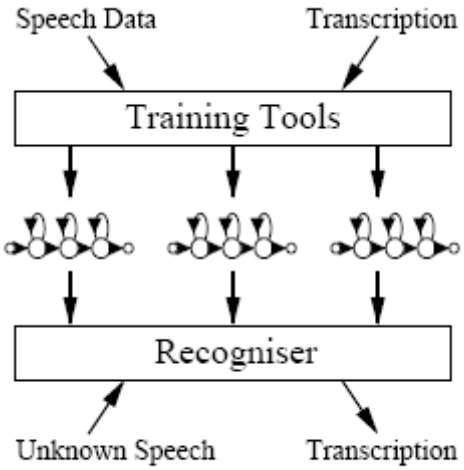
HTK 中文手册

纠错邮箱：**jianglonghu@163.com**

内部资料，请务外传

第一章 HTK 基础



HTK是建立隐马尔可夫模型（HMM）的工具包，HMM能用于模拟任何时间序列，而HTK内核对类似过程是通用的。不过HTK主要用于设计构造基于HMM的语音处理工具，特别是识别器。因此，HTK中的一些基础组件专门用于这一任务。如上图所示，它主要有两个处理阶段。首先，HTK训练工具使用训练语料和相应的标注文件来估计HMM模型集的参数；第二阶段，使用HTK识别工具来识别未知语音。

这本书主体的大部分内容都和这两个处理过程的机制相关。然而，在开始更细致的介绍之前我们需要了解HMM的基本原理，这将有利于我们对HTK工具有个整体把握，对HTK如何组织训练和识别过程也有一定的认识。

本书第一部分提供简要介绍了HMM的基本原理，作为HTK的使用指南。这一章介绍了HMM的基本思想和在语音中的应用。后面一章简要介绍了HTK，而且对老版本的使用者还指出了2.0版及后续版本的主要不同之处。在本书的指南部分的最后一章，第三章，描述了一个简单的小词汇连续语音识别系统，以此为例介绍如何使用HTK构造一个基于HMM的语音识别系统。

这本书的第二部分对第一部分进行了详细的讲解。这部分可以结合第三部分和最后一个部分（HTK的参考手册）来阅读。这个部分包括：每个工具的描述、配置HTK的各个参数和产生错误时的错误信息列表。

最后需要指出的是这本书仅仅把HTK当成一个工具包，并没有提供使用HTK库作为编程环境的相关信息。

1.1 HMM 的一般原理

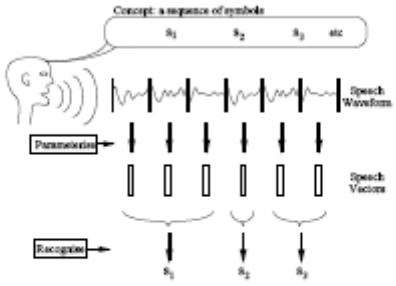


Fig. 1.1 Message Encoding/Decoding

语音识别系统通常假设语音信号是编码成一个或多个符号序列的信息实体（如图1.1）。为了实现反向操作，即识别出给定说话人的语音的符号序列，首先将连续语音波形转换成一个等长的离散参数向量序列。假设这个参数向量序列是语音波形的一个精确表示，在一个向量对应的时间段内（代表性的有10ms等等），语音信号可看成是平稳的。虽然这一假设并不严格，但是这是合理的近似。典型的参数表示法常用的是平滑谱或线性预测系数以及由此衍生的各种其它的表示法。

识别器的任务是在语音向量序列和隐藏的符号序列间实现一个映射。有两个问题使得完成这一任务非常困难，第一，因为不同的隐藏符号能有相似的发音，所以符号到语音的映射不是一一对应的，而且，发音人不同的心情和环境等因素会导致语音波形产生非常多的变化。第二，从语音波形中不能准确地识别出各符号间的边界，因此，不能将语音波形当做一个静态样本连接的序列。

限制识别任务为孤立词识别就可以避免第二个问题中不知道单词边界位置的问题。如图1.2所示，这里的各段语音波形对应了固定词典中的一个简单符号（比如一个单词）。尽管我们对这一问题的简化有点理想化，然而它却有广泛的实际应用。此外，在处理更为复杂的连续语音之前介绍上述方法，为掌握基于HMM识别模型的基本思想打下了很好的基础。因此，我们首先将介绍使用HMMs的孤立词识别模型。

1.2 孤立词识别模型

令每个发音单词用语音向量序列或观察向量 O 表示，定义为：

$$O = o_1, o_2, ..., o_T \tag{1.1}$$

其中 o_t 表示在 t 时刻观察到的语音向量。就可以认为孤立词识别问题是在计算：

$$\arg\max_i \{p(w_i \mid o)\} \tag{1.2}$$

其中 w_i 表示第 i 个词典词。这个概率不是直接计算的，而是由贝叶斯公式给出：

$$p(w_i \mid o) = \frac{p(o \mid w_i)p(w_i)}{p(o)} \tag{1.3}$$

因此，给定先验概率 $p(w_i)$，最可能的发音单词就仅仅取决于概率 $p(o \mid w_i)$。给定观察序列 O 的维数，从发音单词的样本直接计算联合条件概率 $p(o_1, o_2, ..., o_T \mid w_i)$ 是很难实现的。然

而，如果一个单词的参数模型假设是马尔可夫模型，当估计条件观察值密度 $p(o|w_i)$ 的问题被估计马尔可夫参数的简单问题代替，由观察向量计算 $p(o_1,o_2,...,o_T|w_i)$ 就可以实现了。

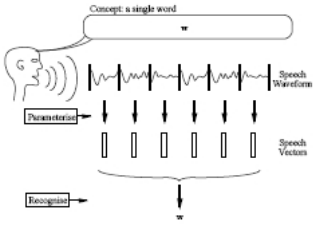


Fig. 1.2 Isolated Word Problem

在基于HMM的语音识别中，假设观察到的语音向量序列对应由马尔可夫模型产生的单词，如图1.3所示。马尔可夫模型是一个有限状态机，它每隔一定时间改变一次状态，在 t 时刻进入状态 j 输出语音向量 o_t 的概率密度为 $b_j(o_t)$，此外，从状态 i 到状态 j 的转移概率为 a_{ij}。图1.3所示的是这一过程的一个例子，其中六个状态模型按状态序列X=1，2，2，3，4，4，5，6 依次转移，产生了从 o_1 到 o_6 的输出序列。需要注意的是，在HTK中一个HMM的入口状态和出口状态是non-emitting的，在后文中我们将对整个模型的构建作更为详细地说明。

模型 M 通过状态序列 X 产生观察序列 O 的联合概率由转移概率和输出概率决定。对图1.3的状态序列 X 有：

$$P(O,X|M)=a_{12}b_2(o_1)a_{22}b_2(o_2)a_{23}b_3(o_3)... \tag{1.4}$$

然而，实际情况下仅仅只知道观察序列 O，状态序列 X 是被隐藏的，这就是为什么称该模型为隐马尔可夫模型了。

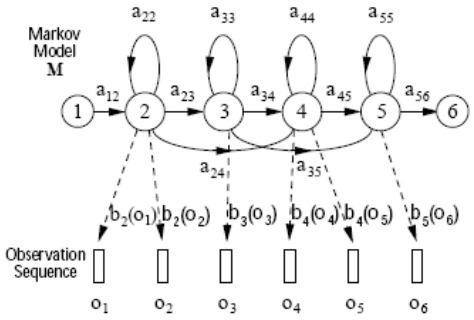


Fig. 1.3 The Markov Generation Model

由于 X 是未知的，我们就要把所有可能的状态序列 $X=x(1),x(2),x(3),...,x(T)$ 考虑进去，则：

$$P(O,X|M)=\sum_X a_{x(0)x(1)}\prod_{t=1}^{T}b_{x(t)}(o_t)a_{x(t)x(t+1)} \tag{1.5}$$

其中 $x(0)$ 表示模型的初态，$x(T+1)$ 表示模型的终态。

对等式(1.5)进行改进，仅仅考虑最有可能的状态序列，则有：

$$\hat{P}(O \mid M) = \max_{X}\{a_{x(0)x(1)}\prod_{t=1}^{T} b_{x(t)}(o_t)a_{x(t)x(t+1)} \qquad (1.6)$$

虽然不易直接计算(1.5)式和(1.6)式，但是使用简单的递推公式可以对它们进行有效的计算。在更深地讨论之前，注意到如果(1.2)式能够求出，那么识别问题也就被解决了。假设一个模型 M_i 对应一个单词 w_i，等式1.2就可以用1.3式求出，另外假设：

$$P(O \mid w_i) = P(O \mid M_i) \qquad (1.7)$$

当然，所有的这些都要假设每个模型 M_i 的参数 $\{a_{ij}\}$ 和 $\{b_{ij}\}$ 是已知的。这里依赖于HMM框架的魅力和能力。假定一组训练样本对应一个特定模型，根据一个稳定有效的重估过程可自动求出该模型的参数。因此，当每个单词都有足够多的具有代表性的样本时，一个HMM就可以构造出来了，其中隐含了对真实语音的所有的内在变化的模拟。图1.4描述了HMM在孤立词识别中的应用。首先，在词典中只有“one”，“two”，“three”三个单词的情况下，用各个词典词的许多样本训练出对应的HMM。然后，为了识别未知单词，计算各个模型生成该单词的似然，找出最有可能的一个模型，这就识别出了这一未知的单词。

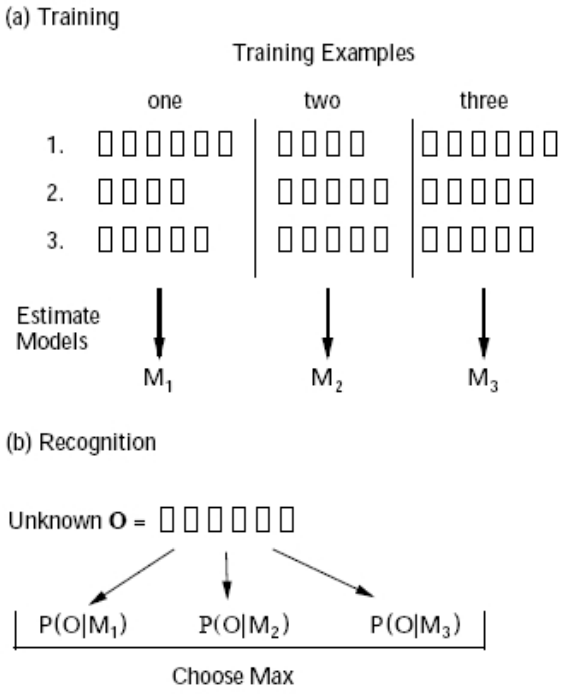


Fig. 1.4 Using HMMs for Isolated Word Recognition

1.3 输出概率

在详细讨论参数估计问题之前，先将输出概率分布 $\{b_j(o_t)\}$ 的形式定义清楚。设计HTK主要就是为了处理连续参数模型，它使用的是连续密度多元输出分布。当然，它也可以处理输出分布为离散概率的离散观察序列。为简单起见，本章仅考虑连续密度分布。在第七章将讲述使用离散概率建模与它的一些细微区别，而在十一章将作更为详细地讨论。

和大多数连续密度HMM系统一样，HTK也使用混合高斯密度来描述输出分布，但它支持的范围更为广泛。HTK允许每个在 t 时刻的观察向量分成 S 独立的数据流 o_{st}，这样计算

$b_j(o_t)$ 的公式就是：

$$b_j(o_t) = \prod_{s=1}^{S} \left[\sum_{m=1}^{M_s} c_{jsm} N(o_{st}; u_{jsm}; \sum_{jsm}) \right]^{r_s} \tag{1.8}$$

其中， M_s 是 s 流中混合高斯成分的数量， c_{jsm} 表示第 m 个成分的权值， $N(\cdot; u, \Sigma)$ 表示多维高斯，均值为 u ，协方差为 Σ 的高斯分布。则有：

$$N(o; u, \sum) = \frac{1}{\sqrt{(2\pi)^n |\sum|}} e^{-\frac{1}{2}(o-u)' \sum^{-1} (o-u)} \tag{1.9}$$

其中 n 为 o 的维数。

指数 r_s 是 s 流的权值[1]，它用于给某些特定的流更高的权值，且只能手工设置。现在的HTK工具还不能估计出它的值。

使用多个数据流可以分别模拟多种信息源，在HTK中流的处理是十分常见的。但是它的语音输入模块假设数据源最多可以分成四个数据流。在第五章将更详细地讨论这些，到目前为止，知道缺省的数据流划分就足够了，缺省数据流包括基本参数向量、一阶导、二阶导和log域的能量。

1.4 Baum-Welch 重估

要求出一个 HMM 的参数，首先需要大致猜测它们可能是什么。之后，使用所谓的 *Baum-Welch* 重估公式就可以使用最大似然判决准则(ML)找出更准确的参数。

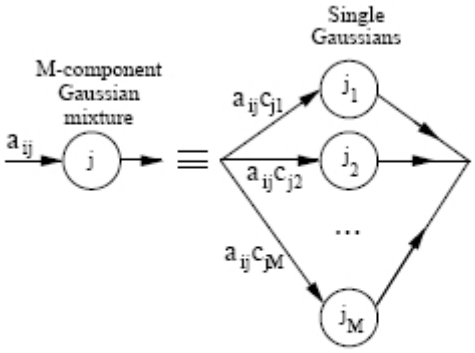


Fig. 1.5 Representing a Mixture

在第八章将详细介绍用在HTK中的公式，在这里只是简单地介绍一些基本公式。首先，需要指出的是，因为可以认为每个流在统计意义上独立，所以包含多个数据流并不会显著地改变问题实质。另外，混合高斯成分可以看成是子状态（sub-state）的一种特殊形式，其转移概率就是该成分的权值（如图1.5）。

因此，问题的实质就是估计HMM的均值和方差，其中每个状态输出分布是一个单高斯，有：

$$b_j(o_t) = \frac{1}{\sqrt{(2\pi)^n |\sum_j|}} e^{-\frac{1}{2}(o_t-u_j)' \sum_j^{-1} (o_t-u_j)} \tag{1.10}$$

如果在这个HMM中只有一个状态 j ，那么参数估计将会很容易。 u_j 和 $\sum_j$ 的最大似然估计

[1] 通常当作一个码本指数来引用。

就是简单的求平均，即：

$$\hat{u}_j = \frac{1}{T}\sum_{t=1}^{T} o_t \tag{1.11}$$

和

$$\hat{\sum}_j = \frac{1}{T}\sum_{t=1}^{T}(o_t - \hat{u}_j)(o_t - u_j)' \tag{1.12}$$

当然在实际中，通常都有很多状态并且因为隐含的状态序列是未知的，所以观察向量和每个状态并不是一一对齐的。注意，如果能够做到向量和状态的近似对应，那么就可以使用公式(1.11)和(1.12)给出参数的初始值。事实上，HTK中的*HInit*工具就是这么实现的。*HInit*首先在模型状态中均匀地划分训练观察向量，然后用公式(1.11)和(1.12)给出每个状态的均值和方差的初始值，再用下面将要提到的Viterbi算法找出最大似然状态序列，重新给状态分配观察向量，最后再用公式(1.11)和(1.12)得到新的更好的初始值。重复上述过程，直到估计值不再改变为止。

因为每个观察序列的全部似然是基于所有可能的状态序列的总和的，所以每个观察向量 o_t 都会影响计算每个状态 j 的最大似然值。换句话说，每个观察向量根据模型的概率按比例分配给每一个状态，而不是在上述的近似中将每个观察向量分配给某个特定状态。因此，如果 $L_j(t)$ 表示在时刻 t 状态 j 的概率，则上述的公式(1.11)和(1.12)变成下面的加权平均：

$$\hat{u}_j = \frac{\sum_{t=1}^{T} L_j(t) o_t}{\sum_{t=1}^{T} L_j(t)} \tag{1.13}$$

和

$$\hat{\sum}_j = \frac{\sum_{t=1}^{T} L_j(t)(o_t - u_j)(o_t - u_j)'}{\sum_{t=1}^{T} L_j(t)} \tag{1.14}$$

其中分母的和包括了所需的规整。

公式(1.13)和(1.14)是计算HMM均值和方差的Baum-Welch重估公式。我们可以推导出相似但稍微复杂一点的转移概率计算公式（见第八章）。

当然，在使用公式(1.13)和(1.14)之前，状态占有率必须先求出来，这个可以用前向-后向算法来计算。对某个有 N 个状态的模型 M ，定义前向概率[2] $a_j(t)$ 为：

$$a_j(t) = P(o_1, ..., o_t, x(t) = j \mid M) \tag{1.15}$$

其中 $a_j(t)$ 是在 t 时刻进入状态 j 观察到的前 t 个语音向量的联合概率。这一前向概率可以使用下面的递归公式计算：

$$a_j(t) = \left[\sum_{i=2}^{N-1} a_i(t-1) a_{ij}\right] b_j(o_t) \tag{1.16}$$

[2] 因为输出分布是密度，所以不是真正意义上的概率，但这是比较方便的假定。

该式取决于在 t 时刻状态为 j 的概率，以及对所有的被转移概率 a_{ij} 加权过的前一时刻状态 i 累加前向概率，由此可推导出观察向量 o_t 。状态1和状态 N non-emitting[3]使上式中的累加范围有了一点独特的限制。上述的初始条件为：

$$a_1(1) = 1 \tag{1.17}$$

$$a_j(1) = a_{1j} b_j(o_1) \tag{1.18}$$

其中 $1 < j < N$，终止条件为：

$$a_N(T) = \sum_{i=2}^{N-1} a_i(T) a_{iN} \tag{1.19}$$

注意到，由 $a_j(t)$ 的定义可以得出，

$$P(O \mid M) = a_N(T) \tag{1.20}$$

因此，前向概率的计算也能求出总概率 $P(O \mid M)$ 。

后向概率定义为 $\beta_j(t)$ ：

$$\beta_j(t) = P(o_{t+1}, ..., o_T \mid x(t) = j, M) \tag{1.21}$$

类似于前向概率，后向概率也可以用下面的递推公式计算出来，

$$\beta_i(t) = \sum_{j=2}^{N-1} a_{ij} b_j(o_{t+1}) \beta_j(t+1) \tag{1.22}$$

初始条件为：

$$\beta_i(T) = a_{iN} \tag{1.23}$$

其中 $1 < i < N$，终止条件为：

$$\beta_1(1) = \sum_{j=2}^{N-1} a_{1j} b_j(o_1) \beta_j(1) \tag{1.24}$$

注意在上面的定义中，前向概率是一个联合概率，而后向概率是一个条件概率。这两个有点不对称的定义是有意的，因为这两个概率的乘积决定了状态占有概率。从定义可推出：

$$\alpha_j(t) \beta_j(t) = P(O, x(t) = j \mid M) \tag{1.25}$$

因此，

$$L_j(t) = P(x(t) = j \mid O, M) = \frac{P(O, x(t) = j \mid M)}{P(O \mid M)} = \frac{1}{P} \alpha_j(t) \beta_j(t) \tag{1.26}$$

---

[3] 为了理解方程在 t 时刻包含了一个non-emitting状态，需要假定在 $t - \delta t$ 时刻是一个入口状态，而在 $t + \delta t$ 时刻是一个出口状态。当帧与帧之间HMMs需要通过non-emitting状态连接在一起时，这一点就变得很重要。

其中 $P = P(O|M)$。

使用Baum-Welch算法对HMM参数进行重估的所有知识都已经给出，下面总结一下该算法的步骤：

1. 对每个需要重估的参数向量或矩阵，给公式(1.13)和(1.14)的分子和分母分配存储空间，这些存储变量可以认为是一个累加器（*accumulators*[4]）。
2. 计算所有状态j和时刻t的前向和后向概率。
3. 对每个状态j和时刻t，用概率 $L_j(t)$ 和当前观察向量 o_t 更新该状态的*accumulators*。
4. 用最终的*accumulators*值求出新的参数值。
5. 如果本次迭代计算得到的 $P = P(O|M)$ 小于前一次的值就停止计算，否则，用新的重估参数值重复以上步骤。

以上所述都假设是用一个观察序列来估计HMM参数，一个单词只对应了一个样本。而在实际应用中，需要使用许多样本，这样可以得到更好的参数估计。可是用更多的观察序列对算法来说并不会增加额外的复杂度，对每个不同的训练序列重复执行上面的第2步和第3步即可。

最后需要提到一点，计算前向后向概率时会有非常多的概率相乘，在实际应用中，也就意味着实际数值变得很小，因此，为了避免数值问题，HTK使用log域计算[译1]来计算前向-后向概率。

在HTK中HRest执行上面的算法。先利用HInit工具估计出初始值，HRest就可以使用Baum-Welch重估算法从一些训练样本中构造出孤立单词的HMM了。

1.5 识别和 Viterbi 解码

前面一节讲述了使用 Baum-Welch 算法进行 HMM 参数重估的基本思想，顺便指出通过递归计算得到前向概率时也能求出总概率 $P(O|M)$。因此，这个算法也可以用来搜索产生最大 $P(O|M_i)$ 的模型，这样它也就能够用在识别上。

然而在实际中，基于最大似然状态序列的识别系统比基于总概率的识别系统更容易扩展到连续语音的情况。这一似然的计算，本质上是和前向概率的计算方法是一样的，只是不再求和，而是求最大值。给定一个模型 M，令 $\Phi_j(t)$ 表示在 t 时刻观测到语音序列从 o_1 到 o_t 并处于状态 j 的最大似然。用下面的公式递归计算得出这部分似然值 (参照公式 1.16)：

$$\Phi_j(t) = \max_i\{\Phi_i(t-1)a_{ij}\}b_j(o_t) \tag{1.27}$$

初始条件为：

$$\Phi_1(1) = 1 \tag{1.28}$$

$$\Phi_j(1) = a_{1j}b_j(o_1) \tag{1.29}$$

其中 $1 < j < N$。由下面的公式可以求出最大似然值 $\hat{P}(O|M)$：

---

[4] 需要指出的是重估公式的分母的累加和通常是和给定状态的参数是相同的，只需要一个通用的存储空间，而且只需要计算一次。然而，因为 HTK 支持参数绑定机制，这就导致分母值变化了，所以在 HTK 中，每个不同的参数向量或矩阵都独立存储和计算上述累加和。

[译1] 转换到 log 域上之后，乘法变成了加法。

$$\Phi_N(T) = \max_i \{\Phi_j(T) a_{iN}\} \tag{1.30}$$

当重估时，直接计算似然值会导致溢出，因此改用 log 似然，这样公式(1.27)就变成：

$$\Psi_j(t) = \max_i \{\Psi_i(t-1) + \log(a_{ij})\} + \log(b_j(o_t)) \tag{1.31}$$

这个递归计算过程就是基本的 Viterbi 算法，如图 1.6，这个算法可以看成是在一个矩阵里寻找最优路径，其中纵轴表示 HMM 的状态，横轴表示语音的帧数(也就是时间)。图中每个大的黑点表示这一时刻观测到这一帧的 log 似然，点与点之间的弧与 log 域转移概率对应。任何路径的 log 似然用沿着该路径的 log 域转移概率和 log 输出概率之和求出。自左向右一列一列地生成这些路径。在 t 时刻，所有状态 i 的每条路径 $\Psi_i(t-1)$ 已知，因此可以使用公式(1.31)计算 $\Psi_j(t)$，将路径扩展一帧。

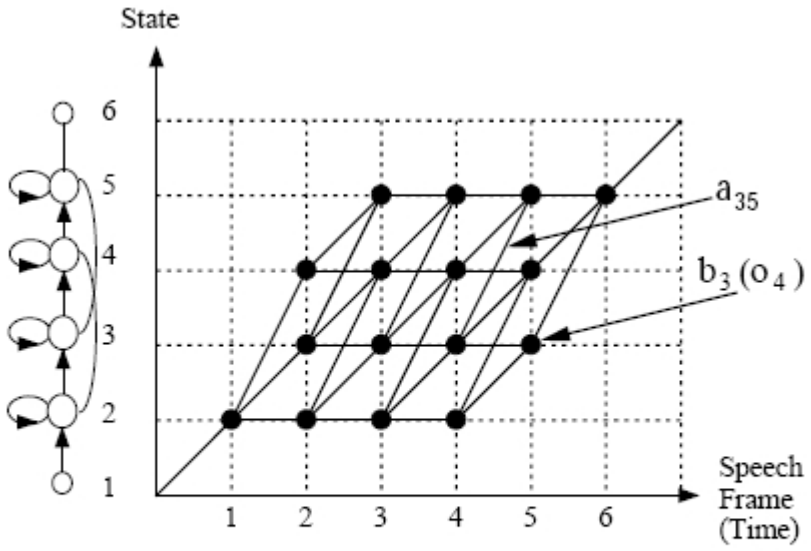


Fig. 1.6 The Viterbi Algorithm for Isolated Word Recognition

路径这一概念非常重要，并在连续语音的处理中普遍使用。

基于HMM的孤立词识别问题已经讨论完毕。HTK中并没有直接实现上述Viterbi算法的工具，取而代之的是提供了*HVite*和它的支持库*HNet*和*HRec*，用于处理连续语音。因为这个识别器有语法结构，所以可以将孤立词识别作为它的一个特殊情形处理，下面将予以详细地讨论。

1.6 连续语音识别

现在回到图1.1所示的语音产生和识别的概念模型，需要明白的是推广到连续语音只是简单地让HMM有序地连接在一起。序列中的每个模型和隐藏的符号直接对应。这些符号可能是针对连接语音识别（*Connected Speech Recognition*）的整个单词，或者是针对连续语音识别的子词（例如音素）。为什么存在没有输出的入口状态和出口状态呢？现在就很明显了，因为这些状态可以用来将各个模型连接在一起。

然而还得克服一些实际问题中遇到的困难。连续语音的训练数据必须包括连续语料，而一般对应到子词模型的每段语音的切分边界都是不知道的。在实际过程中，通常是人工标注少量的数据边界，这样，一个模型对应的所有语音片段都可以提取出来，也可以进行上述的孤立词训练。然而，用这种方法得到的数据是很有限的，并且模型估计结果也很差。此外，

即使有了大量的数据，人工强行标记的边界也可能不是 HMM 使用的最佳边界。因此，在 HTK 中使用 HInit 和 HRest 初始化子词模型可视为引导（*bootstrap*）操作[5]。主要的训练阶段还是用工具 HERest 来完成，它用于嵌入式训练（*Embedded Training*）。

同孤立词训练一样，嵌入式训练也使用 Baum-Welch 算法，不同的是前者对各个模型单独训练，而后者是所有模型并行训练。它具体的工作步骤如下：

1. 为所有模型的所有参数的*accumulators*分配空间，并置0。
2. 得到下一个训练语料。
3. 有序地联结和训练语料的符号脚本相对应的所有HMM，构造出一个复合的HMM。
4. 对这个复合的HMM计算前向-后向概率。在复合模型中，包含不输出的状态，因此前向概率和后向概率的计算有所改变，但变动很小，这在第八章中将详细地讨论。
5. 使用前向和后向概率计算在每一帧的状态占有概率，并用通常的方法更新*accumulators*。
6. 从第二步开始重复上述操作，直到处理完所有的训练语料。
7. 用*accumulators*计算所有HMM的新的参数估计值。

多次重复这些步骤，直到达到要求的收敛为止。需要指出的是，虽然这个过程没有要求训练数据中的符号边界位置，但每个训练语料的符号脚本还是必须的。

对于训练子词模型，Baum-Welch算法需要做的扩展相对Viterbi算法需要做的扩展来说要少一些[6]。

在HTK中，Viterbi算法的另一表述形式为*Token Passing Model*[7]。简单地说，*Token Passing Model*明确地给出状态对齐路径的概念。假设在 t 时刻HMM的每个状态 j 对应了一个可移动的token，该token在其它信息中包含了局部log概率 $\Psi_j(t)$。那么这一token就表示了观测序列 $o_1 \ldots o_t$ 与在 t 时刻进入状态 j 的模型之间的一个局部匹配。这时可用*token passing* 算法来代替公式(1.31)表示的路径扩充算法，它在每一帧 t 都要执行。这个算法的主要步骤如下：

1. 拷贝状态 i 中的每一个token，传给到所有的连接的状态 j，使用 $\log[a_{ij}]+\log[b_j(o(t))]$ 增加所拷贝的token的log概率。
2. 检查每个状态的token，留下最大概率的token，扔掉其他token。

在实际中，处理不输出的状态时需要一些修改，但这些比较简单，如果 token 在入口状态，就假设路径延伸到 $t-\delta t$，而在出口状态则延伸到 $t+\delta t$。

用 *Token Passing Model* 的关键是因为它能很简单地扩展到连续语音的情形。假设 HMM 序列是一个有限状态网络。例如，图 1.7 显示了一个简单的网络，每个单词定义为基于音素的 HMM 序列，所有的单词是放在一个圆环里。在这个网络里，椭圆框表示 HMM 实例，方框表示 *word-end* 节点。这个复合网络正是一个大的 HMM，应用了上面的 Token Passing 算法。现在唯一的不同是比最优 token 的 log 概率需要更多的信息。当最优 token 到达语音结尾时，为了恢复识别出的模型序列，我们必须知道它通过网络的路径。

---

[5] 用 *flat start*（将在 8.3 节讲述）可以不采用这样的方式。

[6] 实际中，为了对大语料库进行有效的操作，还需要做很多的额外工作。比如，HERest中包含的对前向和后向路径的裁剪，以及机器网络的并行处理等功能。

[7] 参见"Token Passing: a Conceptual Model for Connected Speech Recognition Systems", SJ Young, NH Russell and JHS Thornton, CUED Technical Report F_INFENG/TR38, Cambridge University, 1989. 可通过匿名登陆 ftp 服务器 svr-ftp.eng.cam.ac.uk 获得。

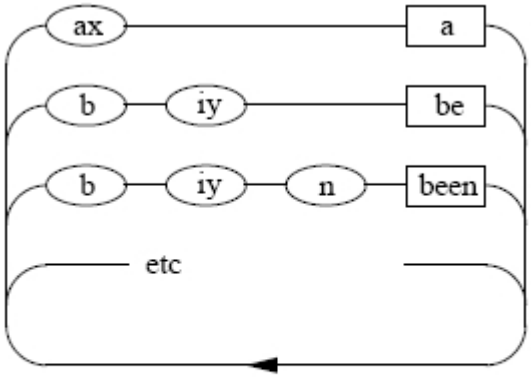


Fig. 1.7 Recognition Network for Continuously Spoken Word Recognition

穿过网络的token路径的历史记录可以按照下面方式有效地记录下来。每个token带有一个“*Word End Link*”指针，当一个token从一个单词的出口状态（用穿过word-end结点表示）传到另一个单词的入口状态产生一个路径时，这一传输表示了一个可能的单词边界。因此生成一个称为“*Word Link Record*”（WLR）的记录存储该单词的刚刚出现的token和token链的当前值。那么这个实际的token链就是用一个新的WLR指针来代替，图1.8说明了这一过程。

一旦处理完所有的未知语音，带有最优token（也就是最大log概率的token）链的WLR能够追溯得到最优匹配单词序列。同时如有需要，也能得到单词边界的位置。

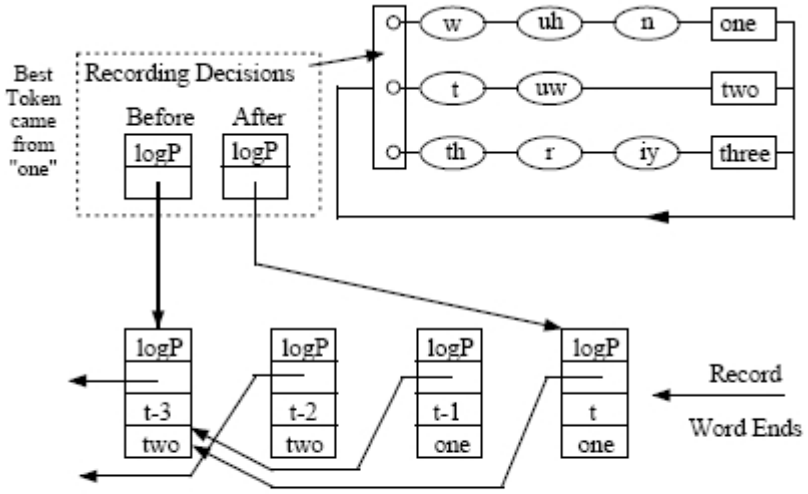


Fig. 1.8 Recording Word Boundary Decisions

上面针对连续语音的Token Passing算法描述仅仅用于记录单词序列。如果需要，该原理还能用来记录在模型和状态级的决策，也能保存每个单词边界上更多的最优路径。这样就为生成hypotheses（译者注：意为可能的识别结果）的lattice（译者注：网格）提供了可能，lattice比单Best输出更有用。基于这个思想的算法称为*lattice N-best*。因为每个状态一个token，限制了可能得到的不同token历史记录的数量，所以他们不是最合适的。若每个模型状态对应多个token，并且如果认为来自不同前序单词的token是不同的，就可以避免上述限制。这类算法称为*word N-best* 算法，经验表明它的性能可以和最优的N-best算法相当。

上面概述了用在HTK中的Token Passing的主要思想。这些算法植入到了库模块HNet和HRec中，它们会被识别工具HVite调用。它们提供单个和多个Token Passing识别，*single-best*输出，网格输出，N-best列表，支持上下文相关*cross-word*，*lattice rescoring*和forced alignment。

1.7 发音人自适应

虽然上面所讲的训练和识别技术能够产生出高效的识别系统，但如果对特定发音人的特征定制 HMM，那么这些系统的效果将得到进一步的提高。HTK 提供了 HEAdapt 和 HVite 工具，它们用少量的 *enrollment* 或自适应数据来执行自适应。这两个工具的不同之处在于，HEAdapt进行离线有监督的适应，而HVite 识别自适应数据并使用生成的脚本来进行自适应。通常，HEAdapt 提供的有监督的自适应更稳定一些，但是给定一个初始的已经很好地训练过的模型集，HVite 仍然能显著地改善其效果。在第九章将描述自适应的细节及它在 HTK 中的使用。