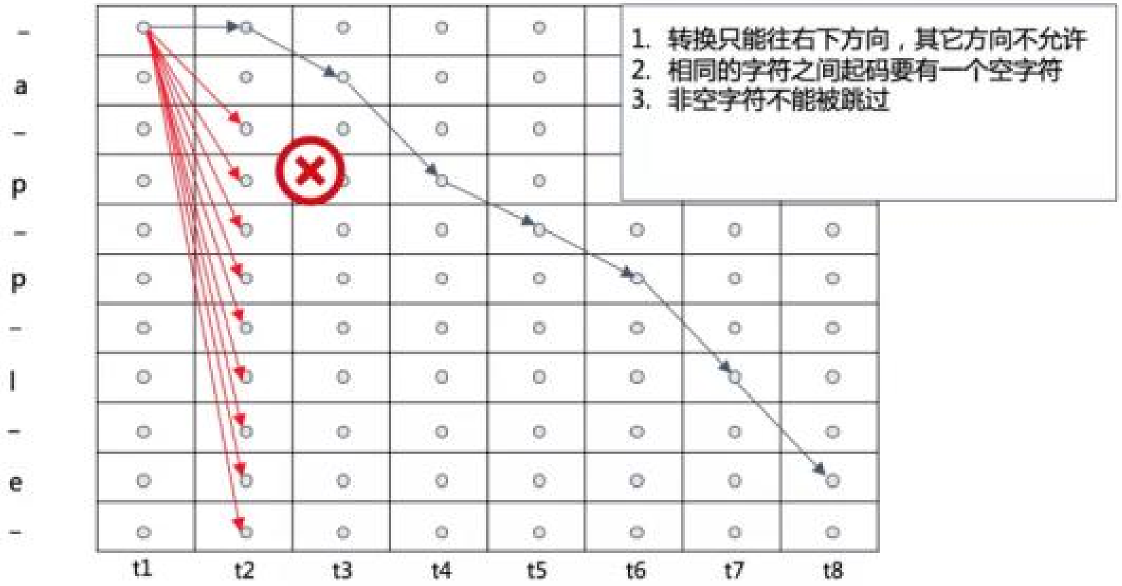


合法路径约束

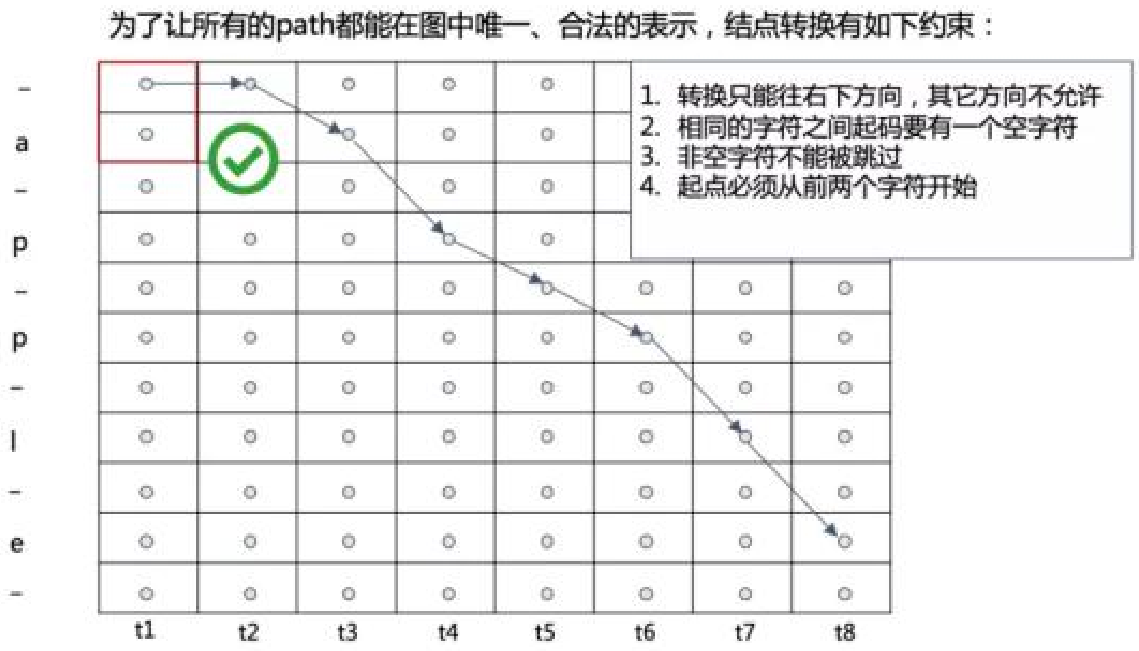
为了让所有的path都能在图中唯一、合法的表示，结点转换有如下约束：

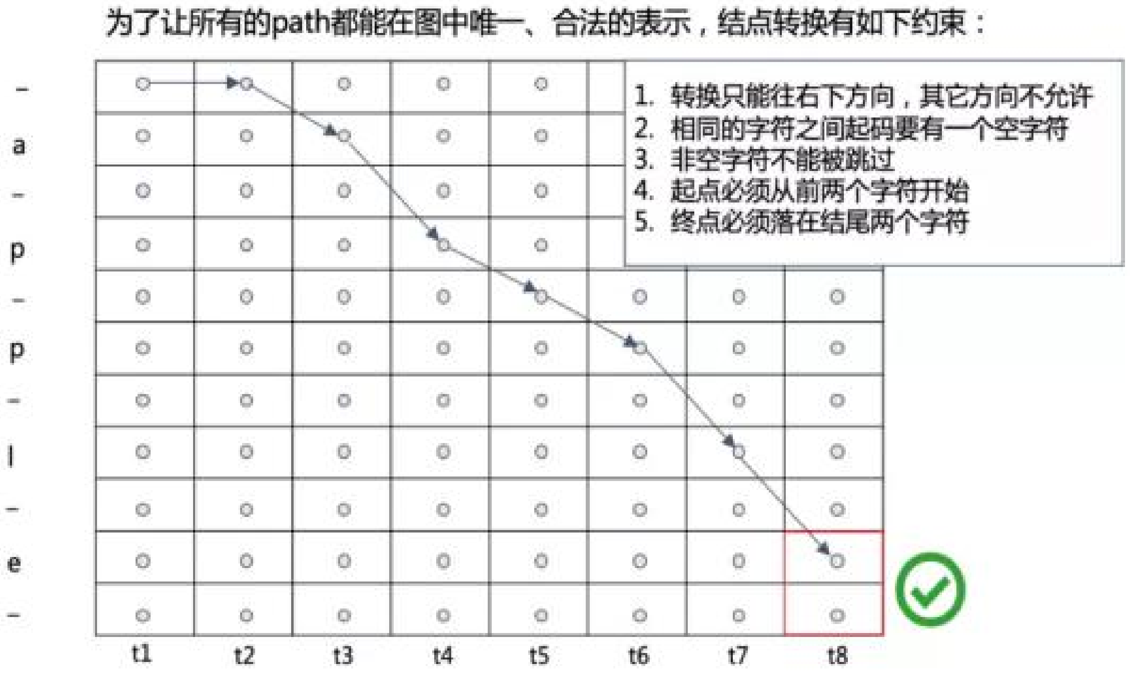


为了让所有的path都能在图中唯一、合法的表示，结点转换有如下约束：

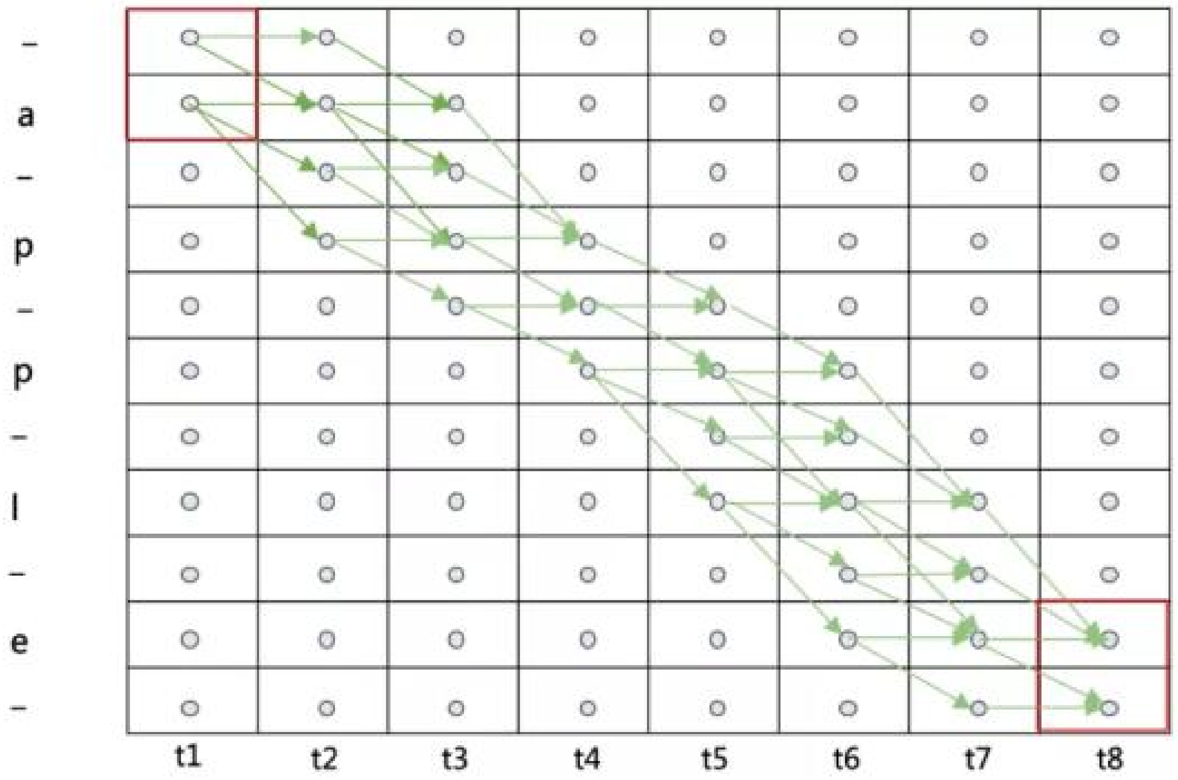


合法路径约束


为了让所有的path都能在图中唯一、合法的表示，结点转换有如下约束：
1. 转换只能往右下方向，其它方向不允许
2. 相同的字符之间起码要有一个空字符
3. 非空字符不能被跳过
4. 起点必须从前两个字符开始
-
a
-
p
-
p
-
l
-
e
-
t1
t2
t3
t4
t5
t6
t7
t8

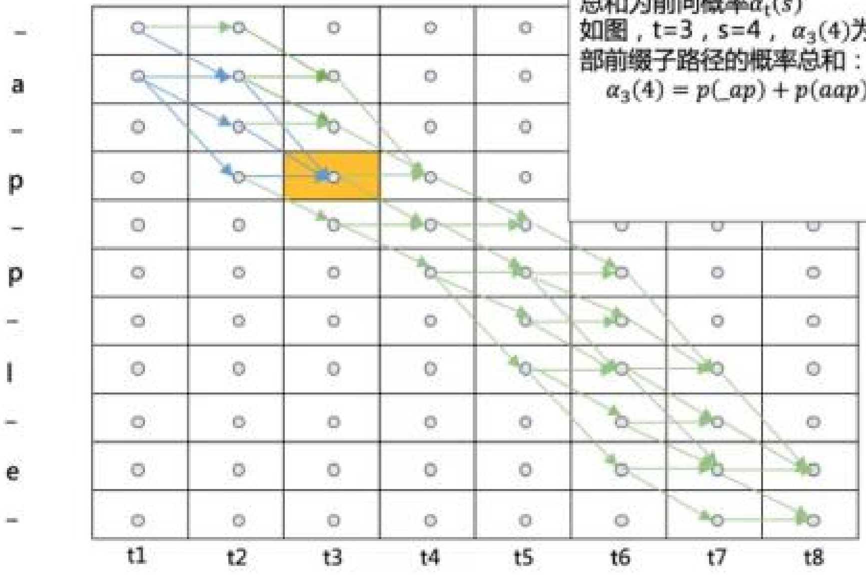

为了让所有的path都能在图中唯一、合法的表示，结点转换有如下约束：
1. 转换只能往右下方向，其它方向不允许
2. 相同的字符之间起码要有一个空字符
3. 非空字符不能被跳过
4. 起点必须从前两个字符开始
5. 终点必须落在结尾两个字符
-
a
-
p
-
p
-
l
-
e
-
t1
t2
t3
t4
t5
t6
t7
t8

合法路径约束



前向后向

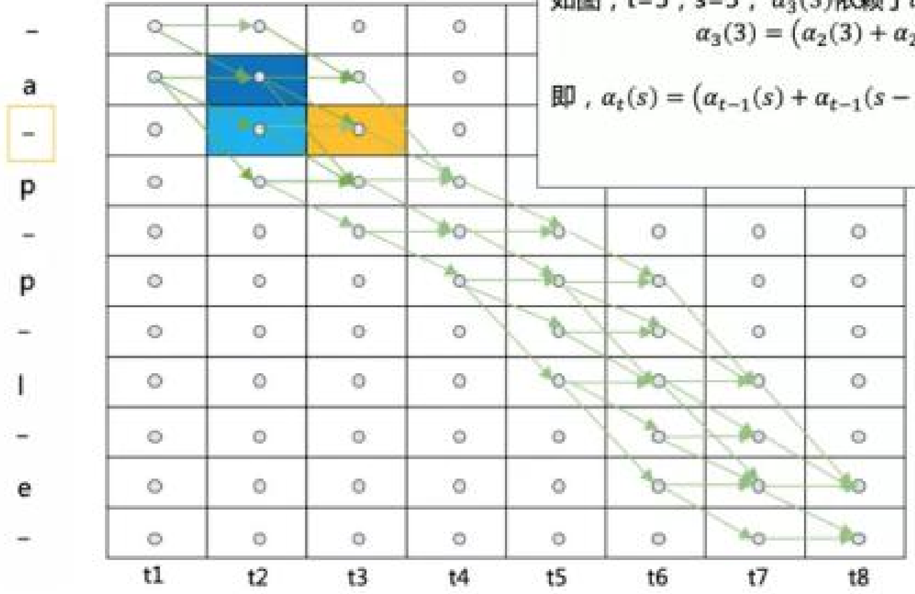
将某节点的全部路劲分为前后两部分



定义在时刻t经过节点s的全部前缀子路径的概率总和为前向概率$\alpha_t(s)$
如图，t=3，s=4，$\alpha_3(4)$为所有经过该节点的全部前缀子路径的概率总和：

$$\alpha_3(4) = p(\_ap) + p(aap) + p(a\_p) + p(app)$$

可以递归求解$\alpha_t(s)$

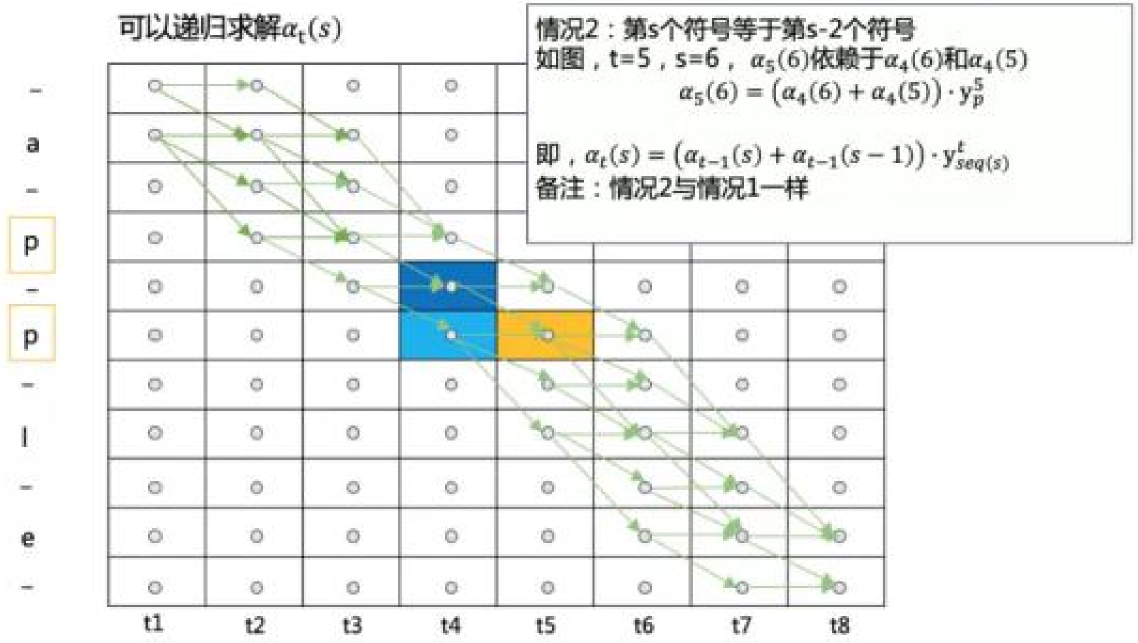


情况1：第s个符号为空符号blank
如图，t=3，s=3，$\alpha_3(3)$依赖于$\alpha_2(3)$和$\alpha_2(2)$

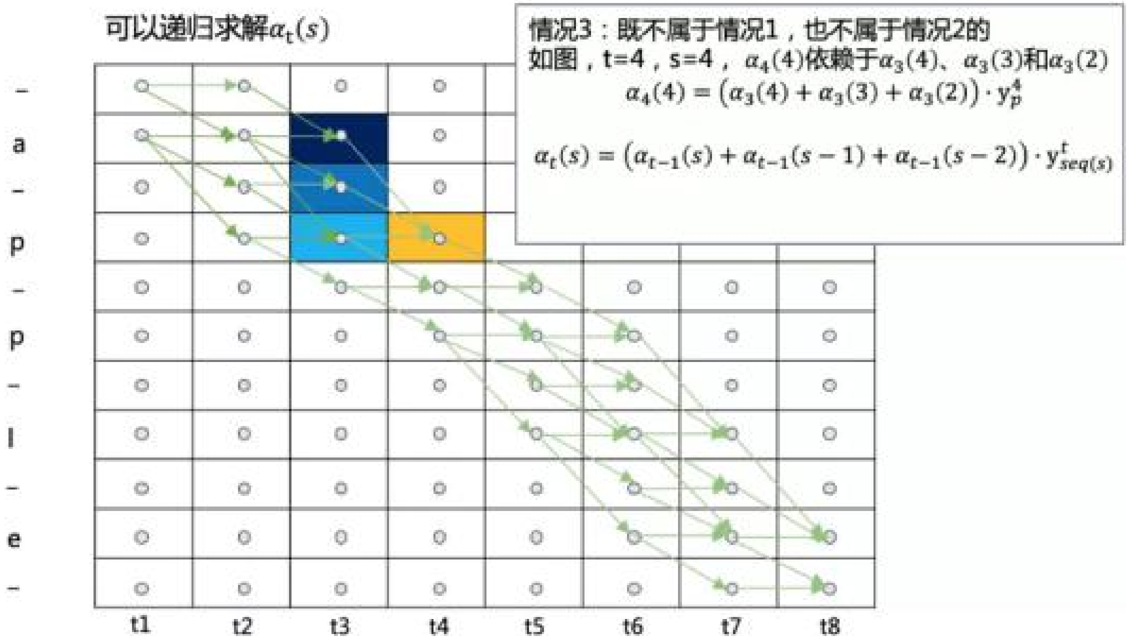
$$\alpha_3(3) = \left(\alpha_2(3) + \alpha_2(2)\right) \cdot y_{\_}^{3}$$

即，$\alpha_t(s) = \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1)\right) \cdot y_{seq(s)}^{t}$

前向后向



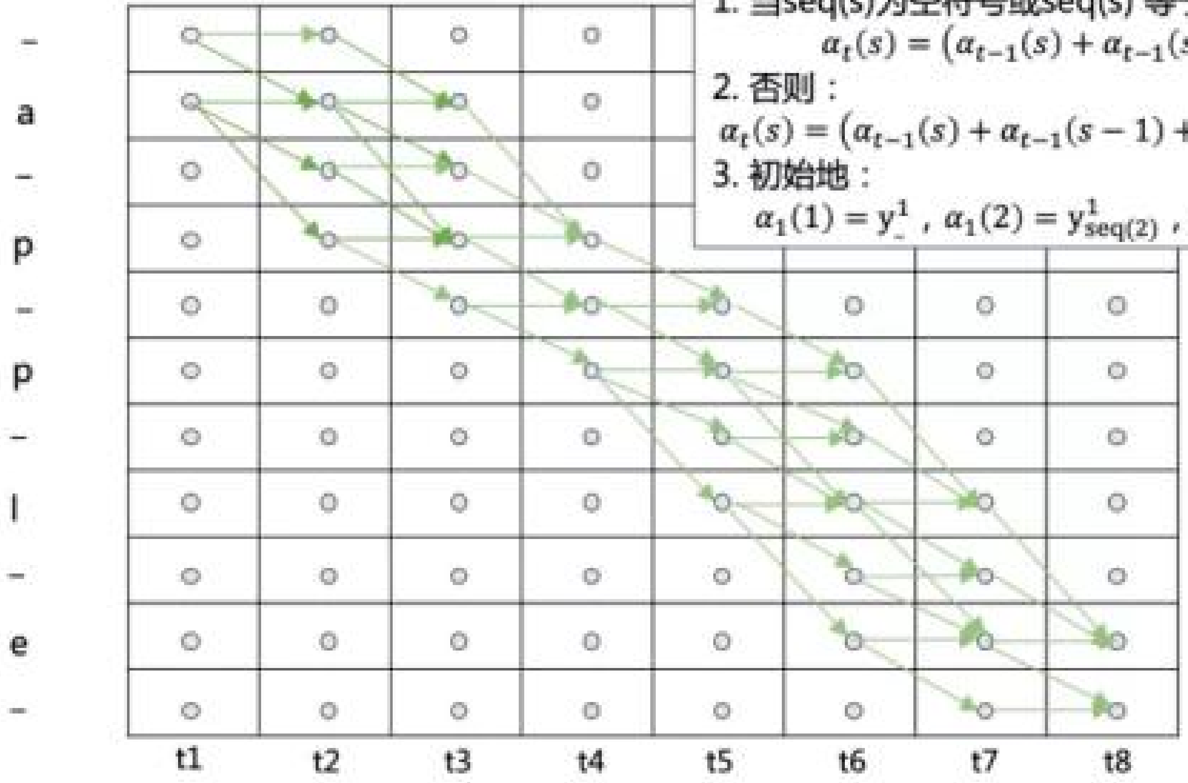
可以递归求解$\alpha_t(s)$
情况2：第s个符号等于第s-2个符号
如图，t=5，s=6， $\alpha_5(6)$依赖于$\alpha_4(6)$和$\alpha_4(5)$
$\alpha_5(6) = \left(\alpha_4(6) + \alpha_4(5)\right) \cdot y_p^5$
即，$\alpha_t(s) = \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1)\right) \cdot y_{seq(s)}^t$
备注：情况2与情况1一样
-
a
-
p
-
p
-
l
-
e
-
t1
t2
t3
t4
t5
t6
t7
t8



可以递归求解$\alpha_t(s)$
情况3：既不属于情况1，也不属于情况2的
如图，t=4，s=4， $\alpha_4(4)$依赖于$\alpha_3(4)$、$\alpha_3(3)$和$\alpha_3(2)$
$\alpha_4(4) = \left(\alpha_3(4) + \alpha_3(3) + \alpha_3(2)\right) \cdot y_p^4$
$\alpha_t(s) = \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2)\right) \cdot y_{seq(s)}^t$
-
a
-
p
-
p
-
l
-
e
-
t1
t2
t3
t4
t5
t6
t7
t8

前向后向

可以递归求解$\alpha_t(s)$



所以，可以基于动态规划算法计算$\alpha_t(s)$：

1. 当seq(s)为空符号或seq(s) 等于seq(s-2)时：

$$\alpha_t(s) = \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1)\right) \cdot y^t_{seq(s)}$$

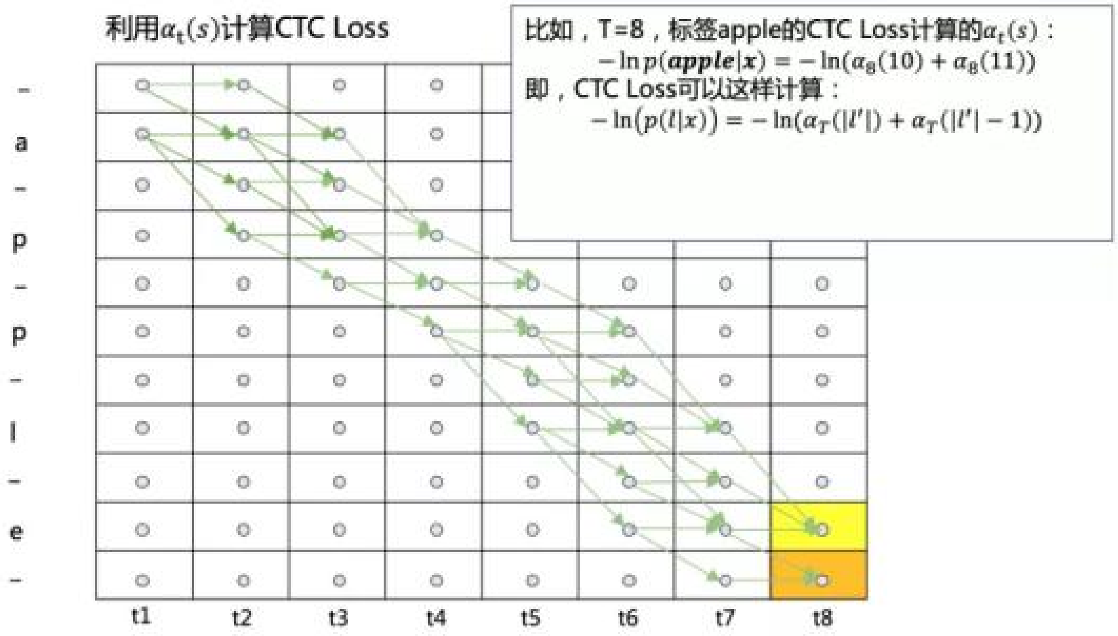
2. 否则：

$$\alpha_t(s) = \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2)\right) \cdot y^t_{seq(s)}$$

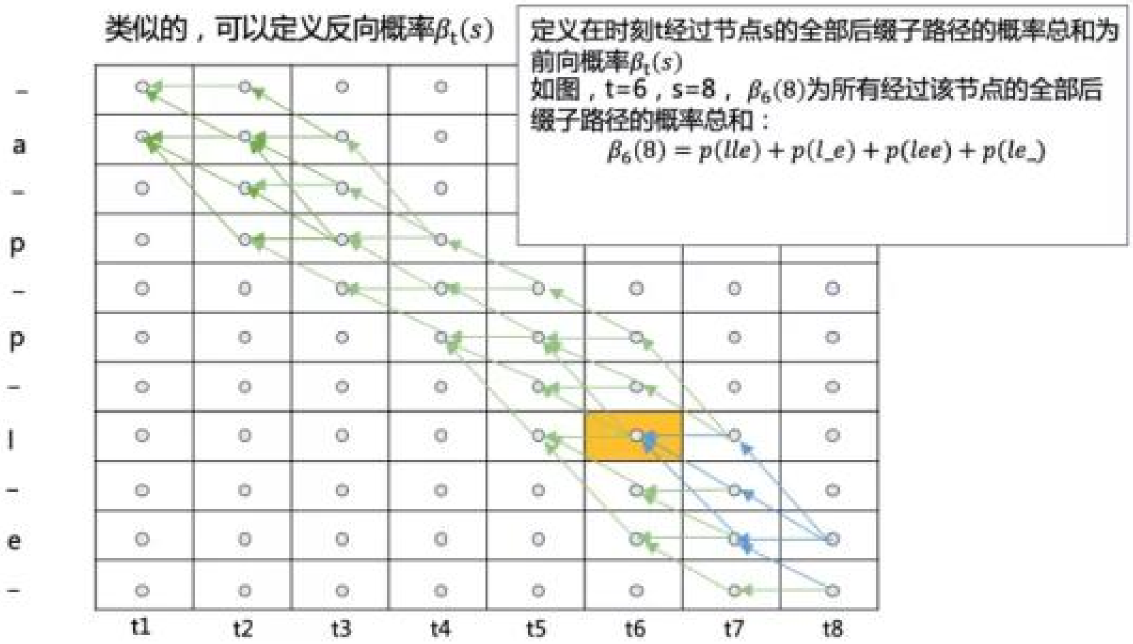
3. 初始地：

$\alpha_1(1) = y^1_-$，$\alpha_1(2) = y^1_{seq(2)}$，$\alpha_1(s) = 0$，$\forall s > 2$

前向后向

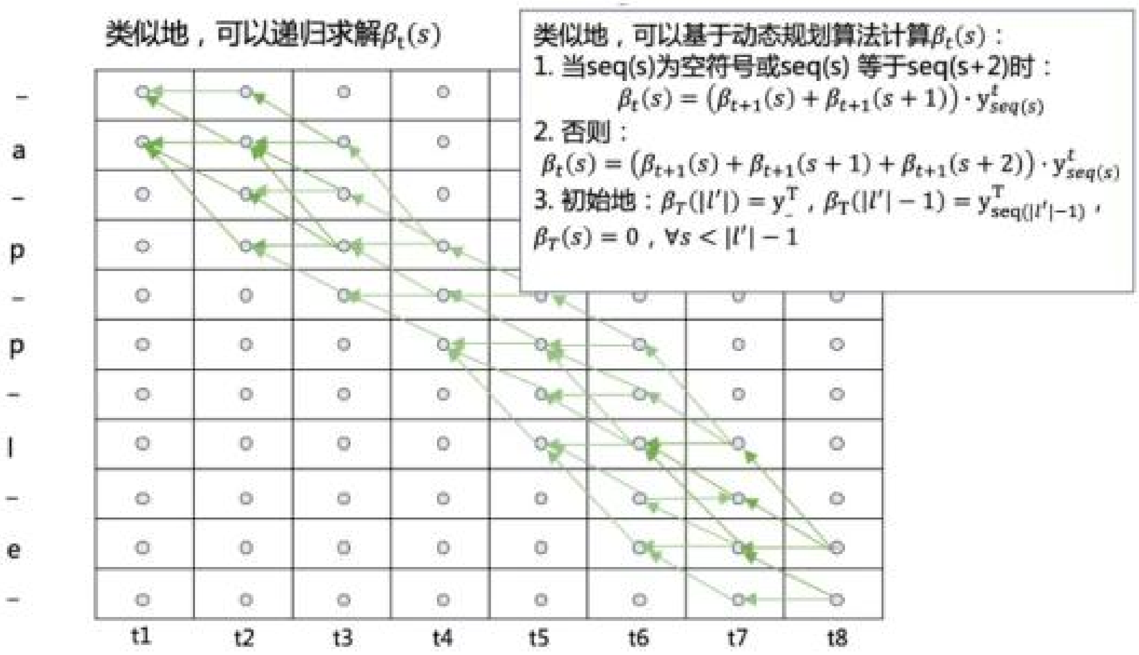


利用$\alpha_t(s)$计算CTC Loss
比如，T=8，标签apple的CTC Loss计算的$\alpha_t(s)$：
$-\ln p(\boldsymbol{apple}|\boldsymbol{x}) = -\ln(\alpha_8(10) + \alpha_8(11))$
即，CTC Loss可以这样计算：
$-\ln(p(l|x)) = -\ln(\alpha_T(|l'|) + \alpha_T(|l'|-1))$
a
p
p
l
e
t1
t2
t3
t4
t5
t6
t7
t8

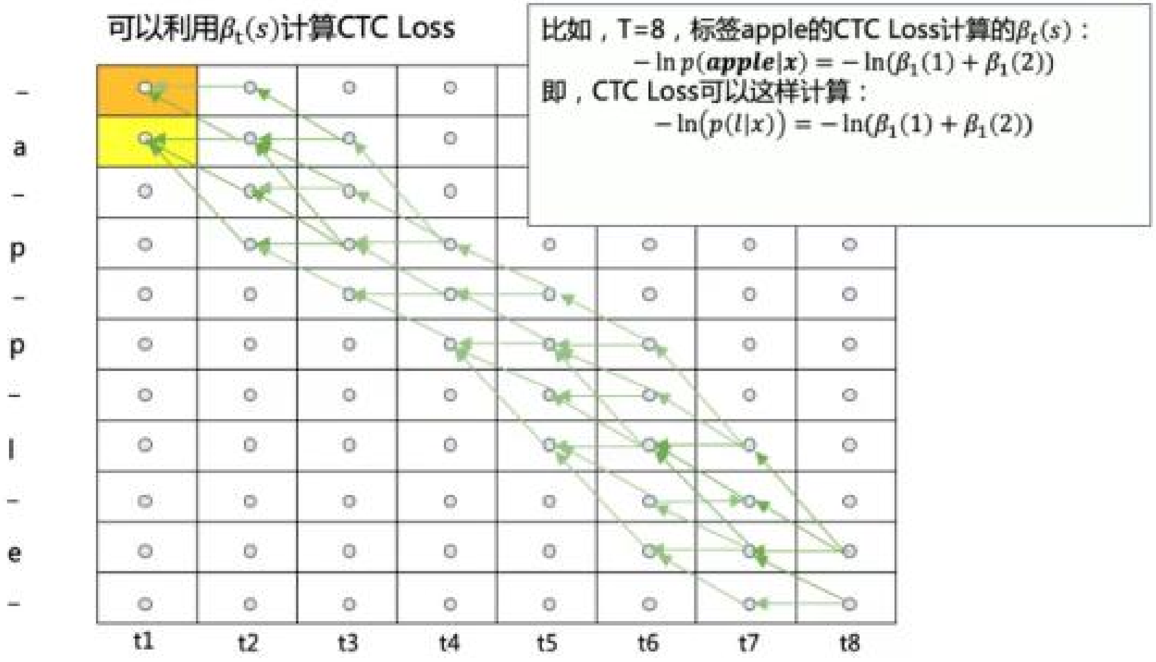


类似的，可以定义反向概率$\beta_t(s)$
定义在时刻t经过节点s的全部后缀子路径的概率总和为前向概率$\beta_t(s)$
如图，t=6，s=8， $\beta_6(8)$为所有经过该节点的全部后缀子路径的概率总和：
$\beta_6(8) = p(lle) + p(l\_e) + p(lee) + p(le\_)$
a
p
p
l
e
t1
t2
t3
t4
t5
t6
t7
t8

前向后向



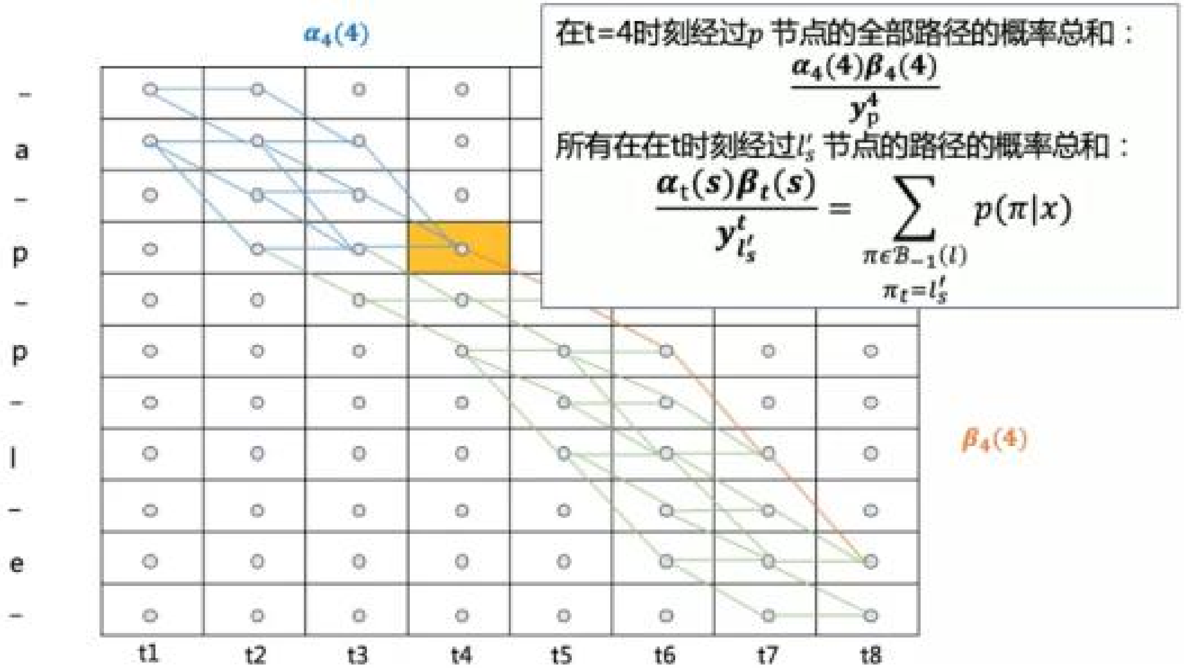
类似地，可以递归求解$\beta_t(s)$
类似地，可以基于动态规划算法计算$\beta_t(s)$：
1. 当seq(s)为空符号或seq(s) 等于seq(s+2)时：
$\beta_t(s) = \left(\beta_{t+1}(s) + \beta_{t+1}(s+1)\right) \cdot y^t_{seq(s)}$
2. 否则：
$\beta_t(s) = \left(\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2)\right) \cdot y^t_{seq(s)}$
3. 初始地：$\beta_T(|l'|) = y^T_-$，$\beta_T(|l'|-1) = y^T_{seq(|l'|-1)}$，
$\beta_T(s) = 0$，$\forall s < |l'| - 1$
\-
a
\-
p
\-
p
\-
l
\-
e
\-
t1
t2
t3
t4
t5
t6
t7
t8

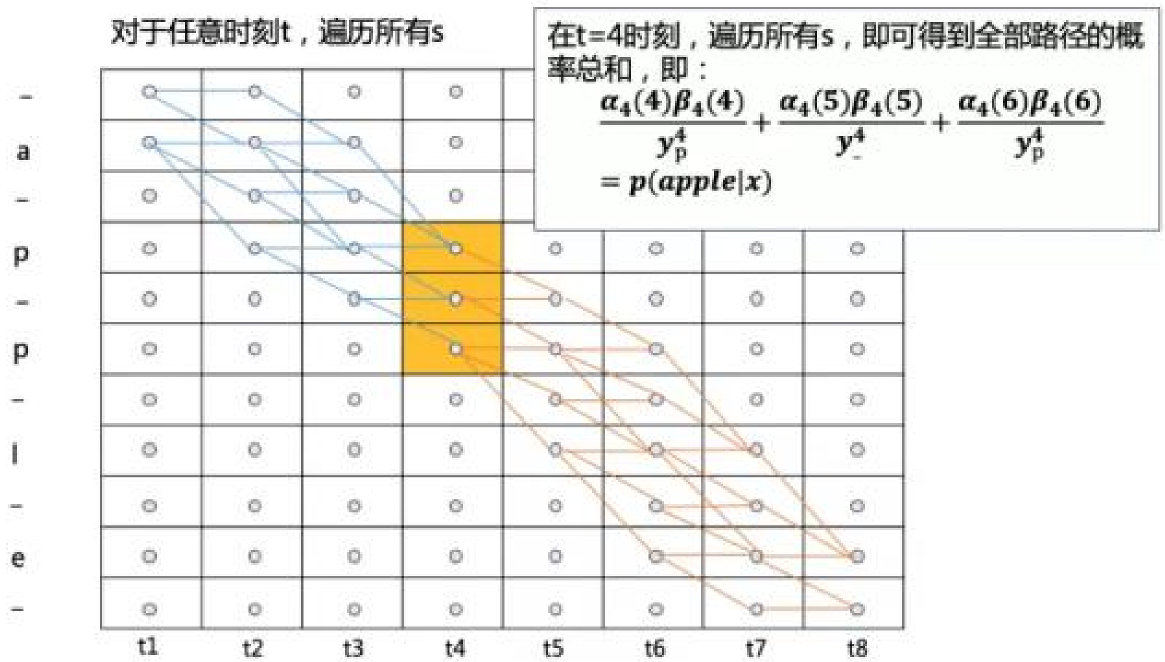


可以利用$\beta_t(s)$计算CTC Loss
比如，T=8，标签apple的CTC Loss计算的$\beta_t(s)$：
$-\ln p(\boldsymbol{apple}|\boldsymbol{x}) = -\ln(\beta_1(1) + \beta_1(2))$
即，CTC Loss可以这样计算：
$-\ln(p(l|x)) = -\ln(\beta_1(1) + \beta_1(2))$
\-
a
\-
p
\-
p
\-
l
\-
e
\-
t1
t2
t3
t4
t5
t6
t7
t8

前向后向

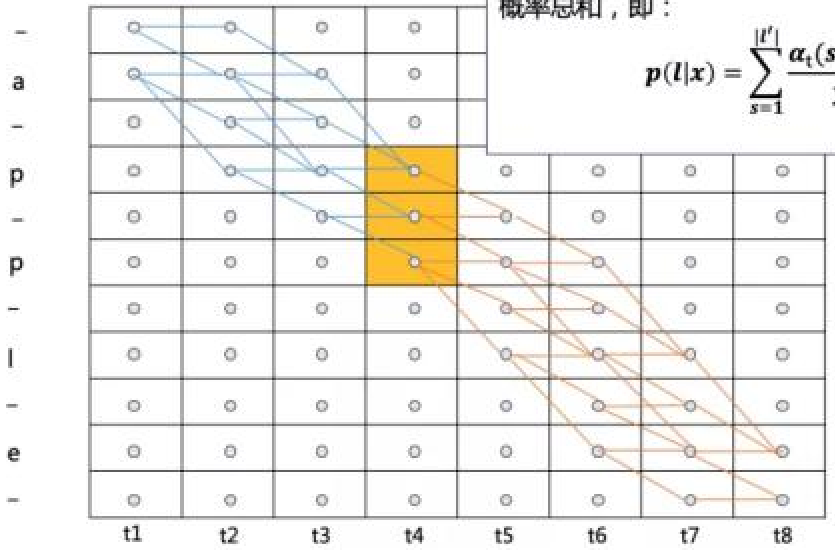
把前向概率和后向概率结合起来也可以计算CTC Loss函数





前向后向

对于任意时刻t，遍历所有s



在任意t时刻，遍历所有s，即可得到全部路径的概率总和，即：

$$p(\boldsymbol{l}|\boldsymbol{x}) = \sum_{s=1}^{|l'|} \frac{\alpha_t(s)\beta_t(s)}{y_{l'_s}^t}$$

前向、后向计算LOSS

- 对于时序长度为T的输入序列x 和输出序列z，前向概率：
  - $\alpha_t(s) = \sum_{\substack{\pi \in \mathcal{B}^{-1}(z) \\ \pi_t = l'_s}} p(\pi_{1:t}|x)$
  - $\alpha_1(1) = y^1_{\_}$，$\alpha_1(2) = y^1_{l'_2}$，$\alpha_1(s) = 0$，$\forall s > 2$
  - $\alpha_t(s) = 0$，$\forall s < |l'| - 2(T - t) - 1$
  - $\alpha_t(s) = \begin{cases} (\alpha_{t-1}(s) + \alpha_{t-1}(s-1))y^t_{l'_s} & if\ l'_s = b\ or\ l'_{s-2} = l'_s \\ (\alpha_{t-1}(s) + \alpha_{t-1}(s-1) + \alpha_{t-1}(s-2))y^t_{l'_s} & otherwise \end{cases}$
- 利用$\alpha_t(s)$计算CTC Loss：
  - $-\ln(p(l|x)) = -\ln(\alpha_T(|l'|) + \alpha_T(|l'| - 1))$

- 对于时序长度为T的输入序列x 和输出序列z，后向概率：
  - $\beta_t(s) = \sum_{\substack{\pi \in \mathcal{B}^{-1}(z) \\ \pi_t = l'_s}} p(\pi_{t:T}|x)$
  - $\beta_T(|l'|) = y^T_{\_}$, $\beta_T(|l'| - 1) = y^T_{l'_{|l'|-1}}$，$\beta_T(s) = 0\ \forall s > |l'| - 1$
  - $\beta_t(s) = 0\ \forall s > 2t\ and\ \forall s > |l'|$
  - $\beta_t(s) = \begin{cases} (\beta_{t+1}(s) + \beta_{t+1}(s+1))y^t_{l'_s} & if\ l'_s = b\ or\ l'_{s+2} = l'_s \\ (\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t-1}(s+2))y^t_{l'_s} & otherwise \end{cases}$
- 利用$\beta_t(s)$计算CTC Loss：
  - $-\ln(p(l|x)) = -\ln(\beta_1(1) + \beta_1(2))$

前向后向计算
LOSS

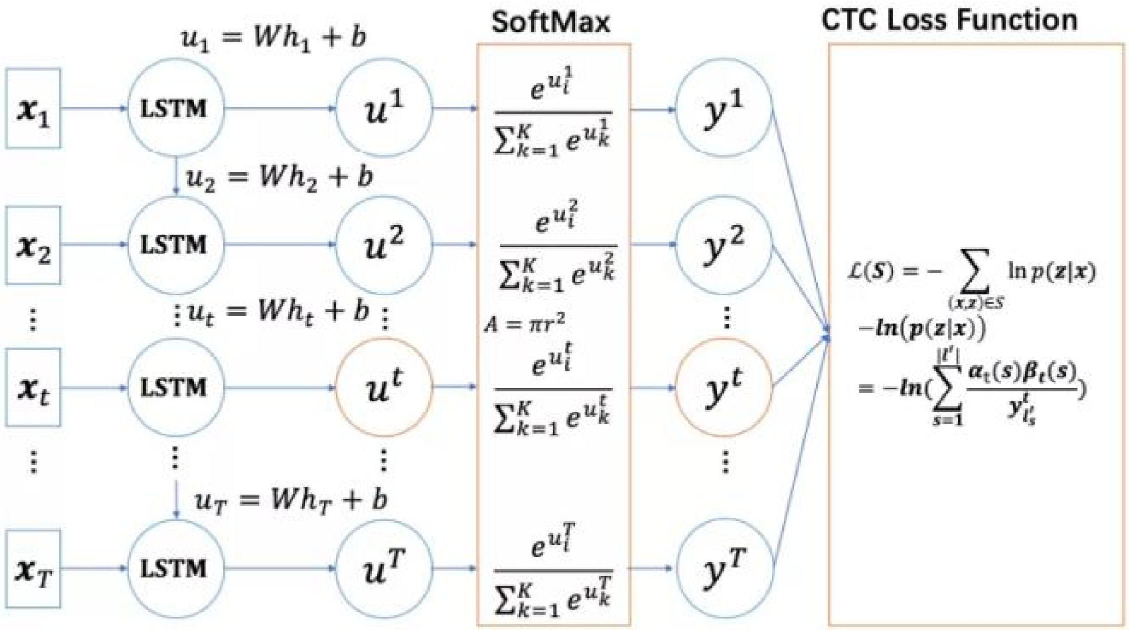
根据任意时刻的前向概率和后向概率计算CTC Loss函数，得到以下结论：

- 对于任意时刻t，利用前向概率和后向概率计算CTC Loss：

$$p(l|x) = \sum_{s=1}^{|l'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{l'_s}}$$

$$-ln\big(p(l|x)\big) = -ln\Big(\sum_{s=1}^{|l'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{l'_s}}\Big)$$

CTC Loss求导



$$p(l|x) = \sum_{s=1}^{|l'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{l'_s}}$$

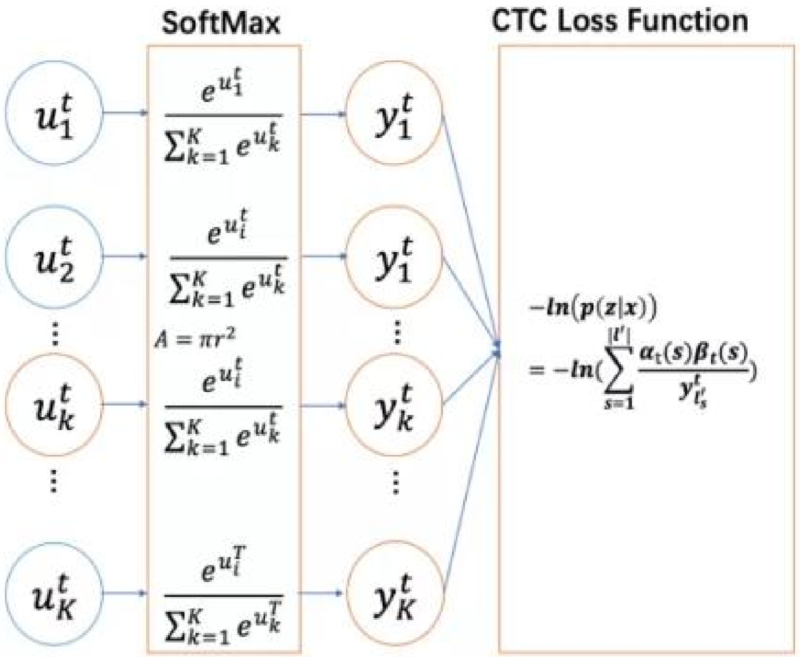
$$\frac{\partial p(l|x)}{\partial y^t_{k'}} = \frac{1}{{y^t_{k'}}^2} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s) \quad (1)$$

$$\frac{\partial ln(p(l|x))}{\partial y^t_{k'}} = \frac{1}{p(l|x){y^t_{k'}}^2} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s) \quad (2)$$

$$\frac{\partial y^t_{k'}}{\partial u^t_k} = y^t_{k'}\delta_{kk'} - y^t_{k'}y^t_k \quad (3)$$

$$-\frac{\partial ln(p(l|x))}{\partial u^t_k} = -\sum_{k'=1}^{K} \frac{\partial ln(p(l|x))}{\partial y^t_{k'}} \frac{\partial y^t_{k'}}{\partial u^t_k} \quad (4)$$

把公式(2)、(3)代入(4)得



CTC Loss求导

$$p(l|x) = \sum_{s=1}^{|l'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{l'_s}} \quad (1)$$

$$\frac{\partial p(l|x)}{\partial y^t_{k'}} = \frac{1}{{y^t_{k'}}^2} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s)$$

$$\frac{\partial ln(p(l|x))}{\partial y^t_{k'}} = \frac{1}{p(l|x){y^t_{k'}}^2} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s) \quad (2)$$

$$\frac{\partial y^t_{k'}}{\partial u^t_k} = y^t_{k'}\delta_{kk'} - y^t_{k'}y^t_k \quad (3)$$

$$-\frac{\partial ln(p(l|x))}{\partial u^t_k} = -\sum_{k'=1}^{K} \frac{\partial ln(p(l|x))}{\partial y^t_{k'}}\frac{\partial y^t_{k'}}{\partial u^t_k} \quad (4)$$

把公式(2)、(3)代入(4)得

$$-\frac{\partial ln(p(l|x))}{\partial u^t_k} = -\sum_{k'=1}^{K} \frac{\partial ln(p(l|x))}{\partial y^t_{k'}}\frac{\partial y^t_{k'}}{\partial u^t_k}$$

$$= \sum_{k'=1}^{K} \frac{y^t_k - \delta_{kk'}}{p(l|x)y^t_{k'}} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s)$$

$$= -\sum_{k'=1}^{K} \frac{y^t_{k'}\delta_{kk'} - y^t_{k'}y^t_k}{p(l|x){y^t_{k'}}^2} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s)$$

$$= \sum_{k'=1}^{K} \frac{y^t_k}{p(l|x)y^t_{k'}} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s) - \sum_{k'=1}^{K} \frac{\delta_{kk'}}{p(l|x)y^t_{k'}} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s)$$

$$= \frac{y^t_k}{p(l|x)} \boxed{\sum_{k'=1}^{K} \frac{1}{y^t_{k'}} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s)} - \sum_{k'=1}^{K} \frac{\delta_{kk'}}{p(l|x)y^t_{k'}} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s)$$

$$= \frac{y^t_k}{p(l|x)} \boxed{p(l|x)} - \sum_{k'=1}^{K} \frac{\delta_{kk'}}{p(l|x)y^t_{k'}} \sum_{s\in lab(l,k')} \alpha_t(s)\beta_t(s)$$

$$= y^t_k - \frac{1}{p(l|x)y^t_k} \sum_{s\in lab(l,k)} \alpha_t(s)\beta_t(s)$$

BeamSearch

Beam Search的过程非常简单，每一步搜索选取概率最大的W个节点进行扩展，W也称为Beam Width，其核心还是计算每一步扩展节点的概率

	t1	t2	t3
b	0.3	0.3	0.1
a	0.2	0.3	0.3
_	0.5	0.4	0.6

横轴表示时间，纵轴表示每一步输出层的概率，T=3，字符集为{a, b}

穷举搜索

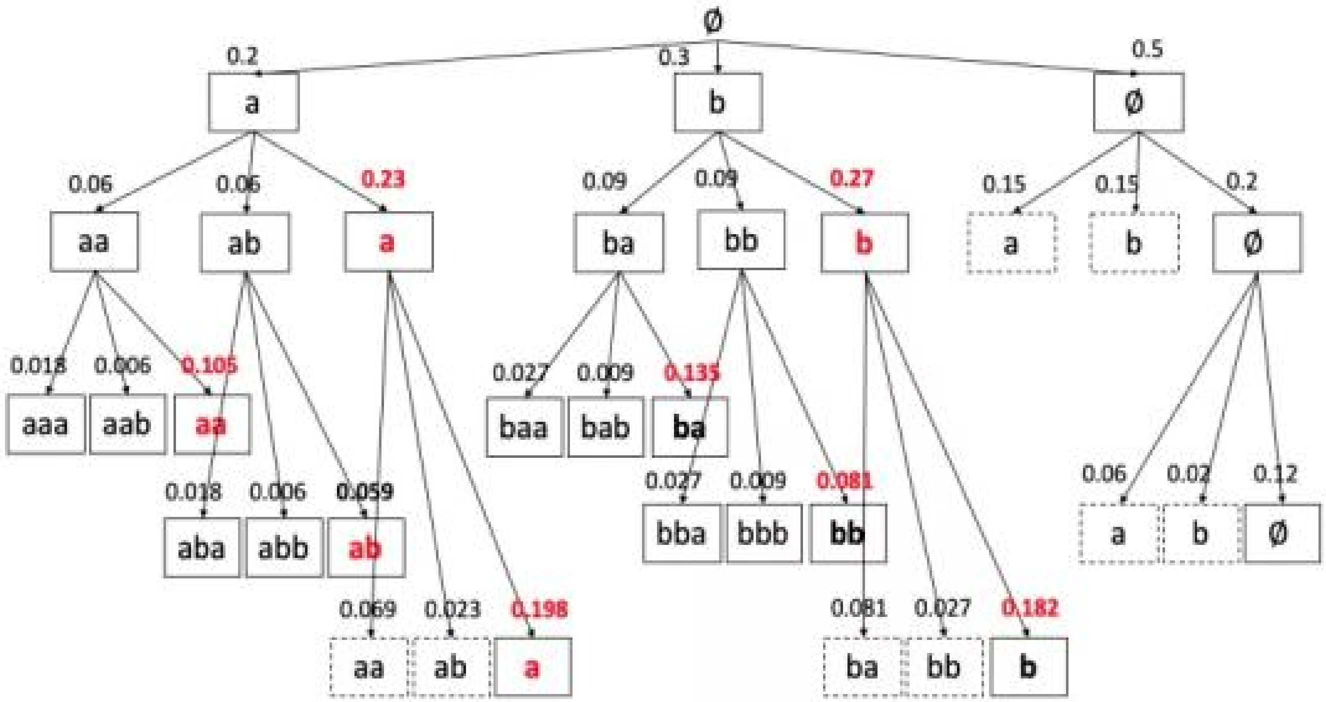
Beam Search的过程非常简单，每一步搜索选取概率最大的W个节点进行扩展，W也称为Beam Width，其核心还是计算每一步扩展节点的概率

	t1	t2	t3
b	0.3	0.3	0.1
a	0.2	0.3	0.3
_	0.5	0.4	0.6

横轴表示时间，纵轴表示每一步输出层的概率，T=3，字符集为{a, b}

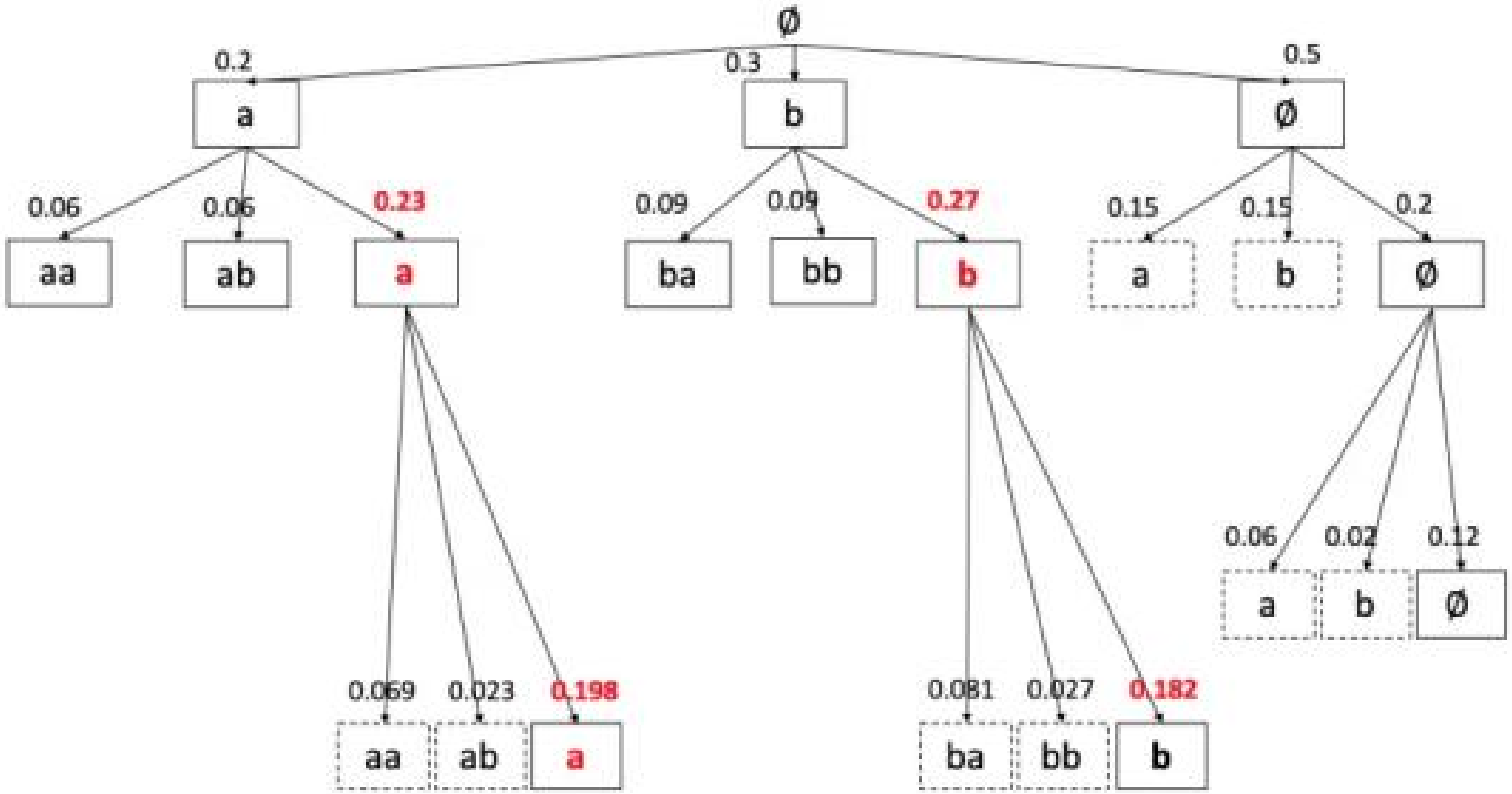
穷举搜索

如果对它的搜索空间进行穷举搜索，则每一步都展开进行搜索



BamSearch

穷举搜索每一步都要扩展全部节点，能保证最终找到最优解（上图中例子最优解l*=b，p(l*)=0.164），但搜索复杂度太高，而Beam Search的思路很简单，每一步只选取扩展概率最大的W个节点进行扩展



感谢聆听