Lecture 9: 大词表连续语音识别（LVCSR）

Kai Yu and Yanmin Qian

Cross Media Language Intelligence Lab (X-LANCE)
Department of Computer Science & Engineering
Shanghai Jiao Tong University

Spring 2021

目录

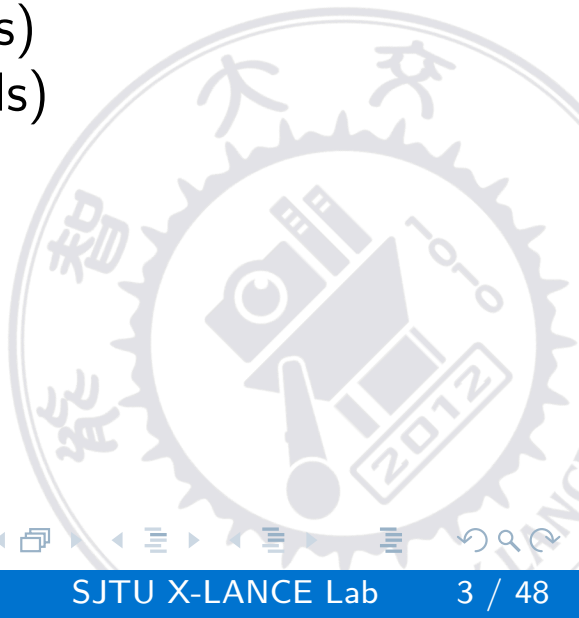
- 声学单元和上下文相关建模
- 状态聚类
- Two model re-estimation 与 single pass re-training
- LVCSR 解码
  - Token Passing Algorithm
  - Beam Search
- 强制对齐
- 词图 (Lattices) 和多候选

大词表连续语音识别 - LVCSR

- **大词表:** 大于 10,000 词，甚至 100,000 词
- **连续语音:** 自然且连续的正常说话类型
- **无语法限制:** 大的 N-gram 语言模型
- **类型多:** 不同的说话人, 混杂的环境

挑战:

- 使用大量多变的数据进行训练
  - 语音数据: 成百上千小时 (hundreds of hours)
  - 文本数据: 成千上亿的词串 (billions of words)
- 解码算法和训练算法的有效性
  - 在声学模型中，成百上千的 Gaussians
  - 在解码网络中，成千上亿的连接关系



声学单元

用 HMM 可以建模什么?

必要条件:

- **规模紧凑:** 针对大词汇的任务，声学单元的数量是有限的。
- **简单的映射关系:** 在声学单元和词之间具有简单的映射关系
- **语音的变化:** 可以被有效的建模和表示 (口音/协同发音)
- **没有出现过的词:** 可以很容易的被处理
- **定义完善:** 允许大数据库的使用

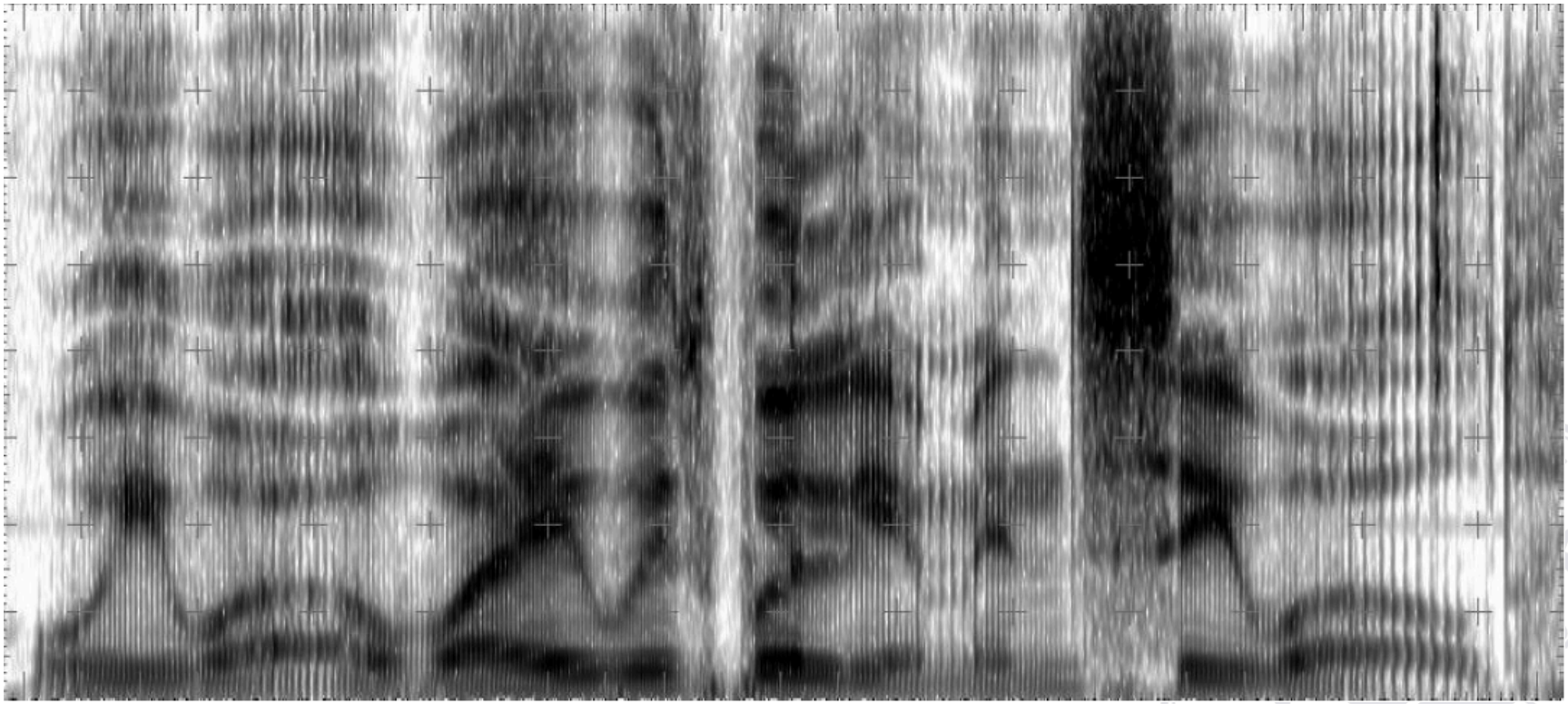
典型的声学单元:

- **词:** 对 LVCSR 不适合 (规模不紧凑, 词内部的语音变化很难去建模, 训练集中没有出现过的词很难处理)
- **子词:** 在 LVCSR 中用的很流行
  - **音素 - Phones:** 最小可辨识的发音声学单元。较高的上下文依赖关系。定义完善。
  - **音节 - Syllables:** 在英语中 $>$10000, 在汉语中 $>$400。得到好的参数估计很困难。

音素变化

协同发音以及上下文的影响

协同发音: 某一个音素在一个特定的音素上下文中的具体发音，受到此特定上下文音素的影响。



We were away with William in Sea World

w 的表现在变化，但相同的模式出现在相同的**上下文**中

上下文相关音素

每一个 HMM 被用来对一个上下文相关音素进行建模。比如，一个孤立词 *SPEECH*:

Context	*SPEECH*	# Contexts
Monophone	**/s p iy ch/**	46
Biphone (L)	**/sil-s s-p p-iy iy-ch/**	2116
Biphone (R)	**/s+p p+iy iy+ch ch+sil/**	2116
Triphone	**/sil-s+p s-p+iy p-iy+ch iy-ch+sil/**	97336
Pentaphone	**/sil-sil-s+p+iy sil-s-p+iy+ch s-p-iy+ch+sil p-iy-ch+sil+sil/**	205962976

实际上，很多上下文关系并不存在 (比如，一个词包含如下上下文音素 z-z-z+z+z!!!).

词的边界信息

词内 (Within word) 上下文相关三音子模型 (triphone):

speech task = / **sil s+p s-p+iy p-iy+ch iy-ch t+ae t-ae+s ae-s+k s-k sil**/

跨词 (Cross word) 上下文相关三音子模型 (triphone):

speech task = / **sil sil-s+p s-p+iy p-iy+ch iy-ch+t ch-t+ae t-ae+s ae-s+k s-k+sil sil** /

数据稀疏问题: 一个 26,000 英文单词的 WSJ 数据库字典:

Triphone	Distinct contexts
Within-word	14300
Cross-word	54400
All possible	97336

使用 HTK-HLEd 进行 Context Expansion

HLEd -i <OUTPUT MLF> <EDIT FILE> <INPUT MLF>
其中 *edit* 文件可以按照如下书写

```
NB sp              # specify that sp is not a inter-word boundary
TC                 # expand triphone context
RE sil sil         # rename *-sil+* as sil (since context is ignored)
RE sp sp           # rename *-sp+* as sp (since context is ignored)
ME sil sil sil     # merge consecutive silences
```

```
#!MLF!#
"sentence.lab"
sil
a
b
sp
c
sil
.
```

⇒

```
#!MLF!#
"sentence.lab"
sil
sil-a+b
a-b+c
sp
b-c+sil
sil
.
```

使用 HTK-HLEd 进行模型克隆和参数绑定

需要将上下文无关单音子声学模型 (CI-monophone) 转换为上下文相关 (CD) 的声学模型:

- **克隆**: CD 模型从 CI 模型克隆得到 (e.g. clone a-b+c and d-b+e using b)
- **绑定**: 所有 CD 模型的转移概率矩阵都共享具有相同中心音子 (centerphones) 的转移概率矩阵

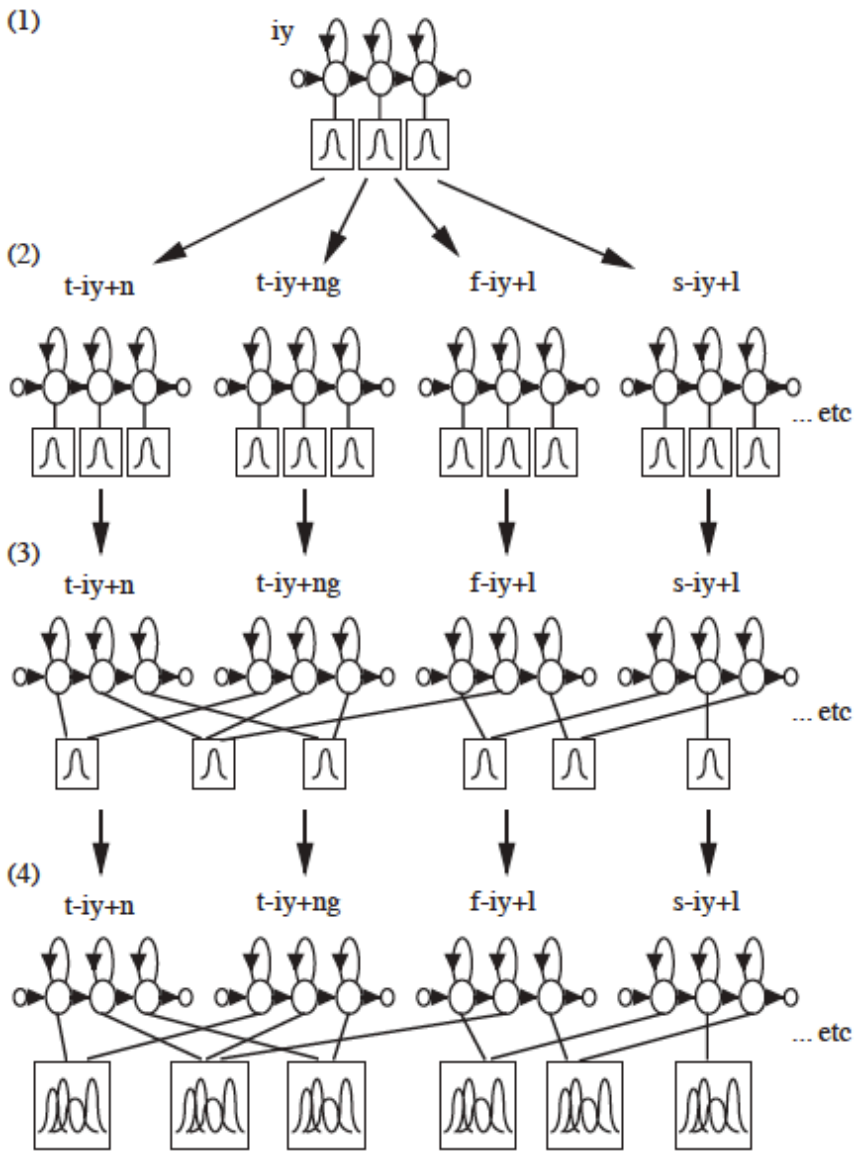
```
HHEd -H <CI_MODEL> -w <CD_MODEL> <EDIT FILE> <MODEL_LIST>
```

其中 *edit* 文件可以按照如下书写

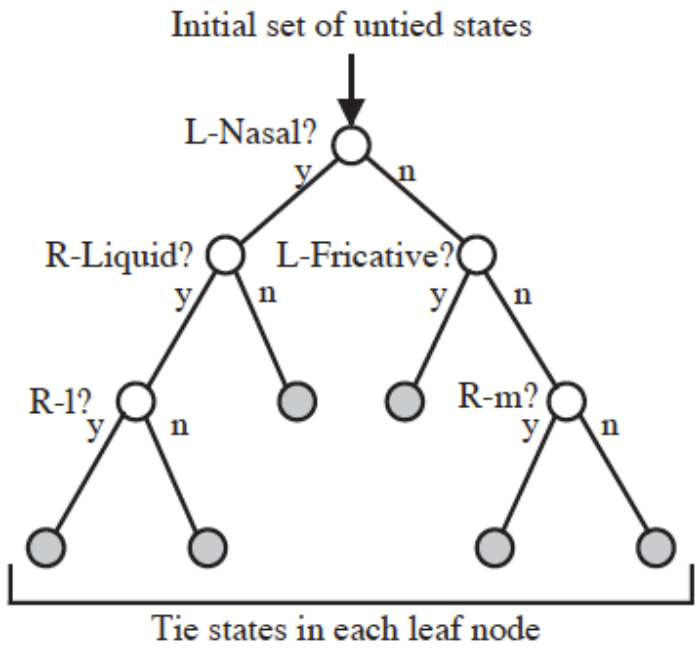
```
CL    context_dependent_model_list
TI    T_a {(*-a+*,a+*,*-a).transP}
TI    T_b {(*-b+*,b+*,*-b).transP}
...
```

参数绑定 是保持模型规模的一个重要的方法。

决策树聚类 - Decision Tree Clustering



- 状态层 **(State-level)** 高斯参数的绑定
- 问题是对音素上下文提的



决策树聚类

细节

- 自上至下的二叉树聚类
- 每一个节点的数据使用单高斯来描述
- 在每一次分裂的Gain都用 似然度增长来定义

步骤:

1. 初始化: 将所有数据放在一块，然后计算根节点的似然度
2. For 每一个叶子节点, do
   - 2.1 For 每一个问题, 计算在节点分裂后的似然增长
   - 2.2 选择似然度增长最大的问题
   - 2.3 假如似然度增加超过了预先定义的门限，使用所选择的问题完成相应的分裂
3. 最后, 检查每一个叶子节点的“occupancy counts”，同时合并那些“occupancy counts”较小的节点

假定状态聚类没有改变帧-状态间的对准

$$\mathcal{L}(\mathbf{S}) = \sum_{f \in F} \sum_{s \in \mathbf{S}} \log p(\mathbf{o}; \boldsymbol{\mu}(\mathbf{S}), \boldsymbol{\Sigma}(\mathbf{S})) \gamma_s(\mathbf{o}_f)$$

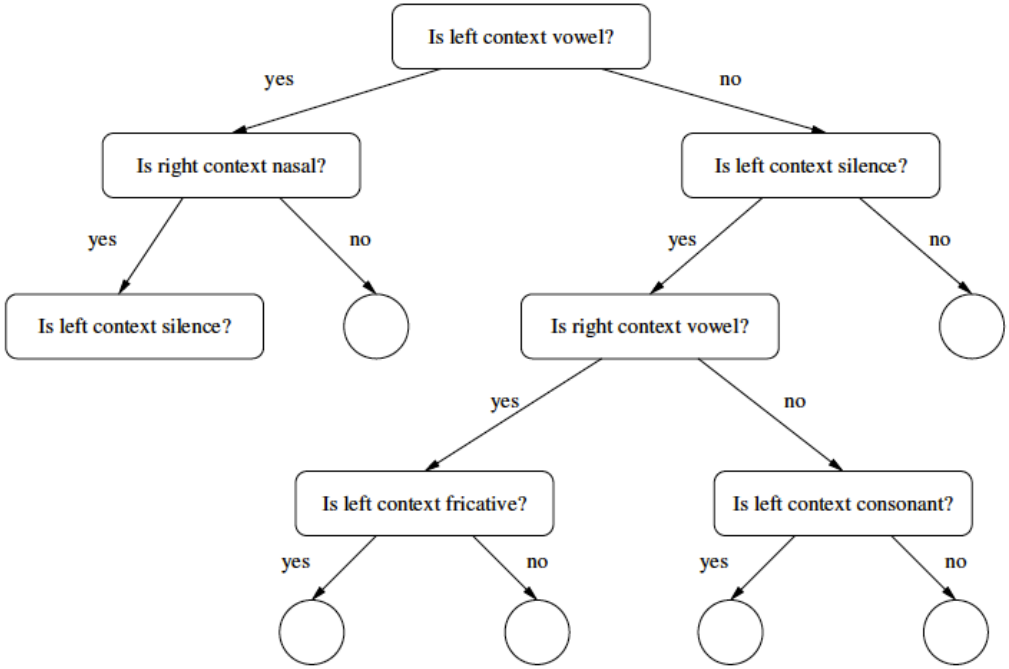
假如状态输出分布是高斯分布

$$\mathcal{L}(\mathbf{S}) = -\frac{1}{2}(\log[(2\pi)^n |\boldsymbol{\Sigma}(\mathbf{S})|)] + n) \sum_{f \in F} \sum_{s \in S} \gamma_s(\mathbf{o}_f)$$

找到最优问题, 来最大化:

$$\Delta \mathcal{L}_q = \mathcal{L}(\mathbf{S}_y(q)) + \mathcal{L}(\mathbf{S}_n(q)) - \mathcal{L}(\mathbf{S})$$

使用 HTK-HHEd 进行决策树聚类




```
QS  "R_Silence"         { *+sil }
QS  "R_Stop"            { *+p,*+pd,*+b,*+t,*+td,*+d,*+dd,*+k,*+kd,*+g }
QS  "R_Nasal"           { *+m,*+n,*+en,*+ng }
QS  "R_Fricative"       { *+s,*+sh,*+z,*+f,*+v,*+ch,*+jh,*+th,*+dh }
QS  "R_y"               { *+y }
QS  "R_z"               { *+z }
...
QS  "L_Silence"         { sil-* }
QS  "L_Stop"            { p-*,pd-*,b-*,t-*,td-*,d-*,dd-*,k-*,kd-*,g-* }
QS  "L_Nasal"           { m-*,n-*,en-*,ng-* }
QS  "L_Fricative"       { s-*,sh-*,z-*,f-*,v-*,ch-*,jh-*,th-*,dh-* }
QS  "L_y"               { y-* }
QS  "L_z"               { z-* }
```

使用 HTK-HHEd 进行决策树聚类

为每一中心音子**center phone**的每一状态**state**构建一颗决策树

```
TI    f "ST_y_2_" {("y","*-y+*","y+*","*-y").state[2]}
TI    f "ST_z_2_" {("z","*-z+*","z+*","*-z").state[2]}
...
TI    f "ST_z_3_" {("z","*-z+*","z+*","*-z").state[3]}
...
RO    f stats_file
AU    unseen_models
CO    compact_models
ST    decision_trees
```

- f: 是一个预先定义好的分裂阈值 (调整树的深度)
- RO: 移除那些 counts 数小于 f 的异常值
- Counts 存储在 stats 文件中 (可以通过使用 HERest 和 '-s' 选项来得到)
- AU: 给所有在训练数据中从没见过的模型做状态决策树聚类
- CO: 压缩 unseen models 到逻辑模型列表 (那些具有相同状态的模型)
- ST: 保存最终优化的决策树

使用 HTK-HHEd 进行决策树聚类

```
HHEd -H <INPUT MODEL> -w <OUTPUT MODEL> <EDIT FILE> <SEEN MODEL LIST>
```

```
RO f stats_file
TR 0
QS ...
...
TR 2
TB ...
TR 1
AU unseen_models
CO compact_models
ST decision_trees
```

⇒

```
...
TB 400.0 ST_y_4_ {}
 Tree based clustering
 Start  y[4] : 163  have LogL=-83.175 occ=13083.6
 Via    y[4] : 10 gives LogL=-81.556 occ=13083.6
 End    y[4] : 10 gives LogL=-81.556 occ=13083.6
 TB: Stats 163->10 [6.1%]  { 50262->4587 [9.1%] total }

AU clustering/unseen
 Creating HMMset using trees to add unseen triphones
 AU: 65561 Log/65561 Phys created from 16756 Log/16756 Phys

CO xwrd.clustered.mlist
 Create compact HMMList
 CO: HMMs 65561 Logical 17104 Physical
...
```

通过调节 RO 和 TB 来调整状态数量

使用 Kaldi 进行决策树聚类

计算决策树聚类过程中需要的统计量：

```
acc-tree-stats --ci-phones=<ciphonelist> <mono-model> \
               <feats> <ali> <treeacc-out>
```

进行聚类：

```
cluster-phones <treeacc> <phone-sets-in> \
               <clustered-phones-out>
compile-questions <topo> <questions-text-file> \
                  <questions-out>
```

通过指定最大叶子数量来构建决策树：

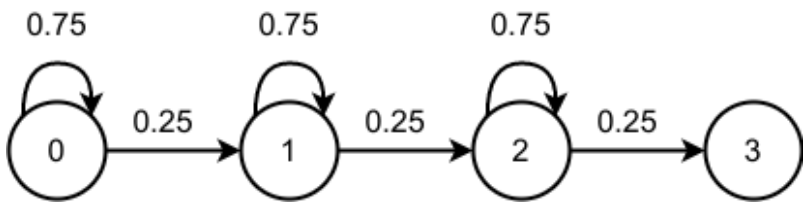
```
build-tree --max-leaves=<max-leaves> \
           <treeacc> <roots-file> <questions-file> \
           <topo-file> <tree-out>
```

使用 Kaldi 进行 HMM 状态的参数绑定

Kaldi 中使用 Transition Model 描述 GMM-HMM 模型结构。

Transition Model 示例

```
<TransitionModel>
<Topology>
<TopologyEntry>
<ForPhones>
11 12 ..... 346
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.75 <Transition> 1 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.75 <Transition> 2 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 </State>
</TopologyEntry>
</Topology>
</TransitionModel>
```



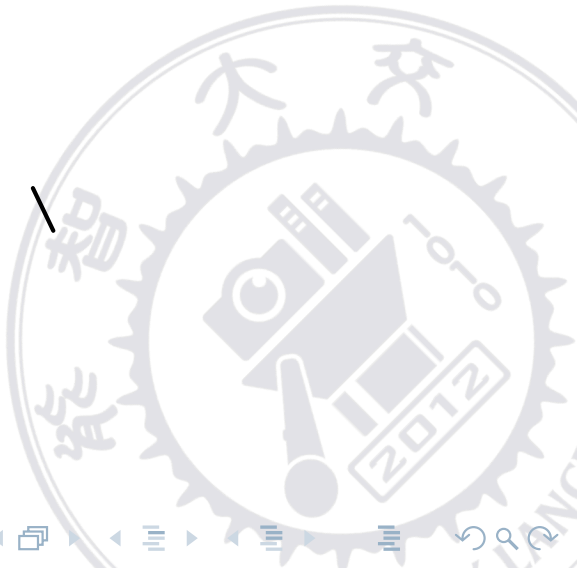
使用 Kaldi 进行 HMM 状态的参数绑定

- Transition Model 以逻辑模型到物理模型的映射的方式定义模型结构。其中 Topology 段描述 GMM-HMM 的逻辑结构，而每个状态中的 PdfClass 则指示该状态所对应的物理模型的编号。
- 因此，尽管逻辑上每个三音子模型拥有独立的 GMM-HMM 结构，但使用相同 PdfClass 的状态将共享相同的分布参数。

在构建决策树后，可使用

```
gmm-init-model <tree-in> <tree-stats-in> \
               <topo-file> <model-out>
```

自动生成含状态绑定的 GMM-HMM 模型。



Two-model Re-estimation

Two-model re-estimation: 用其中一个模型来得到后验概率 (in the E-step), 去更新另一个模型的模型参数 (in the M-step).

- **必要条件:** HMMs 必须拥有相同的状态拓扑
- **应用场景:**
  - GMM $\rightarrow$ Single Gaussian for refined state clustering
  - Diagonal covariance $\rightarrow$ advanced covariance models
- EM 算法修订
  - **E-step:** 使用 HMM θ_A 找到后验概率 (posterior occupancy)

$$\gamma_j(t) = P(q_t = j|\mathbf{O}, \theta_A)$$

  - **M-step:** 针对新模型 HMM θ_B 估计模型参数

$$\hat{\theta}_B = \arg\max_{\theta_B} \sum_j \gamma_j(t) \log p(\mathbf{o}_t|j, \theta_B)$$

Single Pass Retraining

Single Pass Retraining (SPR) 在一个新的声学特征集合上重估声学模型，使得新模型可以使用新特征集合。

- 应用场景:
  - MFCC $\rightarrow$ PLP
  - Wideband $\rightarrow$ narrowband
- EM 算法修订
  - **E-step:** 使用在 $\mathbf{O}^A$ 上训练得到的 HMM，得到后验概率 (posterior occupancy)

$$\gamma_j(t) = p(q_t = j|\theta_A, \mathbf{O}^A)$$

  - **M-step:** 使用新的特征 $\mathbf{O}^B$ 估计 HMM 的模型参数

$$\hat{\theta}_B = \arg\max_{\theta_B} \sum_j \gamma_j(t) \log p(\mathbf{o}_t^B|j, \theta_B)$$

Build State-of-the-art LVCSR System

1. 构建一个 monophone 系统
   - 使用单个高斯分量 flat start (使用初始 pronunciation)
   - 重对齐训练文本 (选择正确的 pronunciation)
2. 构建一个跨词的三音子 triphone 系统
   - 将单音子克隆为三音子
   - 使用未聚类的三音子模型训练 1-2 次迭代
   - 进行决策树状态聚类
   - 进行 Baum-Welch 重估计
   - Iterative mixture splitting
3. 重新构建决策树
   - 使用第 2 步中的最终系统作为对齐模型，对未聚类的单高斯三音子系统进行 2-model re-estimate
   - 进行决策树状态聚类
   - 进行 Baum-Welch 重估计
   - Iterative mixture splitting

令牌传递算法 - Token Passing Algorithm

Viterbi 算法的实际使用

解码 或者搜索过程可以认为是一个传递 (更新)tokens的过程:

- 一个 token是一个信息包，包括
  - 部分路径的对数似然度
  - 令牌的开始时间或帧
  - 词/音素/状态的标识
  - 指向前一个令牌的指针
- 扩展的 HMM 网络的所有状态中，每一个状态拥有一个 token
- 在时刻 t，每个状态中的 token 表示网络中覆盖了从时刻 1 到 t 的输入语音的一条路径
- 初始时, 在所有词的开始状态，产生新的 tokens
- 当 token 进入一个新的词，产生新的 token

Token Passing - Step Model

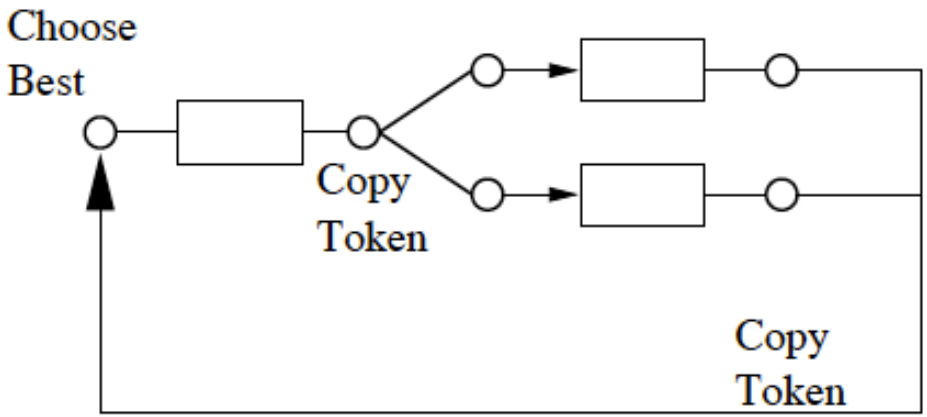
孤立词识别

Start token: $\log P = 0$; **NULL token:** $\log P = -\infty$

```
Put a start token in entry node; Put null tokens in all other nodes;
for each time t = 1 to T do
    /* Start of Step Model */
    for each state i < N do
        Pass a copy of the token Q in state i to all connecting states j;
        Q.LogP = Q.LogP + log(a_ij) + log(b_j(o_t))
    end;
    Discard all original tokens;
    for each state i < N do
        Find token in state i with max LogP and discard the rest
    end;
    for each state i connected to state N do
        Pass a copy of the token Q in state i to state N
        Q.LogP = Q.LogP+ log(a_iN)
    end;
    Find token in state N with max LogP and discard the rest;
    Put null token in entry state
    /* End of Step Model */
end;
```

Token Passing - External Propagation

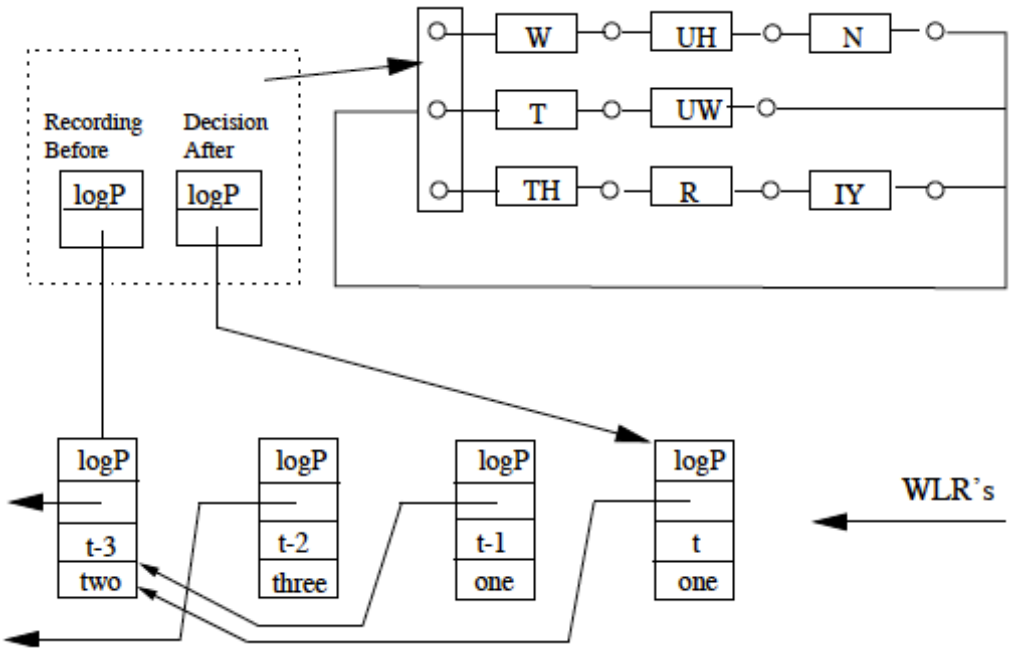
从孤立词到词序列的识别




```
Put a start token in all network entry states;
Put null tokens in all other states;
for each time t = 1 to T do
    Step All Models;
    Propagate Exit Tokens to all connecting entry states;
    Record Decisions;
    Delete all but the best token in each entry state
end
```

Token Passing - Record Decision

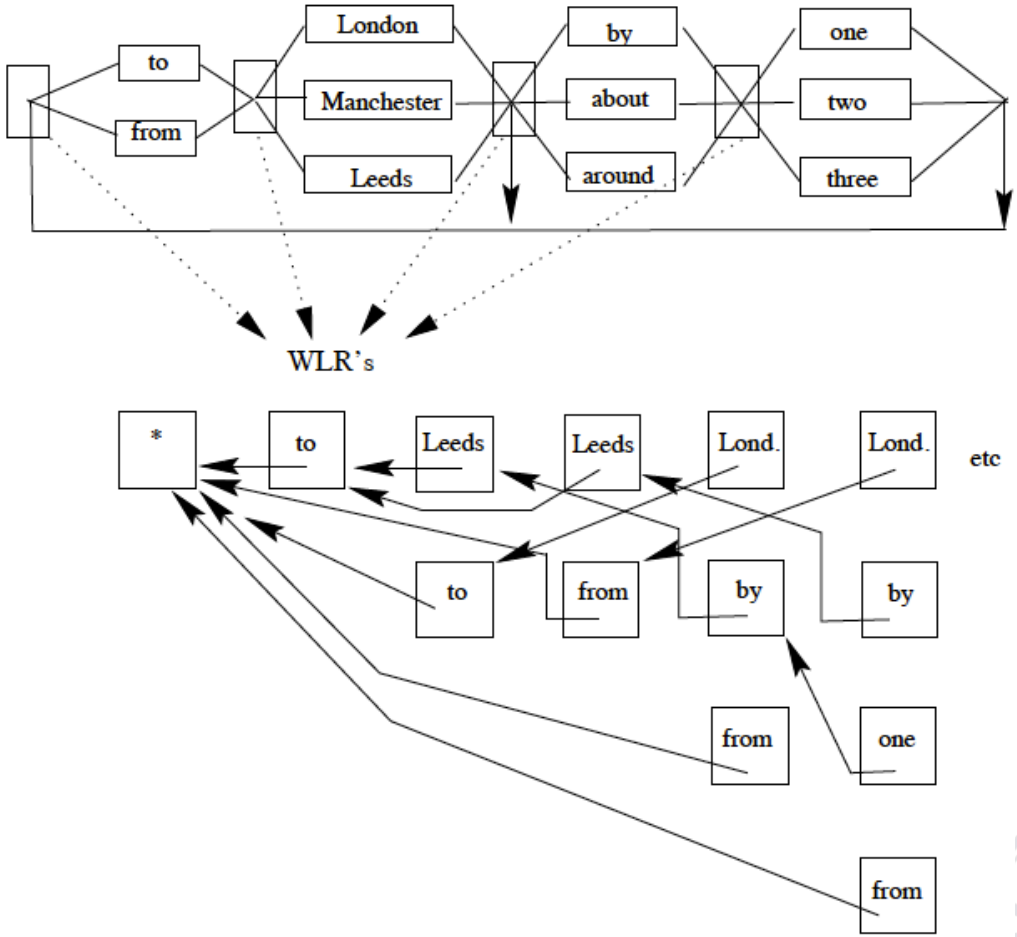
Word link record (WLR)




```
for each best token Q in each entry state at each t do
    Create a new WLR w containing:
            (1) a copy of Q
            (2) time t
            (3) identity of emitting word
    Q.link = w
end;
```

1 WLR is generated per speech frame for each syntactically distinct word

Token Passing - Trace Back




```
Examine WLRs generated at time T, find WLR with max LogP;
Print WLR.time, WLR.LogP, WLR.word;
while WLR.Link != NULL do
      WLR = WLR.Link;
      Print WLR.time, WLR.LogP, WLR.word;
end;
```

搜索复杂度和 Beam Search

概述

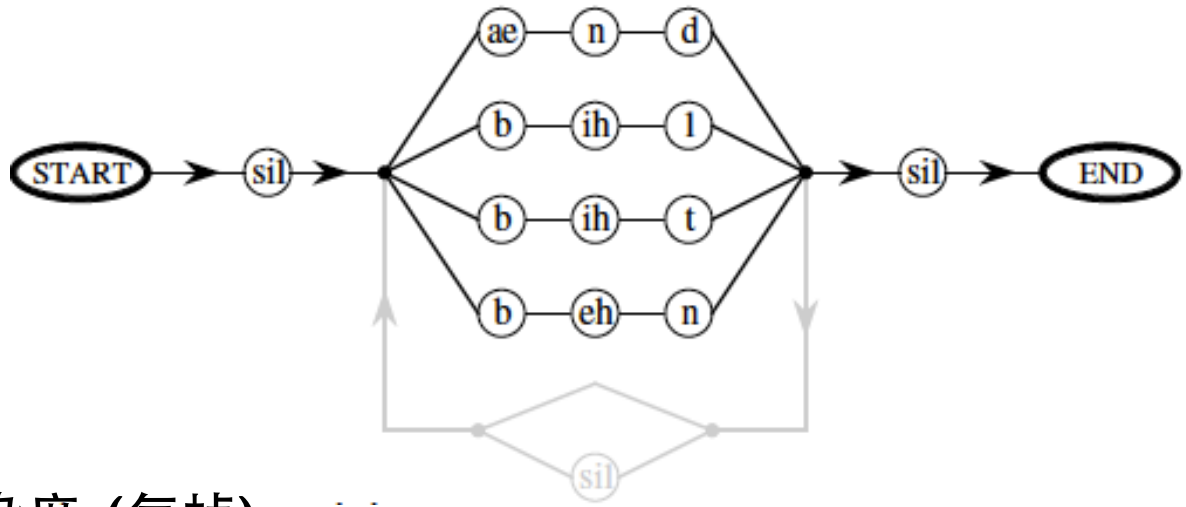
假定有 W 个词, 平均每个词 N 个状态, 每帧需要

1. $W \times N$ 内部令牌的传递
2. W 外部令牌的传递

```
Set all entry models of all network initial words are active;
for each time t = 1 to T do
    for each active model w do
        Step Model;
        Find maximum LogP in w, lMax(w);
    end;
    Find global maximum LogP, gMax
    for each active model w do
        if lmax(w) < gMax - Thresh
           De-Activate w;
    end;
    PropagateExit Tokens to all connecting entry states
             if LogP > gMax - Thresh;
    Record Decisions;
    Delete all but the best token in each entry state;
    Re-Activate all entry models which have just received a new entry token;
end;
```

LVCSR 解码

有限状态网络以及 unigram+monophone/within-word triphone



近似的复杂度 (每帧):

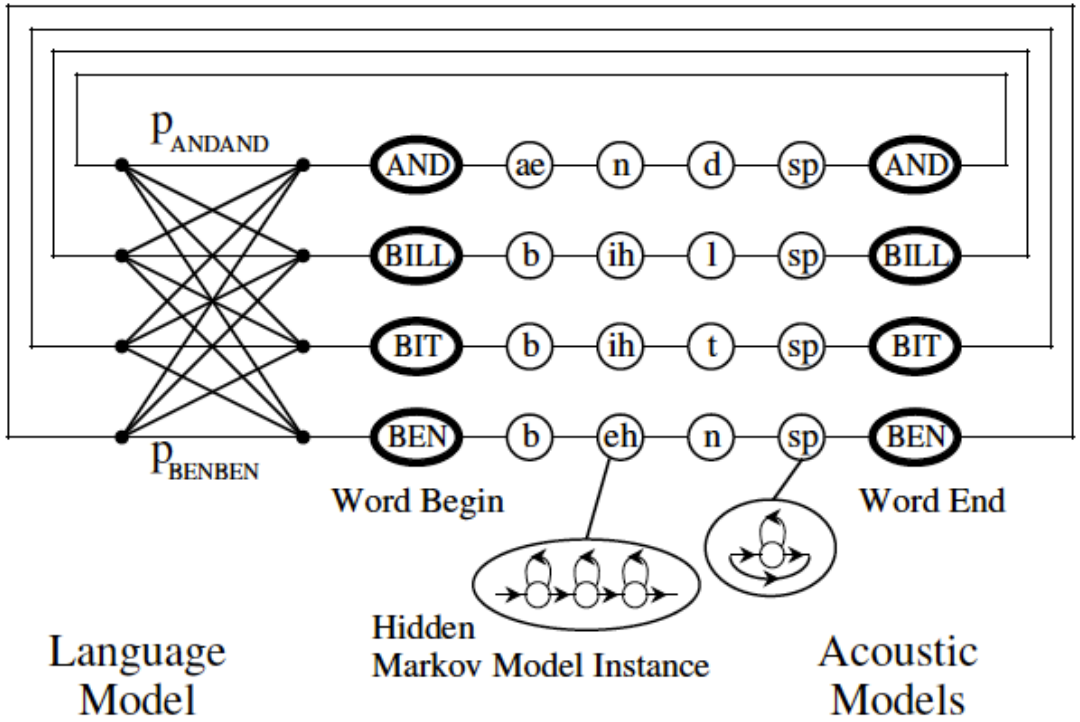
1. 内部传递: $W_{\text{ug}} \times P \times N$, 其中 P 是每个词的平均音素数量, N 是每个模型的状态个数
2. 外部传递: $W_{\text{ug}} \times P + W_{\text{ug}}$

扩展到 unigram 的情况:

- 在每一个词的开始，将 unigram 的概率加到令牌的对数似然度上
- 语言模型的信息尽可能早的合并进来

LVCSR 解码

Bigram+monophone/within-word triphone

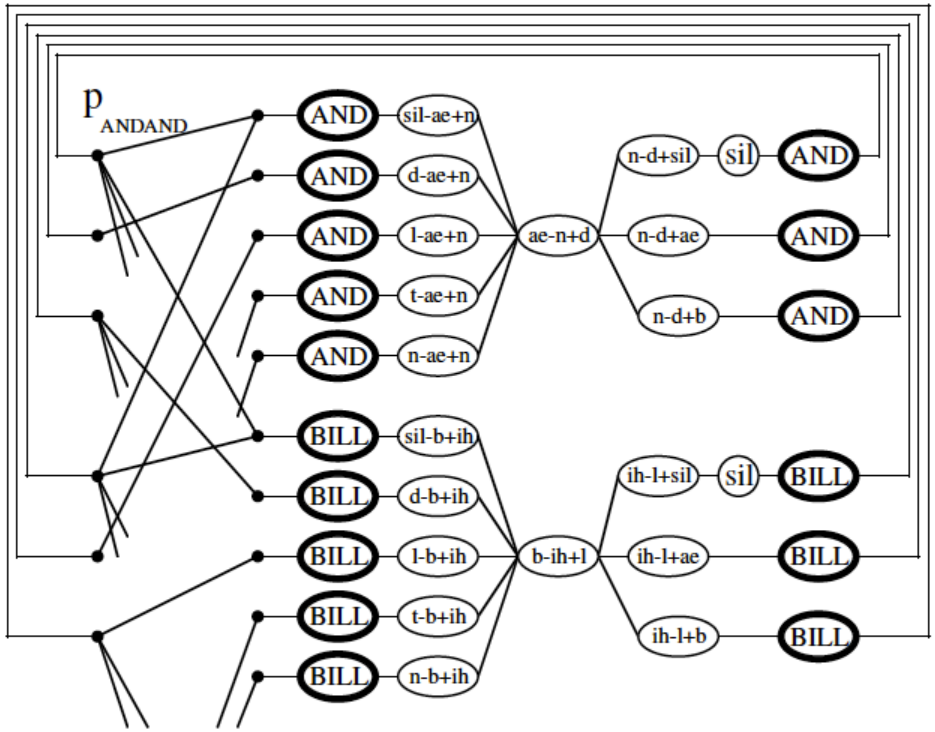


近似的复杂度 (每帧):

1. 内部传递: $W_{\text{ug}} \times (P+1) \times N$
2. 外部传递: $W_{\text{ug}} \times P + W_{\text{ug}}^2$
3. 如果是回退 back-off 语言模型, 词传递大致是 $W_{\text{bg}} + W_{\text{bo}} \times W_{\text{ug}}$

LVCSR 解码

Bigram+cross-word triphone

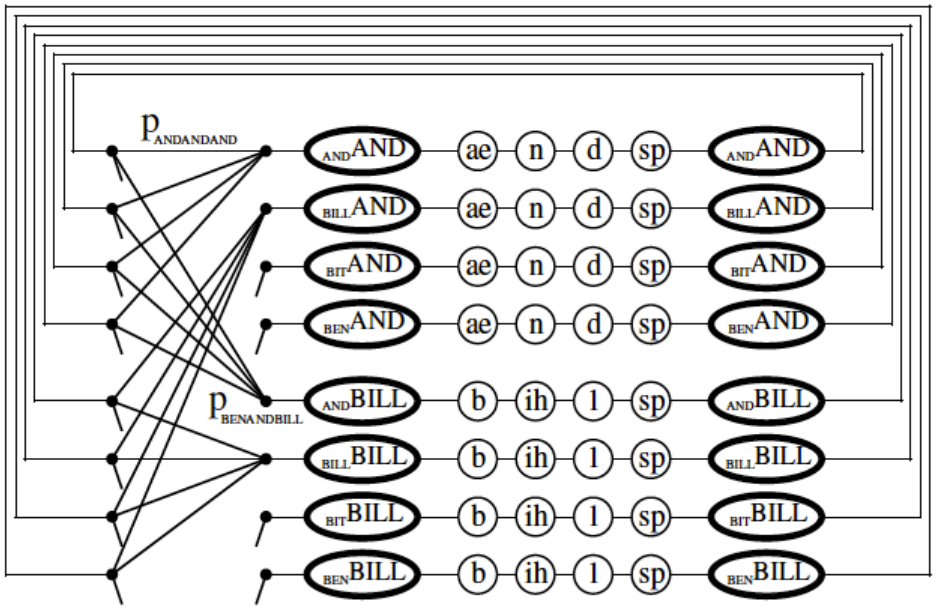


近似的复杂度 (每帧):

1. 内部传递: $W_{\text{ug}} \times (P - 2 + P_{\texttt{prefix}} + P_{\texttt{suffix}}) \times N$
2. 外部传递: $W_{\text{ug}}^2 + W_{\text{ug}} \times (P - 2 + P_{\texttt{prefix}} + P_{\texttt{suffix}})$
3. 如果是回退 back-off 语言模型, 词传递大致是 $W_{\text{bg}} + W_{\text{bo}} \times W_{\text{ug}}$

LVCSR 解码

Trigram+monophone/within-word triphone



近似的复杂度 (每帧):

1. 内部传递: $W_{\text{ug}}^2 \times (P+1) \times N$
2. 外部传递: $W_{\text{ug}}^3 + W_{\text{ug}}^2 \times (P+1)$
3. 如果是回退 back-off LM, 词传递大致是 $W_{\text{tg}} + W_{\text{bo}}^{\text{bg}}(W_{\text{bg}} + W_{\text{bo}} W_{\text{ug}})$
4. 在使用 trigram 语言模型和跨词三音子 triphone 声学模型的情况下，解码复杂度会变得甚至更大!

Beam Search

为了降低解码复杂度和计算量，考虑不去搜索所有可能的路径，而仅搜索更有可能（部分似然度更高）的路径。这部分路径被称为“活跃”(active) 的，每一条活跃路径又称为 beam。

两种方案：

- 阈值限制：剪去似然度过低的路径。阈值可定义为目前所有路径的最高似然度 $\mathcal{L}^{\max}(t)$ 乘一系数 γ。
- 数量限制：仅保留似然度最高的 N 条路径。当 $N = 1$ 时，该方案退化为贪婪搜索。

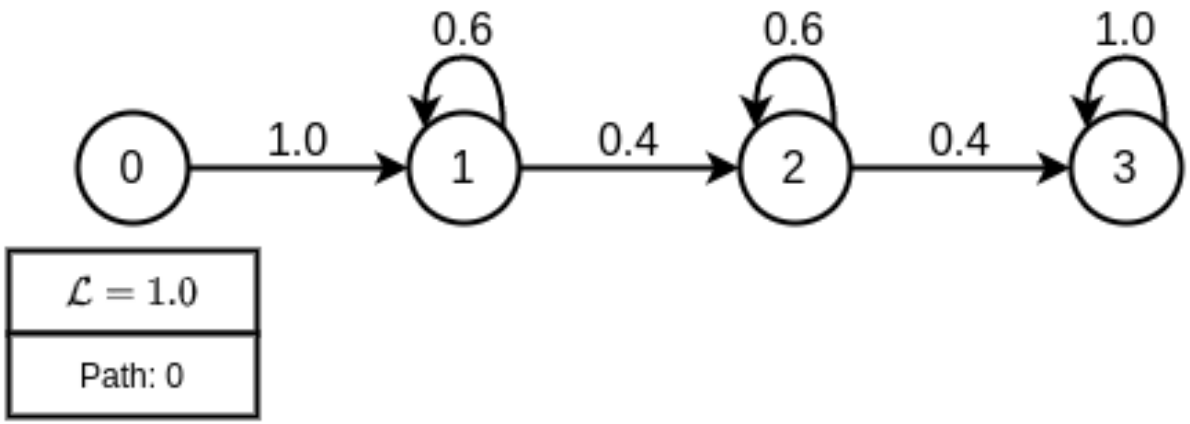
当限制过宽时，计算量不会明显减少；当限制过紧时，算法可能无法找到最优解。因此应选择合适的阈值或 beam 数量进行搜索。

Beam Search 不仅适用于 Token Passing 解码，而且广泛用于各类搜索问题的加速。

Beam Search

示例

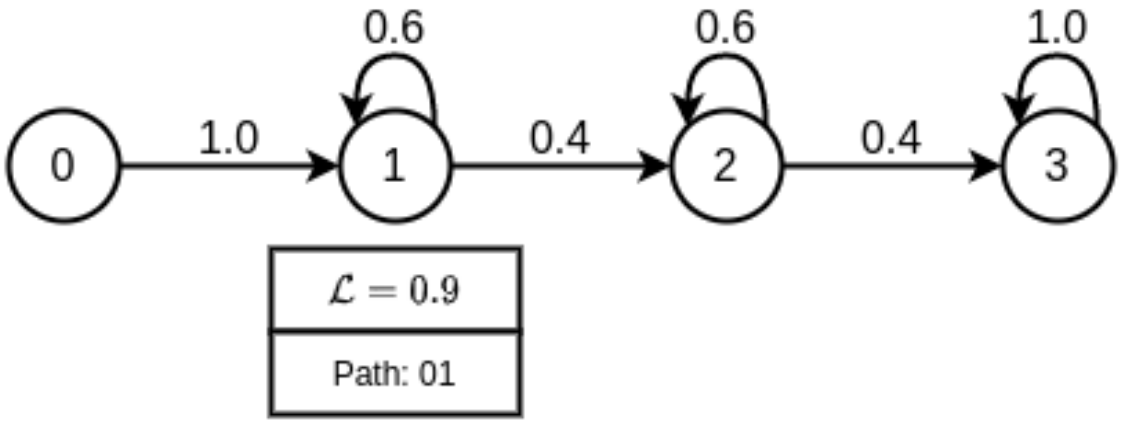
0) 以三状态 HMM 为例：设输入特征序列为 112233，HMM 的每一状态 i 输出特征 i 的概率为 0.9，输出其余特征的概率为 0.1。Beam Search 策略为仅保留似然度最高的 2 条路径。



Beam Search

示例

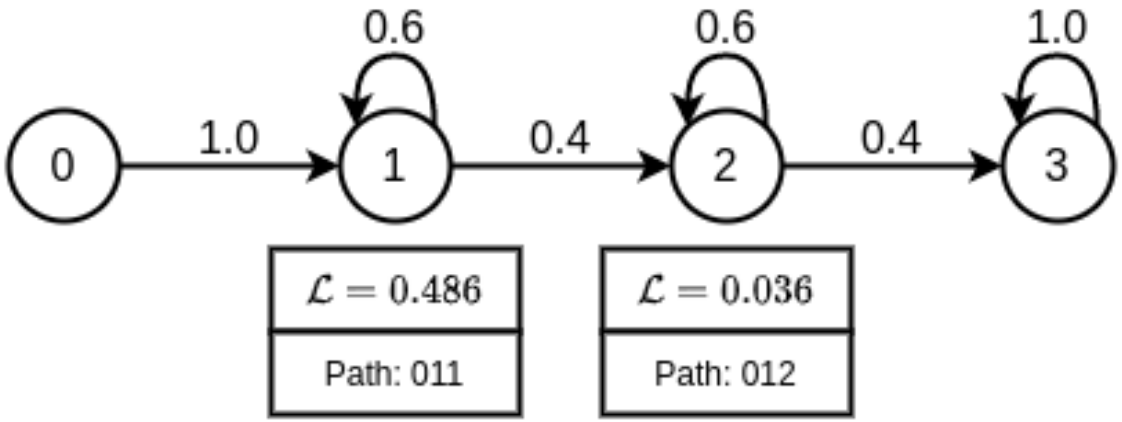
1) 以三状态 HMM 为例：设输入特征序列为 112233，HMM 的每一状态 i 输出特征 i 的概率为 0.9，输出其余特征的概率为 0.1。Beam Search 策略为仅保留似然度最高的 2 条路径。



Beam Search

示例

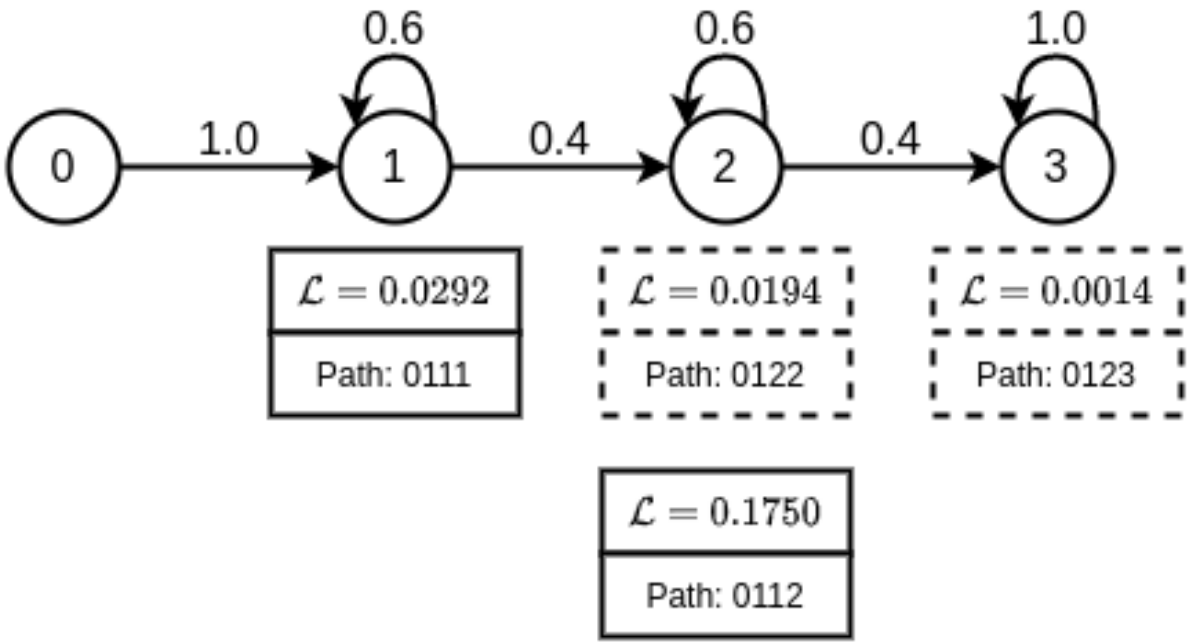
2) 以三状态 HMM 为例：设输入特征序列为 112233，HMM 的每一状态 i 输出特征 i 的概率为 0.9，输出其余特征的概率为 0.1。Beam Search 策略为仅保留似然度最高的 2 条路径。



Beam Search

示例

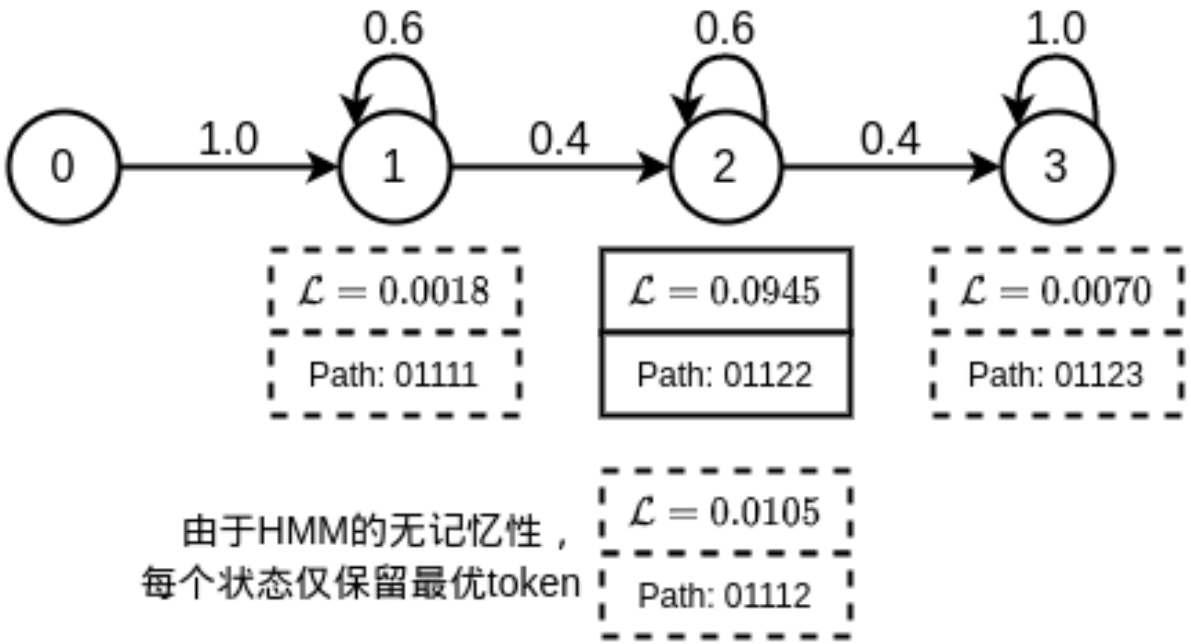
3) 以三状态 HMM 为例：设输入特征序列为 112233，HMM 的每一状态 i 输出特征 i 的概率为 0.9，输出其余特征的概率为 0.1。Beam Search 策略为仅保留似然度最高的 2 条路径。



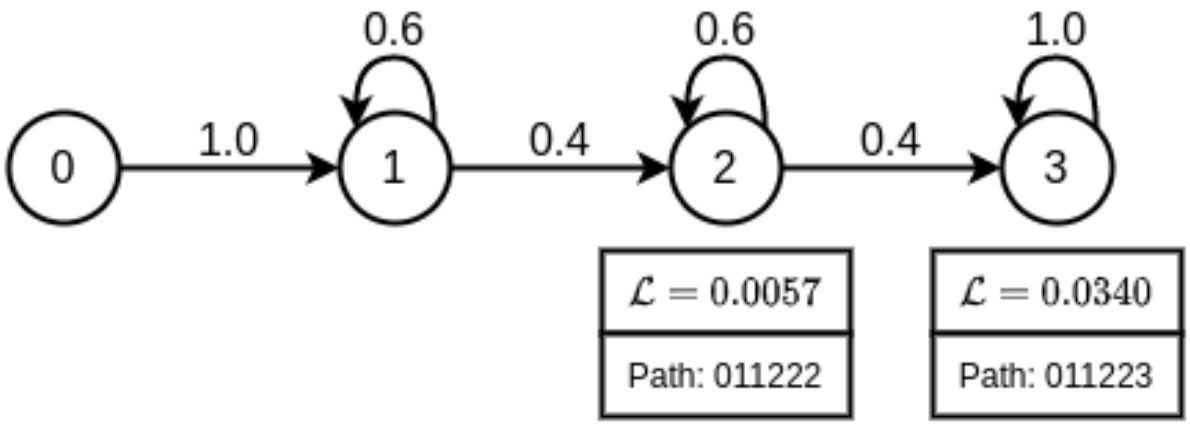
Beam Search

示例

4) 以三状态 HMM 为例：设输入特征序列为 112233，HMM 的每一状态 i 输出特征 i 的概率为 0.9，输出其余特征的概率为 0.1。Beam Search 策略为仅保留似然度最高的 2 条路径。



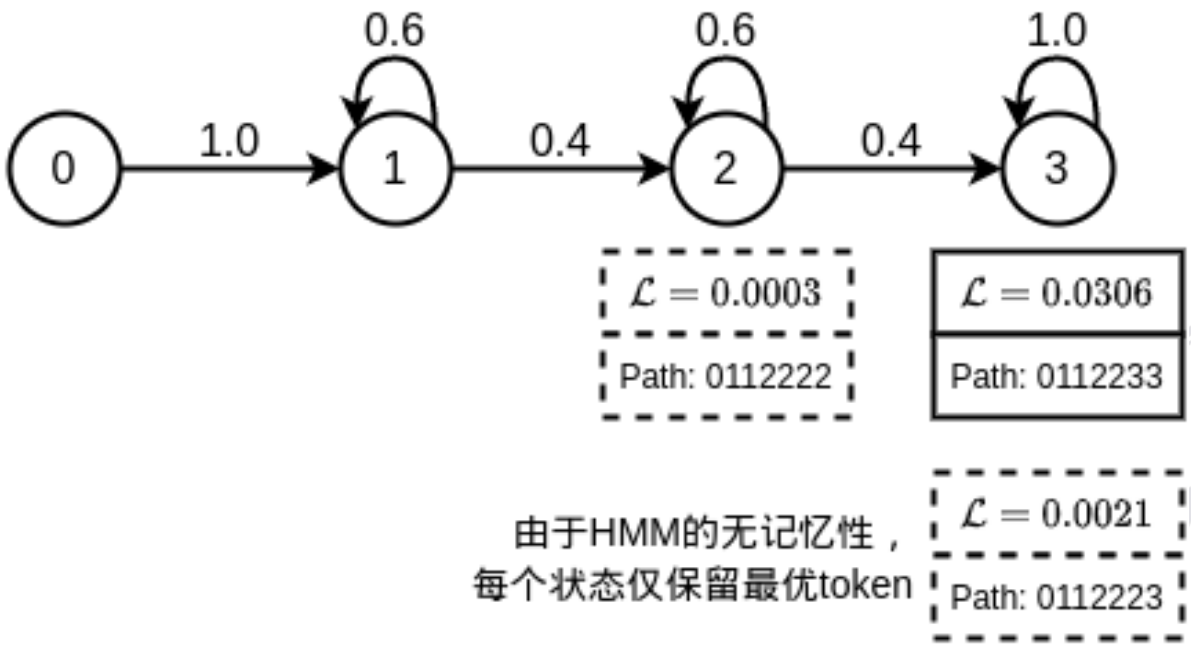
5) 以三状态 HMM 为例：设输入特征序列为 112233，HMM 的每一状态 i 输出特征 i 的概率为 0.9，输出其余特征的概率为 0.1。Beam Search 策略为仅保留似然度最高的 2 条路径。



Beam Search

示例

6) 以三状态 HMM 为例：设输入特征序列为 112233，HMM 的每一状态 i 输出特征 i 的概率为 0.9，输出其余特征的概率为 0.1。Beam Search 策略为仅保留似然度最高的 2 条路径。



剪枝搜索中的不对称性

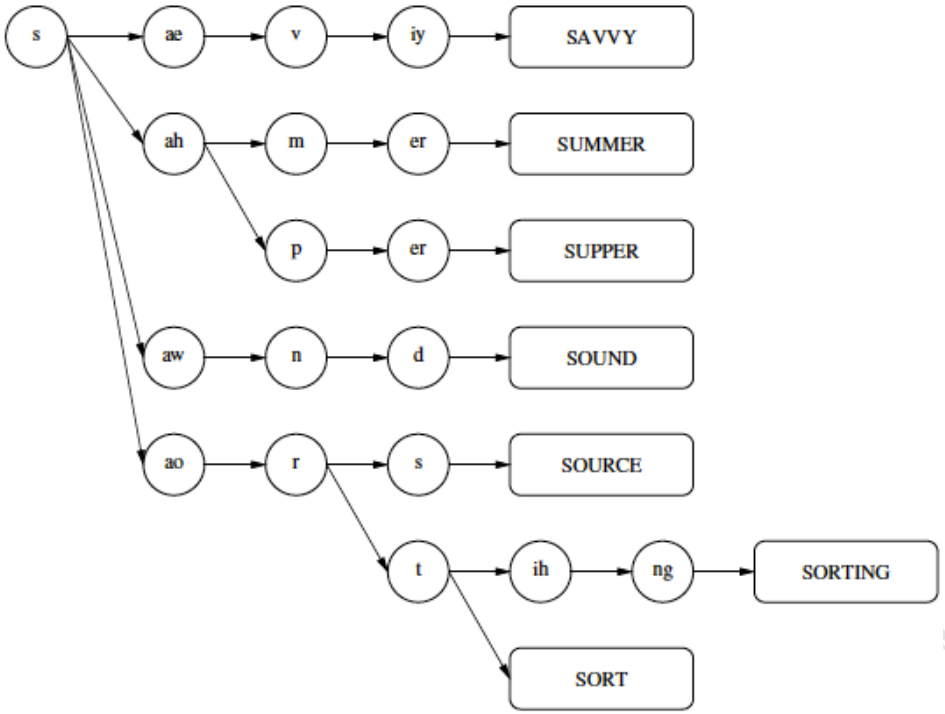
在一个 5000 词的 WSJ 任务上，在不同的词的位置上，平均“活跃”音素的数量是 (word-internal triphones + bigram):

Model position in word	First	Second	Last but one	Last	Word End
Number active	3539	866	265	91	43
Proportion active	65.4%	16.0%	4.9%	1.7%	0.8%
Relative computation	76.0%	18.6%	5.7%	1.9%	

- 对于全连接的 N-gram 上的解码器，大部分搜索工作都集中在词的开始
- 在词的开始部位，有效地减少计算量非常重要
  - 树状结构的解码网络

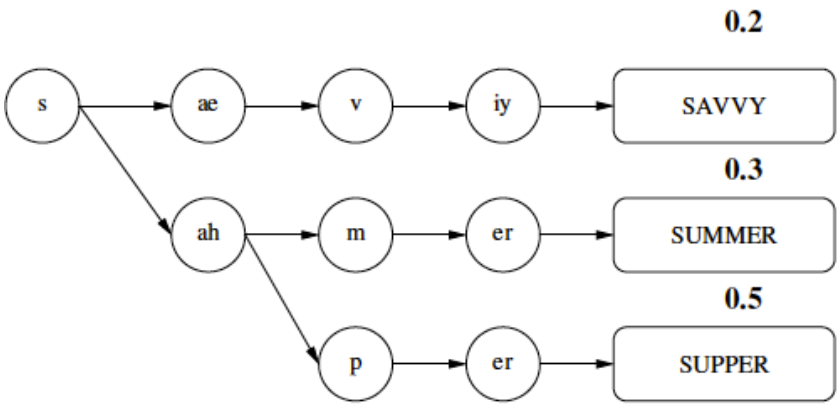
树状结构的字典

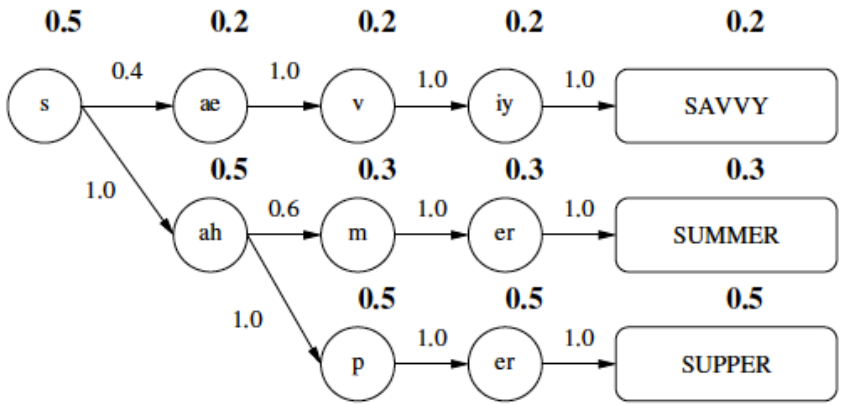
SAVVY	s ae v iy
SUMMER	s ah m er
SUPPER	s ah p er
SOUND	s aw n d
SOURCE	s ao r s
SORT	s ao r t
SORTING	s ao r t ih ng



- 终止节点表示词
- 树状结构表示共同的字典前缀
- 适合在词的开始部分减少 cost

语言模型分数因子





- 字典树的结构使得直到词的结束才能使用 LM 分数，推迟了 LM 分数的使用
- 解决方案: 使用 LM 分数尽可能的早
  - 从终止节点上的词语言模型分数开始
  - 将最好的 LM 分数后向传播到父节点
  - 对连接父节点和孩子节点的每一条弧都赋于一个尺度因子 (scaling factor)(best node has factor 1.0)
  - 重复以上过程直到到达根节点

强制对齐 - Forced Alignment

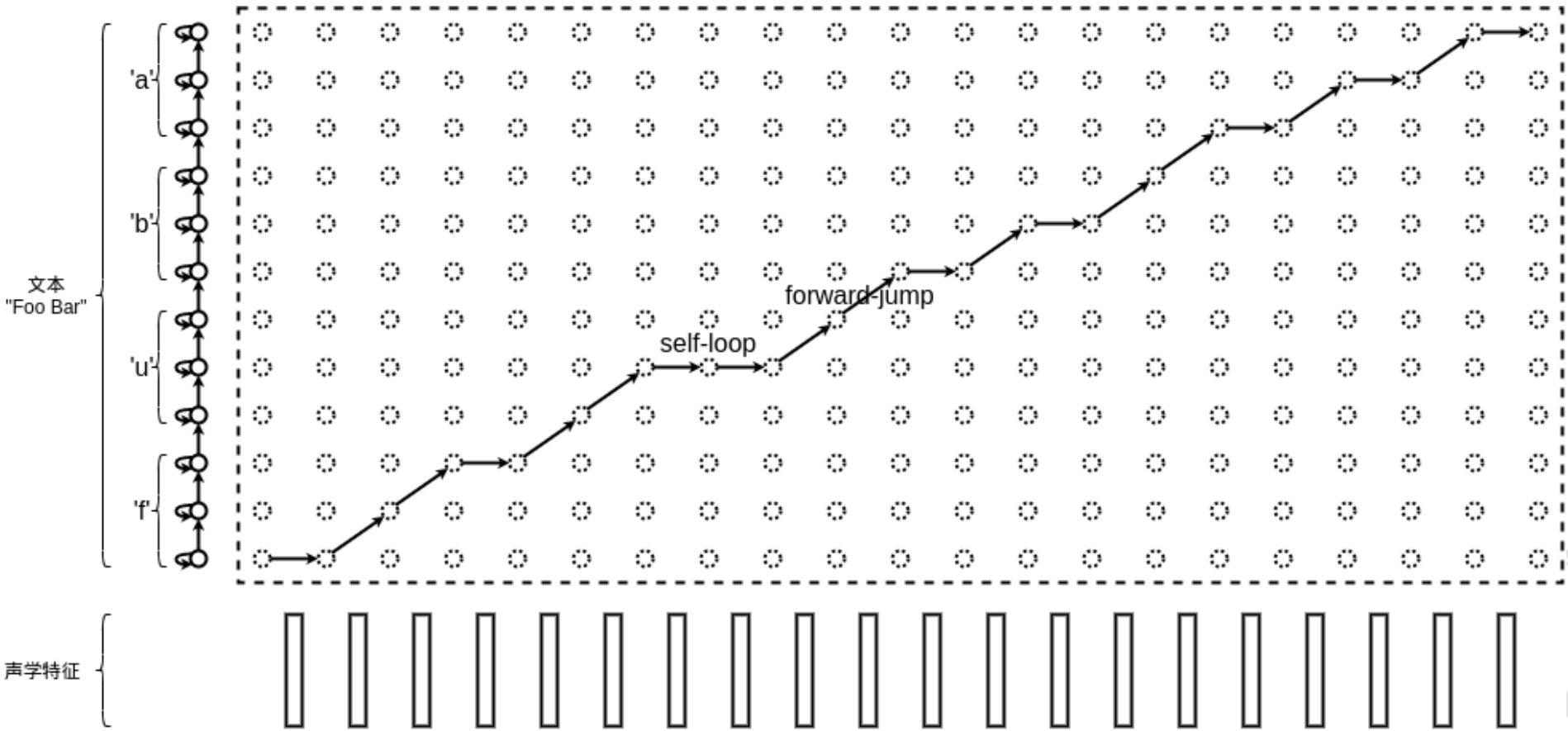
- 当不使用 Baum-Welch 算法优化 GMM-HMM 模型时，则需要预先获得每一帧对应的 HMM 状态以训练 GMM 参数。这被称为 **Viterbi training**。Viterbi training 的优势在于模型训练速度明显加快，而模型性能没有明显损失。
- 但是通常数据没有状态级标注。为了获得状态级标注，需要使用一个预先训练好的声学模型对数据进行**强制对齐**。

强制对齐 - Forced Alignment

- **强制对齐**是一种特殊的解码过程：
  - 使用输入语音对应的标注文本直接构建解码图，图中仅包含标注文本对应的状态。每个状态上有指向自身以及下个状态的跳转。
  - 根据语音帧和已有的声学模型，选取解码图中的一条最优路径将各帧匹配至解码图上，从而获得各帧的状态标注。
- 训练首个声学模型时，由于没有可用于对齐的声学模型，`Kaldi` 的实现中将语音按文本对应的状态数平均分段，作为首次对齐。

强制对齐 - Forced Alignment

示例



上图对文本“Foo Bar”展示了一种可能的最优对齐路径。图中横轴为时间，纵轴为状态。

多候选

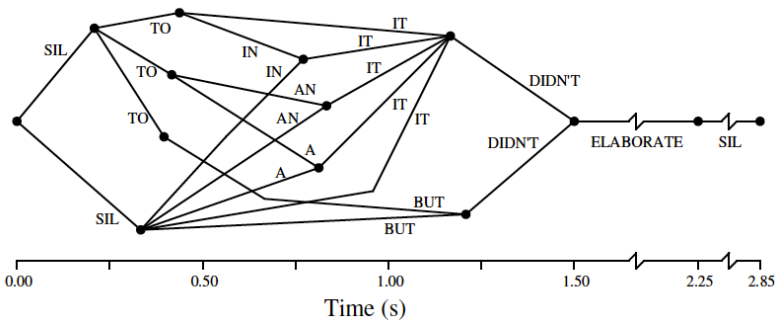
N-Best 或者 Lattices

多候选 是解码器常常需要的一种输出形式:

- 作为初始的输出，用于 multipass 系统 (复杂的模型重打分 (rescoring))
- 作为混淆网络，用于置信度计算
- 用于鉴别性训练-Discriminative training

两种形式的表示

- **N 候选 (N-Best):** 每一候选是一个句子 (词序列)
- **词图 (Word Lattices):** 是一种网络结构，其中包括
  - 节点集合：对应时间信息 (或者词的结束)
  - 弧集合：弧上标有词的标识，以及在节点之间对应某个词的声学模型或者语言模型分数



在解码中生成多候选

Word-pair assumption: 对任意的 word pair，boundary time 仅依赖于这两个词，而不依赖于任何历史。

在解码中步骤的改变:

1. 每一个状态保留有 n 个 tokens
2. 来自每一个相连状态的所有 n 个 tokens 都通过正常的令牌传递准则进行传播
3. 与之前仅保留每一个状态的最优 token，丢弃所有其他 tokens 不同，这里需要进行如下操作
   - 3.1 找到所有那些有相同历史词的 tokens 集合
   - 3.2 对所有的这些状态，通过相加的方法合并 tokens，或者找到部分路径概率中最大的 token
   - 3.3 保留每一个状态的分数最高的 n 个 tokens，同时丢弃其他的 tokens

混淆网络

纯粹的词图

- 移除时间信息，并且合并 lattices 中邻近的词
- 表示了比 lattice 更多的可能