

Reconocimiento Automático del Habla

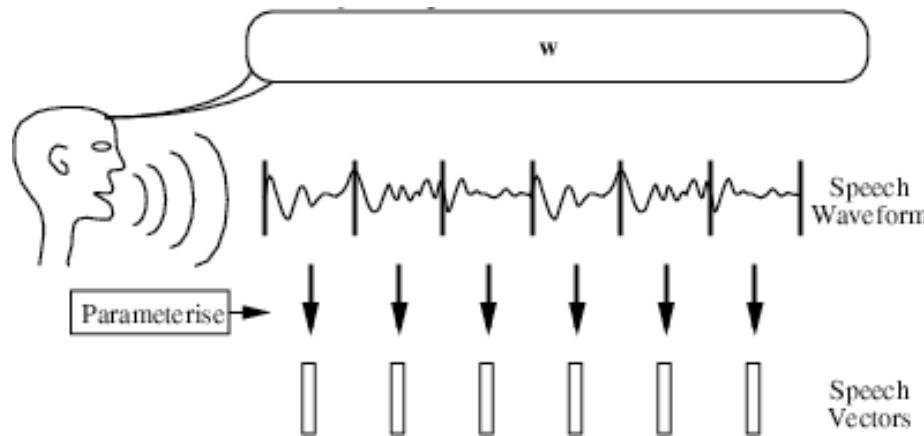
Modelos Ocultos de Markov

María José Castro

mcastro@dsic.upv.es

Secuencias

Al hablar emitimos una secuencia de **observaciones eventos acústicos**.



Podemos considerar que las observaciones son:

- **discretas**: símbolos de un alfabeto (resultado de una cuantificación vectorial, etc.),
- **continuas**: valores reales o vectores de valores reales (la energía de la señal acústica en instantes sucesivos, vectores de valores que describen el espectro en instantes sucesivos, etc.).

¿Cómo caracterizamos esas secuencias?

- **Modelos deterministas:** describen la señal como suma de senoides, o suma de funciones exponenciales, etc. La caracterización es directa: se determinan los parámetros correspondientes.
- **Modelos estadísticos:** se asume que la señal puede describirse mediante un proceso aleatorio parametrizado. Se describen propiedades estadísticas de la señal.

¿Cómo caracterizamos esas secuencias?

- **Modelos deterministas:** describen la señal como suma de senoides, o suma de funciones exponenciales, etc. La caracterización es directa: se determinan los parámetros correspondientes.
- **Modelos estadísticos:** se asume que la señal puede describirse mediante un proceso aleatorio parametrizado. Se describen propiedades estadísticas de la señal.

Hay buenas razones para suponer que el habla se puede modelar adecuadamente como un proceso estocástico:

- El mismo sonido/fonema/palabra suenan diferentes con cada pronunciación.
- Podemos suponer que, al hablar, se transita “aleatoriamente” entre diferentes configuraciones del tracto vocal y en cada configuración se emiten fonemas siguiendo alguna distribución probabilística.

Las Cadenas de Markov y los Modelos Ocultos de Markov (HMM, por Hidden Markov Model) son modelos estadísticos que proporcionan descripciones de secuencias de eventos y que presentan (entre otras) una gran ventaja frente a las plantillas: pueden entrenarse con muchas pronunciaciones y, al descodificar, el coste computacional depende básicamente del número de modelos, no del número de pronunciaciones con que fueron entrenados.

Se conocen desde los años 60 y se aplicaron con éxito en el reconocimiento del habla en los 70 por Jelinek y otros (IBM). Su uso se extendió en los 80.

Cadenas de Markov

Definición

Consideremos un sistema en el que hay N estados etiquetados con números $1, 2, \dots, N$. En instantes de tiempo distribuidos regularmente, $t = 0, 1, 2, \dots$, el sistema transita de un estado a otro (posiblemente al mismo).

Sea x_t el estado en el que se está en el instante t . ¿Con qué probabilidad estamos en el estado j en el instante t ? Un modelado completo hace depender la probabilidad de todos los estados anteriores, es decir, de la **historia**:

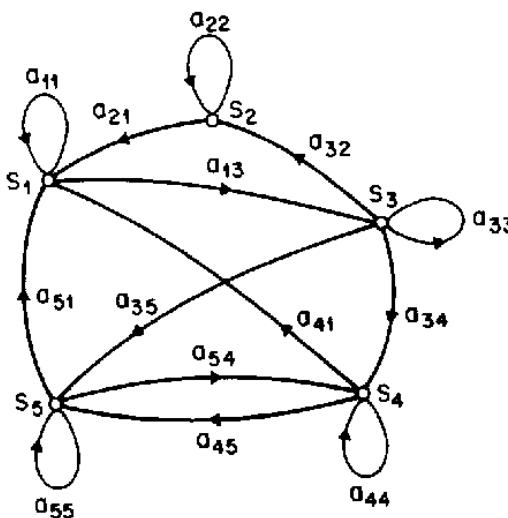
$$P(x_t = j \mid x_{t-1} = i, x_{t-2} = h, \dots, x_0 = a)$$

Una **Cadena de Markov** de primer orden limita la historia al estado anterior (**asunción markoviana**):

$$P(x_t = j \mid x_{t-1} = i, x_{t-2} = h, \dots, x_0 = a) = P(x_t = j \mid x_{t-1} = i).$$

Además, la probabilidad de “saltar” de un estado i a otro j se considera **independiente del instante t** en que tiene lugar:

$$P(x_t = j \mid x_{t-1} = i) = a_{ij}, \quad 1 \leq i, j \leq N.$$



Cadena de Markov con 5 estados

Las transiciones entre estados satisfacen las **restricciones estocásticas**:

$$a_{ij} \geq 0, \quad 1 \leq i, j \leq N;$$
$$\sum_{j=1}^N a_{ij} = 1, \quad 1 \leq i \leq N.$$

En $t = 0$ se puede estar en cualquiera de los N estados de acuerdo con una distribución de probabilidad $\Pi = \{\pi_i\}$, donde $\pi_i = P(x_0 = i)$:

$$\sum_{i=1}^N \pi_i = 1.$$

En un Cadena de Markov los estados son observables: la secuencia a modelar es la secuencia de estados.

Un ejemplo

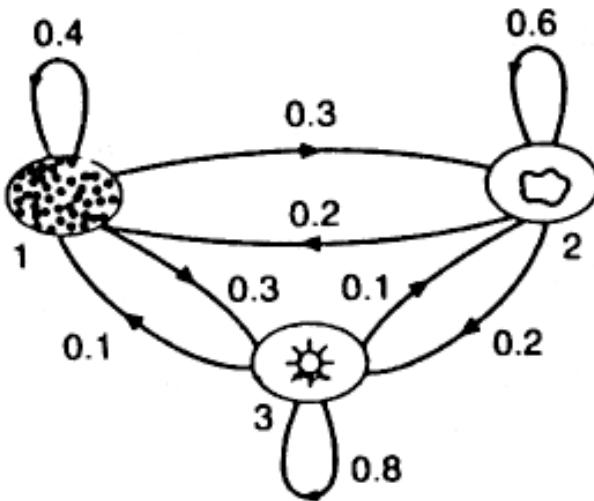
Estados:

1. Lluvia.
2. Nubes.
3. Sol.

Probabilidad de transición:

$$A = \{a_{ij}\} = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{pmatrix},$$

$$\Pi = \{\pi_i\} = (0.0, 0.0, 1.0).$$



Cadena de Markov con 3 estados: sol, nubes, lluvia.

¿Con qué probabilidad el tiempo de 8 días es “sol-sol-sol-lluvia-lluvia-sol-nubes-sol”?

Formalmente: la probabilidad de la observación $O = 3, 3, 3, 1, 1, 3, 2, 3$ es

$$\begin{aligned}
 P(O \mid \text{Modelo}) &= P(3, 3, 3, 1, 1, 3, 2, 3 \mid \text{Modelo}) \\
 &= P(3) \cdot P(3 \mid 3) \dots \cdot P(3 \mid 2) \\
 &= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{13} \cdot a_{11} \cdot a_{31} \cdot a_{23} \cdot a_{32} \\
 &= 1.536 \cdot 10^{-4}
 \end{aligned}$$

En aplicaciones reales observamos eventos físicos. Las Cadenas de Markov representan dichos eventos con los propios estados.

Modelos Ocultos de Markov

Procesos estocásticos a dos niveles

Los Modelos Ocultos de Markov son procesos estocásticos a dos niveles:

- el de los **estados**, que *no* es observable,
- y el de **eventos físicos**, que *sí* es observable.

Decimos que los eventos físicos se **emiten** en los estados.

Ejemplos

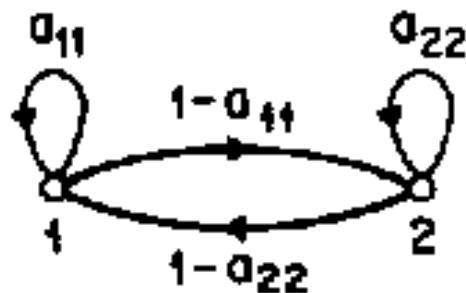
El modelo cara/cruz Estamos en una habitación con una cortina que nos impide ver a una persona que tiene dos monedas cargadas. Esa persona lanza repetidamente una u otra (aleatoriamente) al aire y nos informa de si sale cara (H, por *head*) o cruz (T, por *tail*).

Un ejemplo de secuencia de observaciones es éste:

HHTTHHTHHTTHHTHHTTTHTHHTHHT

No tenemos forma de saber cada H o T de qué moneda es. ¿Cómo explicamos/modelamos las secuencias que observamos?

Podríamos modelar este escenario con un HMM. Cada estado correspondería a una moneda. El hecho de que la persona cambia de moneda aleatoriamente correspondería al hecho de cambiar de estado. La asunción markoviana impone que la probabilidad de coger una u otra moneda dependa exclusivamente de la última moneda lanzada.



$$P(H) = P_1 \quad P(H) = P_2$$

$$P(T) = 1 - P_1 \quad P(T) = 1 - P_2$$

$O = HHTTHHTHHTTH\dots$

$S = 21122212212\dots$

En cada estado hay una probabilidad de “emitir” H o T.

Para completar el modelo, hemos de conocer el valor de 4 parámetros:

- a_{11} ,
- a_{22} ,
- $P_1(H)$,
- $P_2(H)$.

Sólo son 4 porque el resto se puede deducir de ellos:

- $a_{12} = 1 - a_{11}$,

- $a_{21} = 1 - a_{22}$,
- $P_1(T) = 1 - P_1(H)$,
- $P_2(T) = 1 - P_2(H)$.

¿Podemos estimar esos cuatro valores observando únicamente el resultado de una o más secuencias de lanzamientos?

El modelo urna/bola Tratemos con una situación más complicada. Hay N urnas con bolas de M colores distintos. Un diablo actúa tras una cortina que oculta las urnas de acuerdo con el siguiente procedimiento:

1. escoge una urna inicial,
2. extrae una bola y nos la muestra,
3. restituye la bola a la urna de la que la extrajo,
4. selecciona una nueva urna de acuerdo con ciertas probabilidades asociadas a la urna de la que extrajo al última bola,
5. vuelve al paso 2 si aún no se ha extraído un número predeterminado de bolas..



URN 1

$$P(\text{RED}) = b_1(1)$$

$$P(\text{BLUE}) = b_1(2)$$

$$P(\text{GREEN}) = b_1(3)$$

$$P(\text{YELLOW}) = b_1(4)$$

:

$$P(\text{ORANGE}) = b_1(M)$$



URN 2

$$P(\text{RED}) = b_2(1)$$

$$P(\text{BLUE}) = b_2(2)$$

$$P(\text{GREEN}) = b_2(3)$$

$$P(\text{YELLOW}) = b_2(4)$$

:

$$P(\text{ORANGE}) = b_2(M)$$

...



URN N

$$P(\text{RED}) = b_N(1)$$

$$P(\text{BLUE}) = b_N(2)$$

$$P(\text{GREEN}) = b_N(3)$$

$$P(\text{YELLOW}) = b_N(4)$$

:

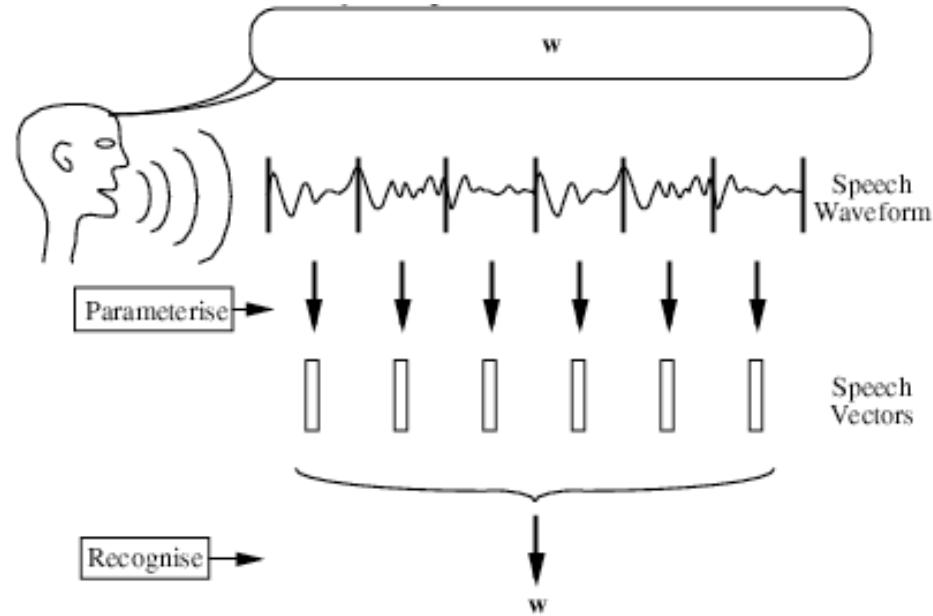
$$P(\text{ORANGE}) = b_N(M)$$

Si sólo podemos ver las bolas que extrae el diablo, ¿cómo modelamos el sistema?

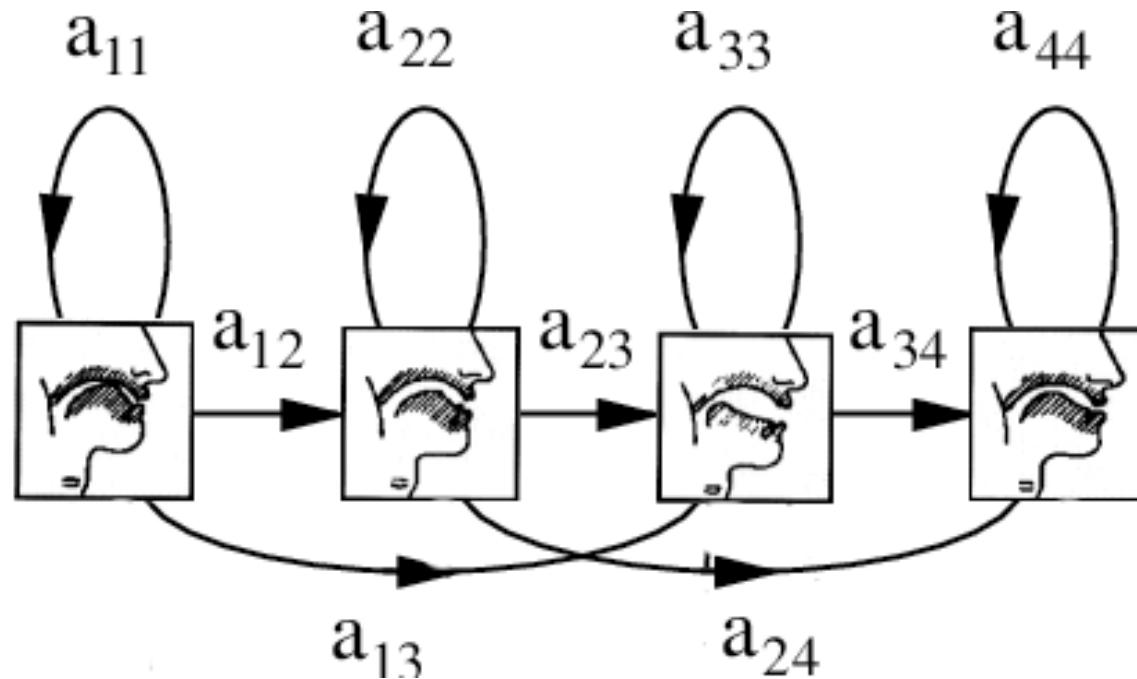
Un HMM modela este escenario con:

- un **estado** por urna,
- una **probabilidad de transición** entre estados que supedita la elección del siguiente estado a (únicamente) el estado actual,
- una probabilidad de “emisión de bolas de color” asociada a cada urna (dada por la proporción de bolas de cada color en la urna),
- una **probabilidad** de que cada urna sea la “urna **inicial**”.

El habla

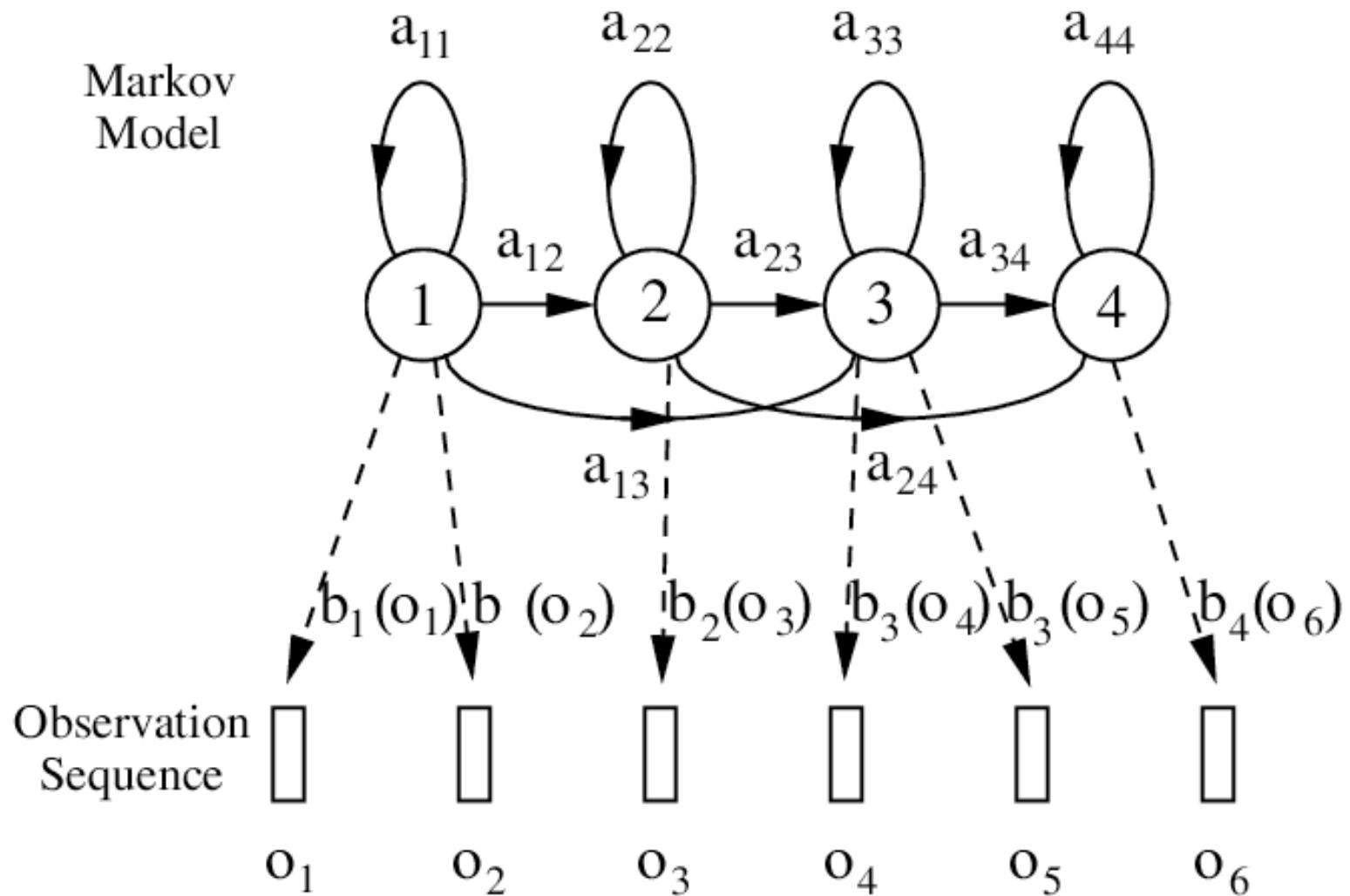


Asimilamos estados a configuraciones del aparato fonador. De una configuración se puede pasar a otra de acuerdo con ciertas reglas probabilísticas.



En cada estado se puede emitir un sonido de entre los de un conjunto con cierta distribución de probabilidad.

Markov
Model



Formalismo

Un modelo oculto de Markov se caracteriza por:

- Un conjunto de N **estados**.
- Un espacio de **características observables**. Asumiremos por el momento que es un conjunto de M símbolos (Modelos Ocultos de Markov **discretos**).
- Una matriz $A = \{a_{ij}\}$ de **probabilidad de transición** entre estados.

El elemento a_{ij} indica la probabilidad de transitar al estado j si se está en el estado i .

Se cumple:

$$a_{ij} \geq 0, \quad 1 \leq i, j \leq N;$$

$$\sum_{j=1}^N a_{ij} = 1, \quad 1 \leq i \leq N.$$

- Una distribución de **probabilidad de emisión** de símbolos en cada estado: $B = \{b_j(k)\}$, para $1 \leq j \leq N$, $1 \leq k \leq M$.

El elemento $b_j(k)$ indica la probabilidad de emitir una observación k en el estado j .

Se cumple:

$$b_i(k) \geq 0, \quad 1 \leq i \leq N, \quad 1 \leq k \leq M;$$

$$\sum_{k=1}^M b_i(k) = 1, \quad 1 \leq i \leq N.$$

- Una distribución de **probabilidad de estado inicial**: $\Pi = \{\pi_i\}$, para $1 \leq i \leq N$.

El elemento π_i indica la probabilidad de que el primer estado en el que estamos (y desde el que se emite el primer símbolo) sea el estado i .

Se cumple:

$$\pi_i \geq 0, \quad 1 \leq i \leq N;$$

$$\sum_{i=1}^N \pi_i = 1, \quad 1 \leq i \leq N.$$

Un modelo queda descrito, pues, con $\lambda = (A, B, \Pi)$.

Un formalismo alternativo: emisión en arcos

Ciertos autores (los de IBM entre otros) prefieren que la emisión de símbolos tenga lugar en los arcos. En lugar de $b_j(k)$ se define

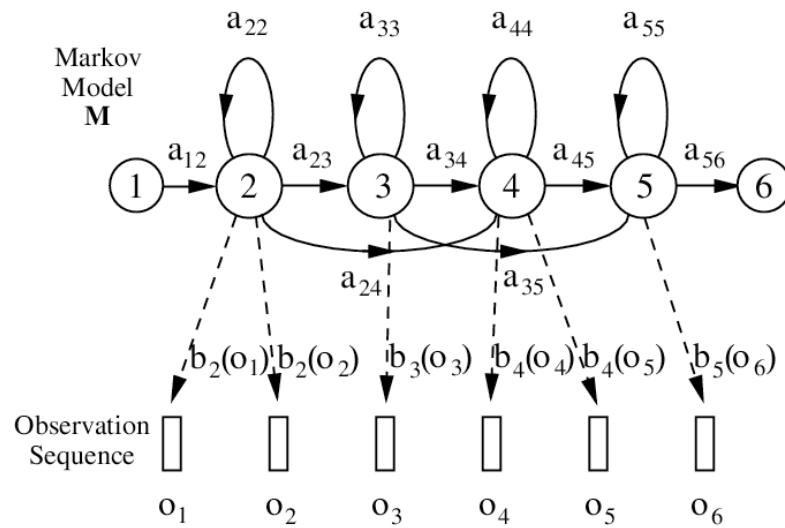
$$b_{ij}(k) = P(o_t = k \mid x_{t-1} = i, x_t = j, \lambda).$$

Esta formulación plantea ciertos problemas al componer HMM, pues surge la necesidad de tratar con *transiciones nulas* (no emisoras) que complican ligeramente los algoritmos de búsqueda.

Otro formalismo alternativo

Presentamos un formalismo ligeramente diferente que, siendo equivalente, simplifica ciertas operaciones y cálculos.

- Sólo se puede empezar en el estado 1, que no tiene arcos de llegada y no emite símbolos ($\pi_1 = 1$).
- Toda secuencia finaliza en el estado N , que no tiene arcos de salida y no emite símbolos.



Una secuencia de estados que emite T símbolos es de la forma $x_0x_1x_2\dots x_{T+1}$, donde $x_0 = 1$ y $x_{T+1} = N$.

Dado que no es necesario especificar Π , asumimos que un modelo λ viene dado por (A, B) .

He aquí (el principio de) una implementación en Python para HMM discretos:

```
from UserDict import UserDict

class ProbDict(UserDict): # Devuelve 0.0 para claves inexistentes
    def __init__(self, dict={}):
        self.data = dict
    def __getitem__(self, key):
        return self.data.get(key, 0.0)

class HMM:
    def __init__(self, N, a=None, b=None):
        self.N = N                  # Número de estados
        self.a = ProbDict(a)        # Matriz de probabilidad de transiciones
        self.b = ProbDict(b)        # Matriz de probabilidades de emisión de símbolos
        # Sucesores y predecesores de cada estado
        self.succ = [ [] for i in range(N) ]
        self.pred = [ [] for i in range(N) ]
        for i, j in self.a.keys():
            self.succ[i].append(j)
            self.pred[j].append(i)
```

...

```
if __name__ == "__main__":
    H = HMM(5,
        # a
        {(0,1): 1.0,
         (1,1): 0.4, (1,2): 0.3, (1,3): 0.3,
         (2,1): 0.6, (2,2): 0.3, (2,3): 0.1,
         (3,1): 0.5, (3,2): 0.2, (3,3): 0.2, (3,4): 0.1},
        # b
        { (1, 'a') : 0.2, (1, 'b') : 0.1, (1, 'c') : 0.7,
          (2, 'a') : 0.3, (2, 'b') : 0.3, (2, 'c') : 0.4,
          (3, 'a') : 0.5, (3, 'b') : 0.3, (3, 'c') : 0.2})
```

HMM como modelos generadores

Desde un punto de vista de generador de secuencias, un HMM funciona así:

1. Fija t a cero y empieza en el estado $x_0 = 1$.
2. Transita al estado x_{t+1} desde x_t con probabilidad $a_{x_t x_{t+1}}$ y fija t a $t + 1$.
3. Si llega al estado N , termina.
4. Escoge o_t de acuerdo con la distribución b_{x_t} . Vuelve al paso 2.

Topologías típicas en Reconocimiento Automático del Habla

Los HMM se representan gráficamente mediante diagramas de estados. Las transiciones de probabilidad 0 no se representan.



En reconocimiento del habla tratamos de modelar un proceso que ocurre a lo largo de tiempo.

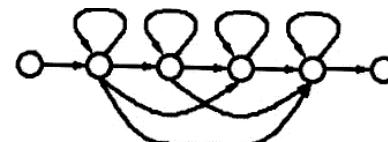
La **topología** de los modelos trata de capturar ese flujo de información temporal:

- **Modelo lineal:**



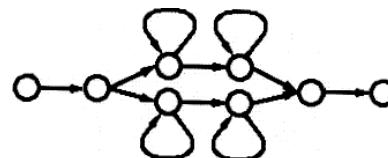
Cada estado emite uno o más símbolos. Se emiten al menos tantos símbolos como estados hay.

- **Modelo de izquierda-a-derecha:**



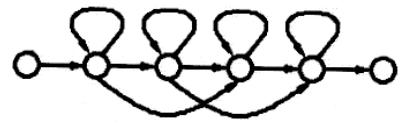
Permite modelar la omisión de uno o más estados gracias a los saltos de más de un estado.

- **Modelo con caminos alternativos:**



Permite modelar pronunciaciones alternativas: o bien se discurre por los estados superiores, o bien por los inferiores.

- **Modelo de Bakis:**



Cada estado se apunta a sí mismo, al siguiente y al que sigue al siguiente. Con 3 estados, es una topología simple muy usada para modelar fonemas: cada estado modela una parte de la pronunciación del fonema (zona inicial, zona media, zona final).

- ...

Los tres problemas básicos de los HMM

Problema 1: El problema de la evaluación Dada una secuencia de observaciones $O = o_1 o_2 \dots o_T$ y un modelo $\lambda = (A, B)$, ¿cómo calculamos eficientemente la probabilidad de que λ genera a O , es decir, $P(O | \lambda)$?

Problema 2: El problema de la descodificación Dada una secuencia de observaciones $O = o_1, o_2, \dots, o_T$ y un modelo $\lambda = (A, B)$, ¿cómo calculamos la secuencia de estados x_0, x_1, \dots, x_{T+1} que mejor “explica” las observaciones?

Problema 3: El problema del entrenamiento Dada una secuencia de observaciones O , ¿cómo estimamos los parámetros de $\lambda = (A, B)$ para que $P(O | \lambda)$ sea máxima?

Problema 1: El problema de la evaluación

Dada una secuencia de observaciones $O = o_1, o_2, \dots, o_T$ y un modelo $\lambda = (A, B)$, ¿Cómo calculamos eficientemente $P(O | \lambda)$?

Podríamos enumerar todas las secuencias posibles de $T + 2$ estados (que emiten un total de T símbolos), calcular para cada una de ellas la probabilidad de generar O y sumar todas estas probabilidades ponderándolas por la probabilidad de cada secuencia.

Tomemos una secuencia cualquiera:

$$X = x_0 x_1 \dots x_{T+1}.$$

1. La probabilidad de observar O dados X y un modelo λ es

$$P(O | X, \lambda) = \prod_{t=1}^T P(o_t | x_t, \lambda) = \prod_{t=1}^T b_{x_t}(o_t).$$

2. La probabilidad de la secuencia de estados X es:

$$P(X | \lambda) = a_{x_0 x_1} \cdot a_{x_1 x_2} \cdot \dots \cdot a_{x_T x_{T+1}}.$$

3. La probabilidad conjunta es:

$$P(O, X \mid \lambda) = P(O \mid X, \lambda) \cdot P(X \mid \lambda)$$

Finalmente, la probabilidad de O dado λ es:

$$\begin{aligned} P(O \mid \lambda) &= \sum_X P(O \mid X, \lambda) \cdot P(X \mid \lambda) \\ &= \sum_{x_0 x_1 \cdots x_{T+1}} a_{x_0 x_1} \cdot b_{x_1}(o_1) \cdot a_{x_1 x_2} \cdot \dots \cdot a_{x_{T-1} x_T} \cdot b_{x_T}(o_T) \cdot a_{x_T x_{T+1}} \\ &= \sum_{x_0 x_1 \cdots x_{T+1}} a_{x_0 x_1} \cdot \prod_{t=1}^T b_{x_t}(o_t) \cdot a_{x_t x_{t+1}}. \end{aligned}$$

Un cálculo directo de la fórmula es $O(TN^T)$

¿Se puede calcular más eficientemente?

El algoritmo forward Definimos la “**probabilidad forward**” como la probabilidad de observar la secuencia parcial $o_1 o_2 \cdots o_t$ y estar en el estado j de un modelo λ en el instante t :

$$\alpha_j(t) = P(o_1 o_2 \cdots o_t, x_t = j \mid \lambda).$$

El valor de $\alpha_j(t)$ puede calcularse recursivamente:

$$\alpha_j(t) = \begin{cases} 1, & \text{si } t = 0 \text{ y } j = 1; \\ \left(\sum_{i=1}^{N-1} \alpha_i(t-1) \cdot a_{ij} \right) b_j(o_t), & \text{si } 1 < t \leq T \text{ y } 1 < j < N; \\ \sum_{i=1}^{N-1} \alpha_i(T) \cdot a_{iN}, & \text{si } t = T + 1 \text{ y } j = N; \\ 0, & \text{en otro caso.} \end{cases}$$

La probabilidad de observar una secuencia O dado el modelo λ es

$$P(O \mid \lambda) = \alpha_N(T + 1).$$

La recursión de α puede expresarse también así:

$$\alpha_j(t) = \begin{cases} 1, & \text{si } t = 0 \text{ y } j = 1; \\ \left(\sum_{i:a_{ij} \neq 0} \alpha_i(t-1) \cdot a_{ij} \right) b_j(o_t), & \text{si } 1 \leq t \leq T \text{ y } 1 < j < N; \\ \sum_{i:a_{iN} \neq 0} \alpha_i(T) \cdot a_{iN}, & \text{si } t = T + 1 \text{ y } j = N; \\ 0, & \text{en otro caso.} \end{cases}$$

Esta versión hace explícito que sólo hemos de considerar los estados i “predecesores” de cada estado j .

El valor de α puede calcularse eficientemente mediante un algoritmo iterativo (programación dinámica):

```
...
class HMM:
    ...
    def forward(self, O):
        alpha = [ [0.0] * self.N  for i in range(len(O)+2) ]
        alpha[0][0] = 1.0
        for t in xrange(1, len(O)+1):
            for j in xrange(1, self.N-1):
                for i in self.pred[j]:
                    alpha[t][j] += alpha[t-1][i] * self.a[i, j]
                    alpha[t][j] *= self.b[j, O[t-1]]
        alpha[len(O)+1][self.N-1] = 0.0
        for i in self.pred[self.N-1]:
            alpha[len(O)+1][self.N-1] += alpha[len(O)][i] * self.a[i, self.N-1]
        return alpha

    def probability(self, O):
```

```

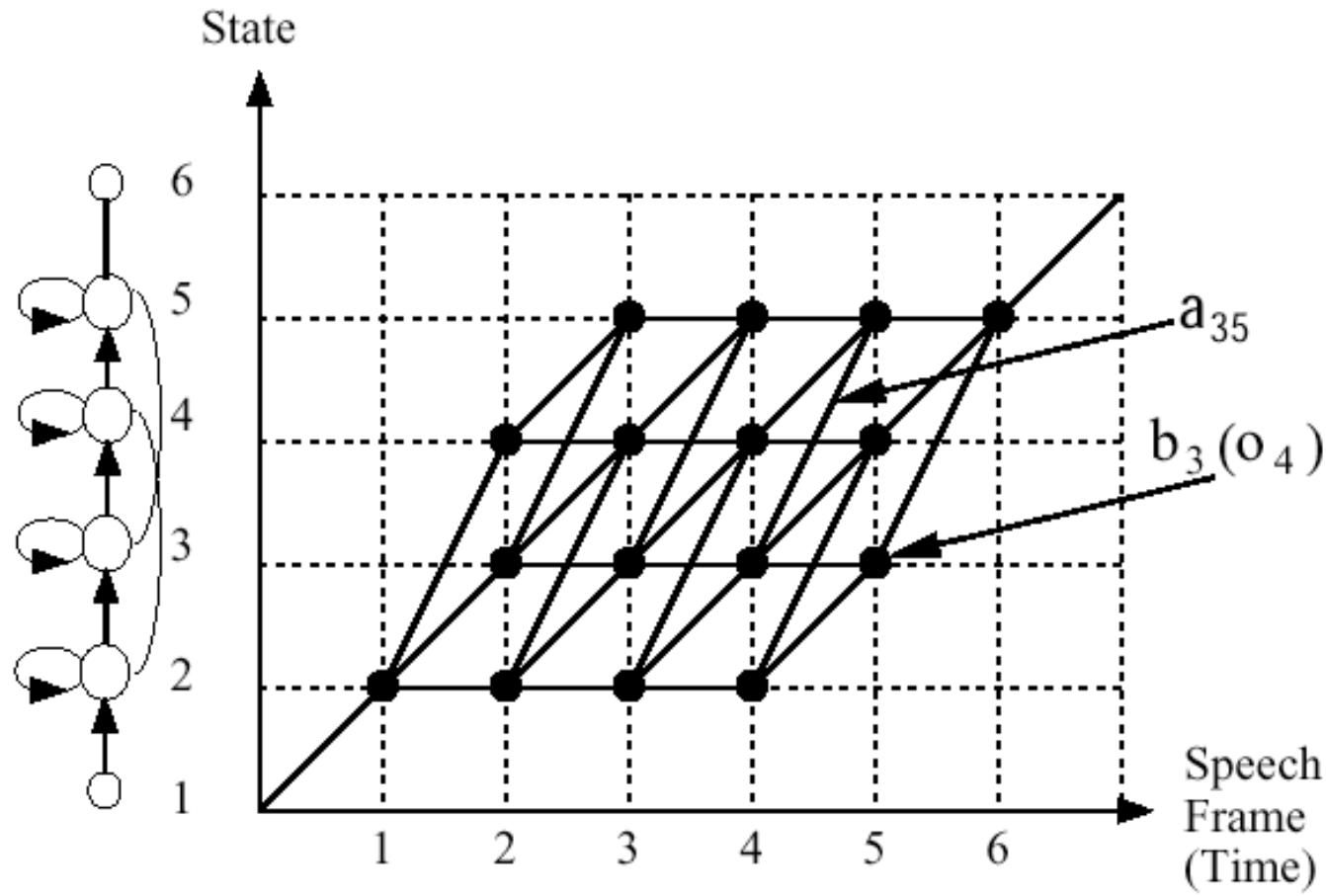
alpha = self.forward(0)
return alpha[len(0)+1] [self.N-1]

...
if __name__== "__main__":
    H = HMM(5,
            ...
        )
print H.forward('aabccc')

```

El coste temporal del algoritmo es $O(N^2T)$.

El *trellis* o grafo de programación dinámica del algoritmo es éste:



Trellis para HMM con topología de Bakis.

El procedimiento backward También podemos definir la “probabilidad backward”, $\beta_i(t)$. Es la probabilidad de la observación parcial $o_{t+1} o_{t+2} \dots o_T$ partiendo del estado i del modelo λ en el instante t :

$$\beta_i(t) = P(o_{t+1} o_{t+2} \dots o_T \mid x_t = i, \lambda).$$

Tenemos que:

$$P(O \mid \lambda) = \beta_1(0).$$

Se puede calcular β recursivamente:

$$\beta_i(t) = \begin{cases} a_{iN}, & \text{si } t = T; \\ \sum_{j=2}^{N-1} a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_j(t+1), & \text{si } 0 \leq t < T \text{ y } 1 \leq i < N; \\ 0, & \text{en otro caso.} \end{cases}$$

O, alternativamente,

$$\beta_i(t) = \begin{cases} a_{iN}, & \text{si } t = T; \\ \sum_{j:a_{ij} \neq 0} a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_j(t+1), & \text{si } 0 \leq t < T \text{ y } 1 \leq i < N; \\ 0, & \text{en otro caso.} \end{cases}$$

Las ecuaciones se pueden resolver iterativamente en $O(TN^2)$.

```
class HMM:  
    ...  
    def backward(self, O):  
        beta = [ [0.0] * self.N for i in xrange(len(O)) ]  
        beta.append( [ self.a[i,self.N-1] for i in xrange(self.N) ] )  
        for t in xrange(len(O)-1, -1, -1):  
            for i in xrange(self.N):  
                for j in self.succ[i]:  
                    beta[t][i] += self.a[i, j] * self.b[j, O[t]] * beta[t+1][j]  
        return beta  
    ...
```

Problema 2: El problema de la secuencia óptima de estados

¿Qué entendemos por secuencia óptima de estados? Es la secuencia de estados que con mayor probabilidad ha producido la secuencia de observaciones, es decir, la que maximiza

$$P(X | O, \lambda).$$

Puntuación de Viterbi Dada una secuencia de observaciones O y un modelo λ , definimos $\phi_t(j)$ como la probabilidad de la secuencia de estados que finaliza en el estado j , y más probablemente genera el prefijo $o_1 o_2 \cdots o_t$:

$$\phi_t(j) = \max_{x_0 x_1 \cdots x_t} P(x_0 x_1 \cdots x_t, x_t = j, o_1 \cdots o_t | \lambda).$$

$\phi_j(t)$ es la **puntuación de Viterbi**.

Podemos calcular recursivamente $\phi_t(j)$ (Viterbi):

$$\phi_j(t) = \begin{cases} 1, & \text{si } t = 0 \text{ y } j = 1; \\ \max_{1 \leq i < N} (\phi_i(t-1) \cdot a_{ij}) \cdot b_j(o_t), & \text{si } 1 \leq t \leq T \text{ y } 1 < j < N; \\ \max_{1 \leq i < N} \phi_i(T) \cdot a_{iN}, & t = T + 1 \text{ y } j = N; \\ 0, & \text{en otro caso.} \end{cases}$$

La secuencia más probable dada una secuencia de observaciones O tiene probabilidad

$$\max_X P(X \mid O, \lambda) = \phi_N(T).$$

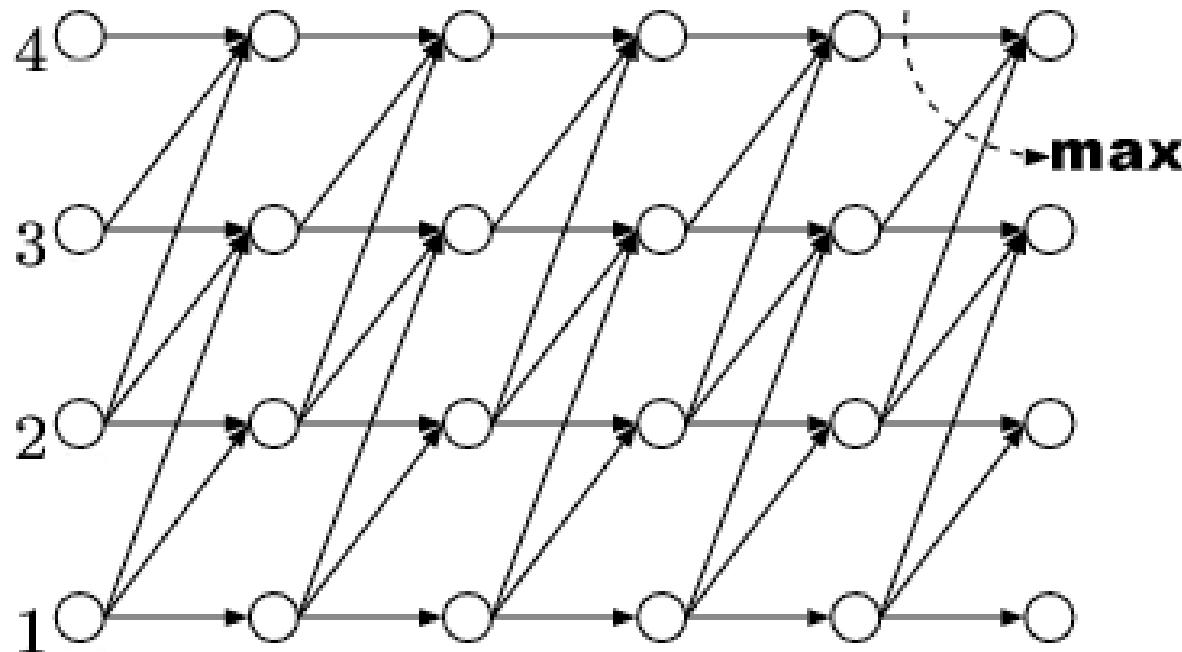
Nuevamente, podemos reescribir la ecuación recursiva para hacer explícito que la recursión

sólo considera estados predecesores:

$$\phi_j(t) = \begin{cases} 1, & \text{si } t = 0 \text{ y } j = 1; \\ \max_{i:a_{ij} \neq 0} (\phi_i(t-1) \cdot a_{ij}) \cdot b_j(o_t), & \text{si } 1 \leq t \leq T \text{ y } 1 < j < N; \\ \max_{i:a_{iN} \neq 0} \phi_i(T) \cdot a_{iN}, & t = T + 1 \text{ y } j = N; \\ 0, & \text{en otro caso.} \end{cases}$$

Este algoritmo recursivo puede transformarse en otro iterativo equivalente por programación dinámica:

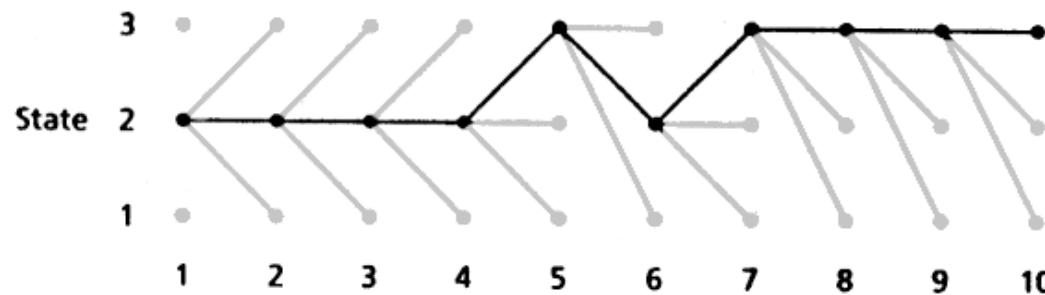
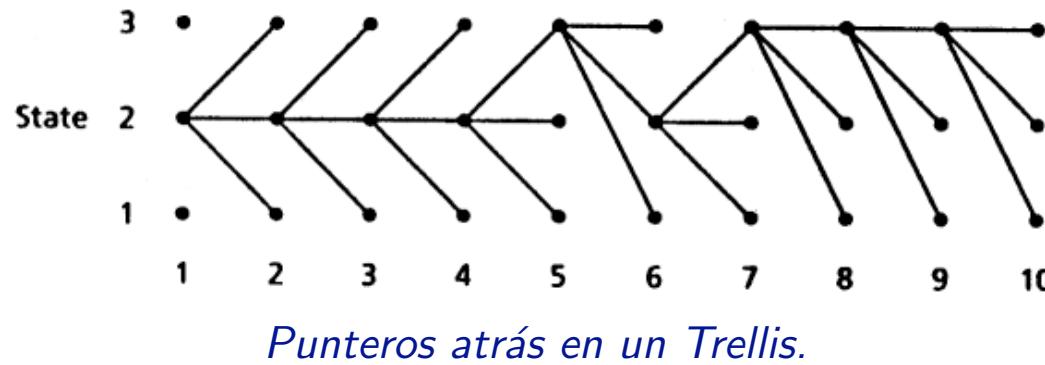
```
class HMM:  
    ...  
    def viterbi(self, O):  
        prev, score = [0.0] * self.N, [1.0] + [0.0] * (self.N-1)  
        for t in range(len(O)):  
            prev, score = score, prev  
            for j in range(1, self.N-1):  
                score[j] = max([prev[i] * self.a[i, j] * self.b[j, O[t]]]  
                               for i in self.pred[j]])  
        return max([score[i] * self.a[i, self.N-1] for i in range(1, self.N-1)])  
    ...
```



Trellis para la puntuación de Viterbi.

Recuperación de la secuencia de estados No nos interesa tanto $\phi_j(t)$ como la propia secuencia de estados que produjo el valor óptimo.

Resulta sencillo extraer la secuencia óptima de estados haciendo *backtracing*:



Camino óptimo desde el nodo $(10, 3)$ siguiendo los punteros hacia atrás en un Trellis.

```

class HMM:
    ...
    def viterbi_path(self, O):
        prev, score = [0.0] * self.N, [0.0] * self.N
        score[0] = 1.0
        backpointer = [ [None] * (self.N) for i in range(len(O)+1)  ]
        for t in range(len(O)):
            prev, score = score, prev
            for j in range(1, self.N-1):
                score[j], backpointer[t][j] = max([(prev[i]*self.a[i,j]*self.b[j,O[j]] )  

                                                    for i in self.pred[j]]])
            score[0] = 0
            score[self.N-1], backpointer[len(O)][self.N-1] = max([(score[i]*self.a[i,score[i]] )  

                                                               for i in self.pred[se
path = [ self.N-1 ]
for t in range(len(O), -1, -1):
    path.append( backpointer[t][path[-1]] )
path.reverse()
return (score[self.N-1], path)

```

...

Una ventaja de la puntuación de Viterbi La secuencia de estados que **maximiza la probabilidad** de generar una secuencia determinada es la misma que **maximiza el logaritmo de dicha probabilidad**.

Trabajar con logaritmos de probabilidad (**logprob**) transforma los productos en sumas, operaciones menos costosas computacionalmente y que, además, evitan la comisión de *underflows*:

$$\phi'_j(t) = \begin{cases} 0, & \text{si } t = 0 \text{ y } j = 1; \\ \max_{i:a_{ij} \neq 0} (\phi'_i(t-1) + \log a_{ij}) + \log b_j(o_t), & \text{si } 1 \leq t \leq T \text{ y } 1 < j < N; \\ \max_{i:a_{iN} \neq 0} \phi'_i(T) + \log a_{iN}, & t = T + 1 \text{ y } j = N; \\ \infty, & \text{en otro caso.} \end{cases}$$

Nota: es frecuente plantear la recursión como minimización y trabajar con el logaritmo cambiado de signo.

Problema 3: El problema del entrenamiento

Se trata del más difícil de los tres problemas.

No hay forma conocida de determinar analíticamente qué valores de $\lambda = (A, B)$ maximizan la probabilidad de $P(O | \lambda)$.

Reestimación de parámetros Queremos reestimar $A = \{a_{ij}\}$ y $B = \{b_j(k)\}$ de modo tal que se maximice

$$P(O | \lambda).$$

Si partimos de A y B arbitrarios es posible obtener una estimación $\bar{A} = \{\bar{a}_{ij}\}$ y $\bar{B} = \{\bar{b}_j(k)\}$ que aumente o deje igual el valor de $P(O | \lambda)$ mediante un procedimiento iterativo.

$$\begin{aligned}
\bar{a}_{ij} &= \frac{\text{número esperado de veces que se usa la transición } i, j \text{ al generar } O}{\text{número esperado de veces que se visita } i \text{ al generar } O} \\
&= \frac{\sum_{t=0}^T P(x_t = i, x_{t+1} = j, O | \lambda)}{\sum_{t=0}^{T+1} P(x_t = i, O | \lambda)}, \\
\bar{b}_j(k) &= \frac{\text{número esperado de veces que se observa el símbolo } k \text{ en } j \text{ al generar } O}{\text{número esperado de veces que se visita } j \text{ al generar } O} \\
&= \frac{\sum_{t=0}^{T+1} P(x_t = j, o_t = k, O | \lambda)}{\sum_{t=0}^{T+1} P(x_t = j, O | \lambda)}.
\end{aligned}$$

Observa que

$$\begin{aligned}
P(x_t = i, O | \lambda) &= \alpha_i(t) \cdot \beta_i(t), \\
P(x_t = i, x_{t+1} = j, O | \lambda) &= \alpha_i(t) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_j(t+1), \\
P(x_t = j, o_t = k | \lambda) &= \alpha_j(t) \cdot \beta_j(t) \cdot \delta(o_t, k), \\
P(O | \lambda) &= \alpha_N(T),
\end{aligned}$$

donde $\delta(o_t, k)$ es 1 si $o_t = k$ y 0 en caso contrario.

Tenemos que, para $1 < i, j < N$,

$$\bar{a}_{ij} = \frac{\sum_{t=0}^{T-1} \alpha_i(t) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_j(t+1)}{\sum_{t=0}^T \alpha_i(t) \cdot \beta_i(t)}$$

Y en los estados inicial ($i = 1$) y final ($j = N$):

$$\begin{aligned}\bar{a}_{1j} &= \frac{\alpha_j(1)\beta_j(1)}{\alpha_N(T+1)} \\ \bar{a}_{iN} &= \frac{\alpha_i(T)\beta_i(T)}{\sum_{t=0}^T \alpha_i(t)\beta_i(t)}\end{aligned}$$

El número esperado de veces que se observa el símbolo k en el estado j es

$$\sum_{t=0}^{T+1} P(x_t = j, o_t = k, O \mid \lambda) = \sum_{t=0}^{T+1} P(x_t = j, O \mid \lambda) \cdot \delta(o_t, k).$$

Así pues,

$$\bar{b}_j(k) = \frac{\sum_{t=0}^{T+1} \alpha_j(t) \cdot \beta_j(t) \cdot \delta(o_t, k)}{\sum_{t=0}^T \alpha_j(t) \cdot \beta_j(t)}$$

Si tenemos un modelo $\lambda = (a, b)$ y estimamos con él el modelo $\bar{\lambda} = (\bar{A}, \bar{B})$, $\bar{\lambda}$ satisface

$$P(O \mid \lambda) \leq P(O \mid \bar{\lambda}).$$

El procedimiento iterativo de entrenamiento: el algoritmo Baum-Welch

1. Se parte de λ inicial arbitrario.
2. Calcular $\bar{\lambda}$ a partir de λ .
3. Si $P(O | \lambda) = P(O | \bar{\lambda})$ (o está “suficientemente” próximo), terminar; si no, hacer que λ sea igual a $\bar{\lambda}$ y volver al paso 1.

Una cuestión de implementación: el escalado Hay un problema de desbordamiento de precisión (*underflow*) al efectuar muchos productos de valores entre 0 y 1. Es necesario efectuar algún tipo de escalado.

Se puede escalar $\alpha_i(t)$ con un valor q cualquiera independiente de i y $\beta_i(t)$ con otro valor r cualquiera.

Ambos factores se cancelan al efectuar operaciones como ésta:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^T q \cdot \alpha_i(t) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_j(t+1) \cdot r}{\sum_{t=1}^T q \cdot \alpha_t(i) \cdot \beta_t(i) \cdot r}$$

Inicialización del entrenamiento La calidad de la solución hallada por el método de Baum-Welch depende del modelo inicial. ¿Cómo podemos encontrar un buen modelo inicial?

1. ¿Cómo estimamos A ?

La experiencia indica que una inicialización uniforme de A proporciona buenos resultados.

2. ¿Cómo estimamos B ?

Hemos de tener una buena estimación inicial de $b_j(k)$, para $1 \leq j \leq N$ y $1 \leq k \leq M$.

Podemos segmentar la entrada en tantas partes como estados tiene el modelo y asignar inicialmente cada segmento a un estado. La segmentación puede ser:

- En partes iguales.
- Manualmente.
- Con un procedimiento automático como el “segmental k -means” con agrupamiento.

Entrenamiento con conjuntos de secuencias Hemos desarrollado el método suponiendo que sólo hay una secuencia de observaciones O .

Una buena estimación requiere un conjunto grande de secuencias de observaciones $\mathcal{O} = \{O^1, O^2, \dots, O^S\}$.

Queremos maximizar ahora

$$P(\mathcal{O} | \lambda) = \prod_{s=1}^S P(O^s | \lambda).$$

Basta con reestimar los parámetros considerando todas las secuencias de observaciones en cada iteración.

$$\begin{aligned}\bar{a}_{ij} &= \frac{\sum_{s=1}^S P(O^s | \lambda) \sum_{t=1}^{T_s} \alpha_i^s(t) a_{ij} b_j(o_{t+1}^k) \beta_j^s(t+1)}{\sum_{s=1}^S P(O^s | \lambda) \sum_{t=1}^{T_s} \alpha_i^s(t) \beta_i^s(t)}, \\ \bar{b}_j(k) &= \frac{\sum_{s=1}^S P(O^s | \lambda) \sum_{t=1}^{T_s-1} \alpha_i^s(t) \beta_i^s(t) \delta(o_t, k)}{\sum_{s=1}^S P(O^s | \lambda) \sum_{t=1}^{T_s} \alpha_i^s(t) \beta_i^s(t)}.\end{aligned}$$

El problema de la falta de datos

Al estimar $b_j(k)$ es posible que no se observe nunca el símbolo k en el estado j y se infiera que $b_j(k) = 0$.

El modelo resultante asigna probabilidad 0 a cualquier secuencia que suponga la emisión del símbolo k en j .

Soluciones:

- Conseguir **más datos** para entrenamiento.
Difícil y/o costoso.
- **Reducir el número de estados** del modelo.
Puede haber razones físicas que desaconsejen eliminar estados.
- **Suavizar la estimación** fijando umbrales de probabilidad mínima y reescalando las probabilidades para satisfacer las restricciones estocásticas.

$$\hat{b}_j(k) = \begin{cases} s \cdot b_j(k), & \text{si } b_j(k) \geq \nu; \\ \nu, & \text{en otro caso.} \end{cases}$$

El valor de s se fija para que $\sum_{k=1}^M b_j(k) = 1$.

Un problema similar puede surgir con la estimación de los a_{ij} : si uno de ellos llega a 0, ya no puede tomar un valor distinto por el procedimiento de reestimación estudiado. Es preciso fijar algún umbral mínimo.

Modelos continuos

Hemos supuesto que los Modelos de Markov emiten símbolos de un alfabeto discreto y finito.

Los eventos acústicos resultantes de la parametrización son, normalmente, vectores de características:

- cepstrales,
- derivadas de cepstrales,
- ...

¿Cómo tratar el caso en el que el alfabeto es continuo?

Es necesario redefinir la probabilidad de emitir un “símbolo” desde un estado, es decir, b_i , para $1 \leq i \leq N$.

Alfabeto continuo y escalar Imaginemos que cada observación es una medida de la energía de la señal. Podemos suponer que, en cada estado i , la energía observada sigue una distribución normal:

$$b_j(o_t) = \mathcal{N}(o_t; \mu_j, \sigma_j) = \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{o_t - \mu_j}{\sigma_j} \right)^2}.$$

La distribución del estado i quedaría caracterizada por dos parámetros:

- la media μ_j .
- y la desviación típica σ_j .

La estimación por Baum-Welch de los parámetros $\bar{\mu}_j$ y $\bar{\sigma}_j$ de b_j se hace con la fórmula

$$\begin{aligned}\bar{\mu}_j &= \frac{\sum_{t=1}^T P(x_t = j, O | \lambda) \cdot o_t}{\sum_{t=1}^T P(x_t = j, O | \lambda)}, \\ \bar{\sigma}_j &= \frac{\sum_{t=1}^T P(x_t = j, O | \lambda) \cdot (o_t - \mu_j)^2}{\sum_{t=1}^T P(x_t = j, O | \lambda)}.\end{aligned}$$

Alfabeto continuo y vectorial Al parametrizar la voz vimos que obtenemos un vector de características (valores reales) en cada instante de tiempo.

Nuestra secuencia de observaciones es una secuencia de vectores:

$$\mathbf{O} = \mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_T.$$

Supondremos que los vectores son de dimensión n .

Distribución normal n -dimensional:

$$b_j(\mathbf{o}_t) = \mathcal{N}(\mathbf{o}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}_j|}} e^{-\frac{1}{2}(\mathbf{o}_j - \boldsymbol{\mu}_j)' \boldsymbol{\Sigma}_j^{-1} (\mathbf{o}_j - \boldsymbol{\mu}_j)}.$$

Podemos trabajar con una distribución normal multidimensional en cada estado. La distribución del estado j quedaría caracterizada por

- la media $\boldsymbol{\mu}_j$, un vector de n parámetros,
- y la matriz de covarianzas $\boldsymbol{\Sigma}_j$, una matriz de n^2 parámetros.

La estimación por Baum-Welch de los parámetros $\bar{\mu}_j$ y $\bar{\sigma}_j$ de b_j se hace con la fórmula

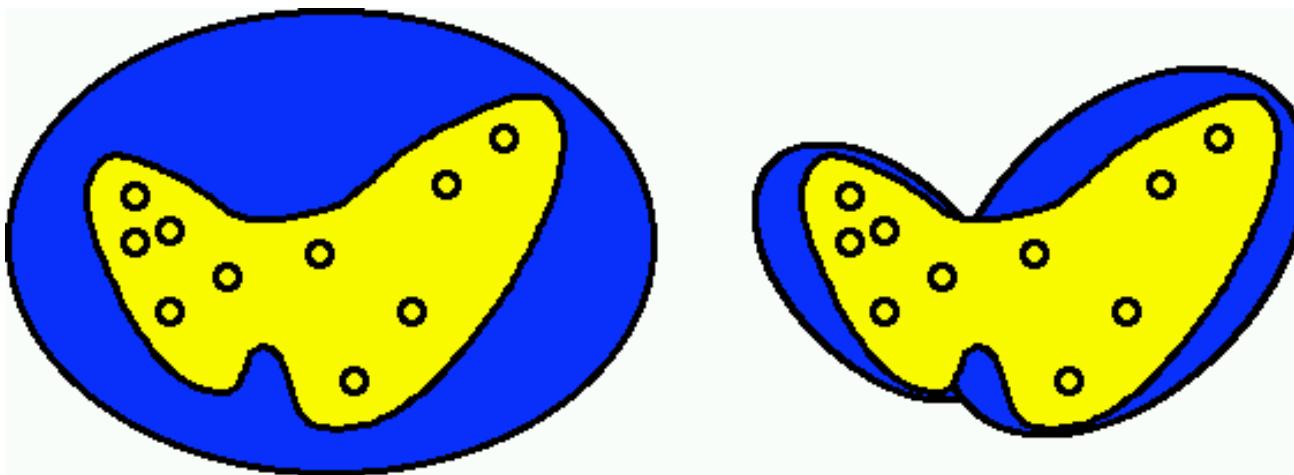
$$\begin{aligned}\bar{\mu}_j &= \frac{\sum_{t=1}^T P(x_t = j, O | \lambda) \cdot \mathbf{o}_t}{\sum_{t=1}^T P(x_t = j, O | \lambda)}, \\ \bar{\Sigma}_j &= \frac{\sum_{t=1}^T P(x_t = j, O | \lambda) \cdot (\mathbf{o}_t - \bar{\mu}_j)(\mathbf{o}_t - \bar{\mu}_j)'}{\sum_{t=1}^T P(x_t = j, O | \lambda)}.\end{aligned}$$

Estimar las N matrices de covarianzas Σ_j es, en la práctica, imposible (excesivo número de parámetros). Se suele asumir que las matrices de covarianzas son **diagonales**.

La asunción es razonable si los parámetros no están muy correlacionados, algo que hemos procurado al calcular el cepstrum del banco de filtros o de la LPC.

Mixturas de Gaussianas La asunción de que las observaciones emitidas en un estado siguen una distribución normal deja de ser razonable cuando se comprueba que son multimodales.

Una técnica que permite tratar con distribuciones multimodales es el modelado con mixturas de Gaussianas. Una mixtura es una combinación lineal de G distribuciones normales (Gaussianas).



Distribución bimodal modelada con una Gaussiana (izquierda) y con una mixtura de dos Gaussianas (derecha).

La distribución del estado j quedaría caracterizada por

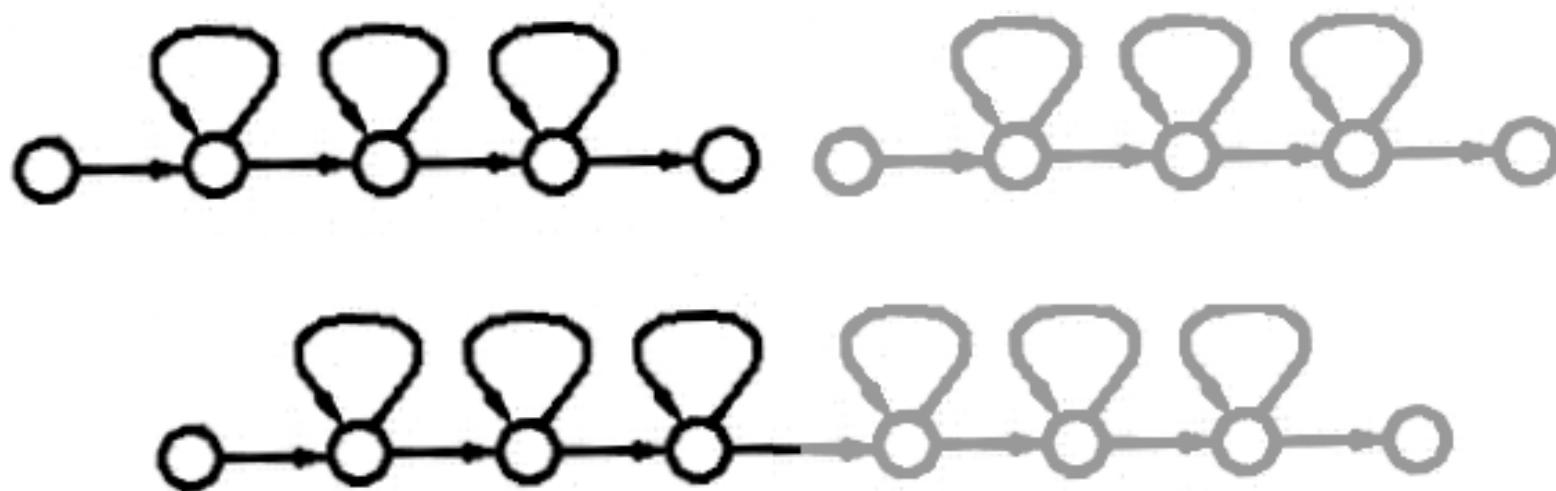
- un vector de G pesos $(c_{j1}, c_{j2}, \dots, c_{jG})$,
- las medias μ_{jg} para $1 \leq g \leq G$, que son G vectores de n parámetros,
- y las matrices de covarianzas Σ_{jg} para $1 \leq g \leq G$, que son G matrices de n^2 parámetros (o sólo n , si las matrices son diagonales).

$$b_j(\mathbf{o}_t) = \sum_{g=1}^G c_{jg} \mathcal{N}(\mathbf{o}_t; \boldsymbol{\mu}_{jg}, \boldsymbol{\Sigma}_{jg}).$$

Usualmente se entrena inicialmente los modelos con una sola Gaussiana. Una última fase convierte las Gaussianas simples en mixturas de Gaussianas por un proceso de partición de mixturas (*mixture splitting*). El número de mixturas para cada estado se determina empíricamente.

Composición de HMMs

Los HMMs pueden componerse: podemos concatenar o alternar HMMs y obtener un nuevo HMM equivalente.



Disponer de estados iniciales y finales no emisores simplifica la composición de modelos.

Reconocimiento de palabras aisladas

Clasificación

Disponemos de un vocabulario con V palabras.

- Se construye un modelo λ_v para cada palabra del vocabulario.
- En principio, dada una pronunciación $O = o_1 o_2 \dots o_T$, el índice de la palabra que más probablemente ha sido pronunciada es

$$\arg \max_{1 \leq v \leq V} P(O | \lambda_v) = \arg \max_{1 \leq v \leq V} \alpha_{N_v}(T + 1).$$

valor que podemos calcular mediante el algoritmo forward.

- Pero en la práctica es frecuente proporcionar como resultado

$$\arg \max_{1 \leq v \leq V} \max_X P(O, X | \lambda_v) = \arg \max_{1 \leq v \leq V} \phi_N(T + 1) = \arg \max_{1 \leq v \leq V} \log \phi_N(T + 1)$$

es decir, la palabra que proporciona mayor puntuación de Viterbi que es, en la práctica, más rápido de calcular y presenta menos problemas de *underflow* al trabajar con logprobs.

Coste de la clasificación $O(VTN^2)$.

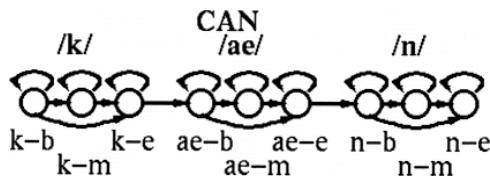
Construcción de modelos

- Posibilidad 1: un estado por “sonido elemental”. Topología lineal con posibles saltos/desviaciones en lugares con pronunciaciones alternativas.
Las palabras constan de entre 2 y 10 sonidos elementales, típicamente.
- Posibilidad 2: un estado por cada 10–15 ms de la duración máxima de la pronunciación de una palabra. Topología de Bakis.

¿Un modelo por palabra o un modelo por fonema?

Hemos desarrollado nuestro primer sistema basado en modelos de Markov asumiendo que cada modelo corresponde a una palabra. Es posible definir modelos para cada fonema y entrenarlos con las pronunciaciones de palabras, no de fonemas aislados. La ventaja estriba en que podemos obtener modelos mejor entrenados al entrenarse con todas las apariciones de un mismo fonema.

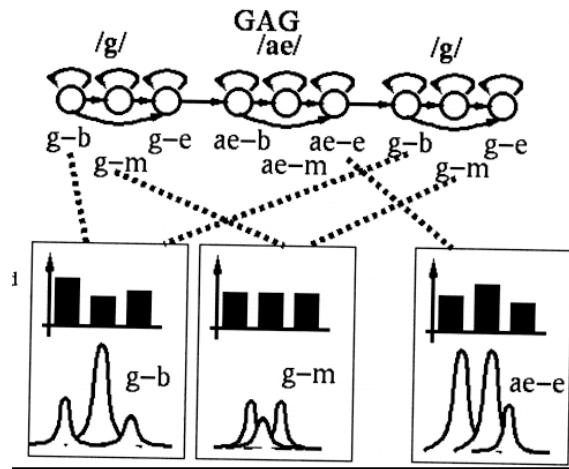
- Raramente se usan HMM discretos. Se usan mixturas de Gaussianas.
- Los HMM de fonemas suelen presentar pocos estados (típicamente 3 sin contar los estado inicial y final) y cada estado modela una parte del fonema (inicio, central, final).
- Los modelos de fonema pueden componerse para formar modelos de palabra.



HMM para la palabra “can”. Está formado por tres modelos de fonema. Cada modelo de fonema tiene tres estados: inicial (-b), medio (-m) y final (-e), cada uno de los cuales modela una porción del correspondiente fonema.

- Los estados que modelan un mismo fenómeno acústico pueden compartir el mismo modelo

acústico (ligaduras).



Por ligar se entiende compartir un conjunto de parámetros. Si dos estados están ligados, comparten la (parámetros de la) distribución de probabilidad de emisión de observaciones.

Por ejemplo, las explosivas /p, t, k/ comparten un tramo inicial de silencio muy similar. Si el primer estado emisor de varios de nuestros HMMs se destina a modelar ese silencio, es posible “ligar” las probabilidades de emisión de ese estado en todos los modelos.

- **Menos parámetros** que aprender...
- con lo que cada parámetro ligado “toca” a **más datos** para ser estimado.

Sobre la inicialización de los modelos

- Se puede utilizar una estimación muy grosera (unas medias y covarianzas idénticas para todos los estados calculadas con toda la voz disponible) para inicializar los modelos. No suele dar buenos resultados.
- Normalmente, es mejor utilizar una **segmentación manual** de la voz de entrenamiento (o de un subconjunto) de ésta en la que indicamos con qué tramas acústicas debe inicializarse cada modelo.

Obtener una segmentación manual es un proceso costoso y propenso a la comisión de errores.

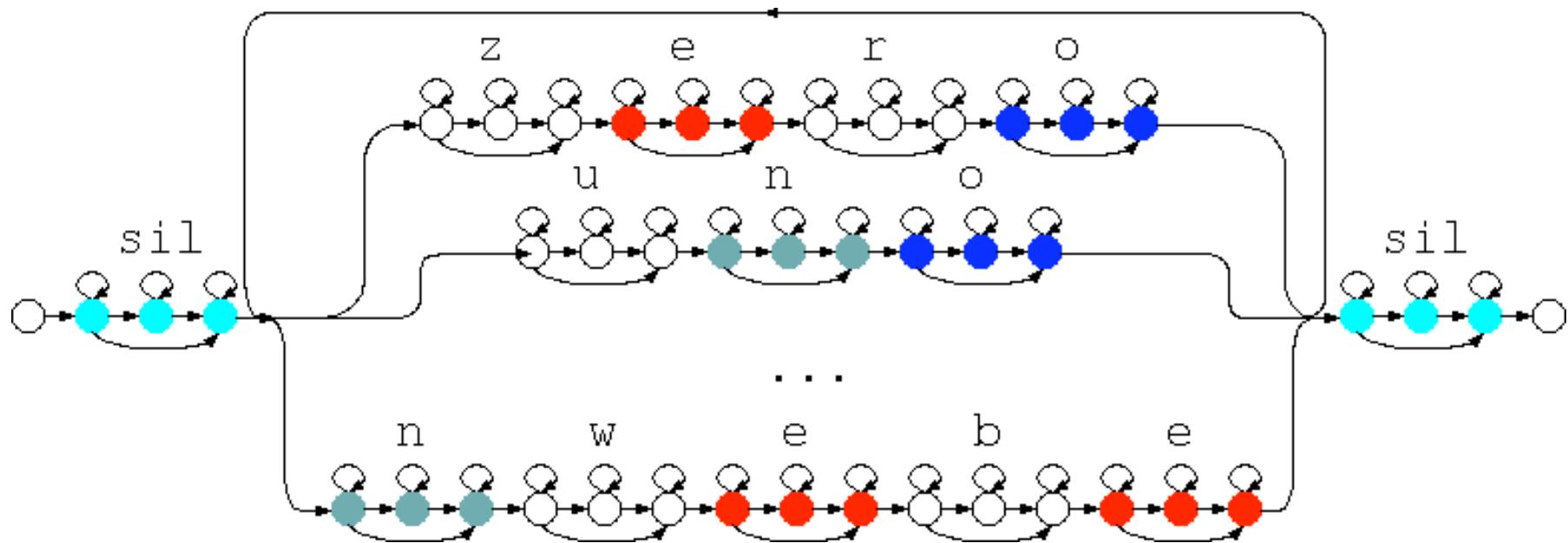
- Para evitar tener que segmentar manualmente, se puede utilizar la puntuación de Viterbi para determinar alineamientos óptimos entre voz y HMMs y, en consecuencia, encontrar segmentaciones óptimas (en cierto sentido).

Inicialmente, se considera que cada pronunciación se divide en partes iguales y se hace una primera estimación de los modelos bajo este supuesto. Entonces empieza un proceso iterativo de **segmentación/entrenamiento** hasta satisfacer algún criterio de convergencia.

Reconocimiento de palabras conectadas

Vamos a diseñar un sistema de reconocimiento de palabras conectadas como ejemplo. Trabajaremos nuevamente con dígitos, pero supondremos esta vez que éstos aparecen pronunciados en sucesión sin pausas entre ellos.

La idea es definir un modelo integrado como éste:

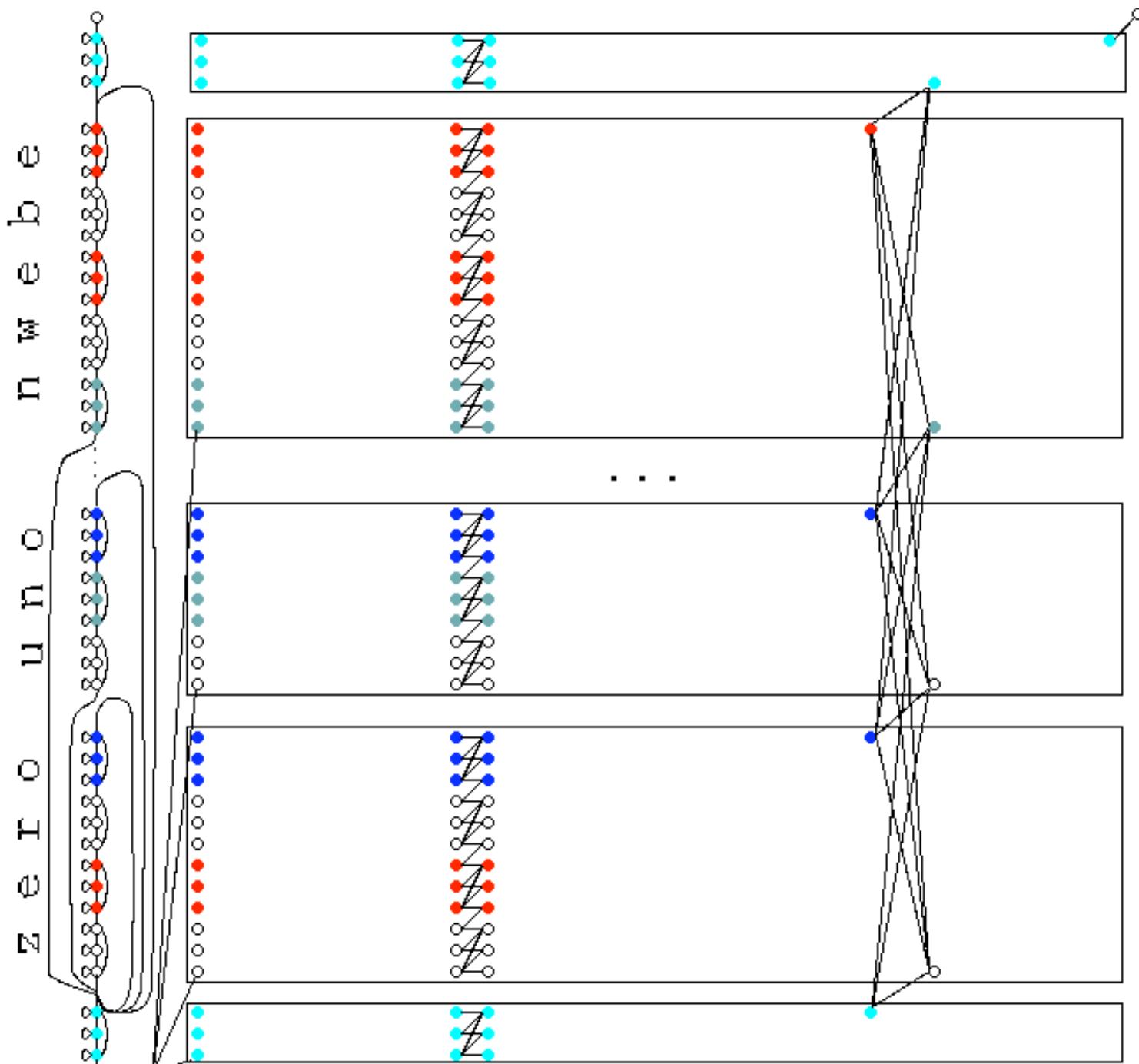


Nota: la figura muestra una versión simplificada al mostrar un único arco de salida desde el

estado final del último modelo de cada cifra. En realidad son once: el que va al modelo de “silencio” y los que van al inicio de cada cifra.

Dada una pronunciación, la secuencia de estados más probable en el modelo integrado contiene implícitamente la secuencia de palabras que más verosímilmente se ha pronunciado.

El Trellis que corresponde al modelo integrado para una pronunciación es éste:



El trellis modela un caso particular del problema de encontrar el camino óptimo (secuencia de estados) que visita n estados (el número de observaciones acústicas) en un grafo que es el modelo integrado.

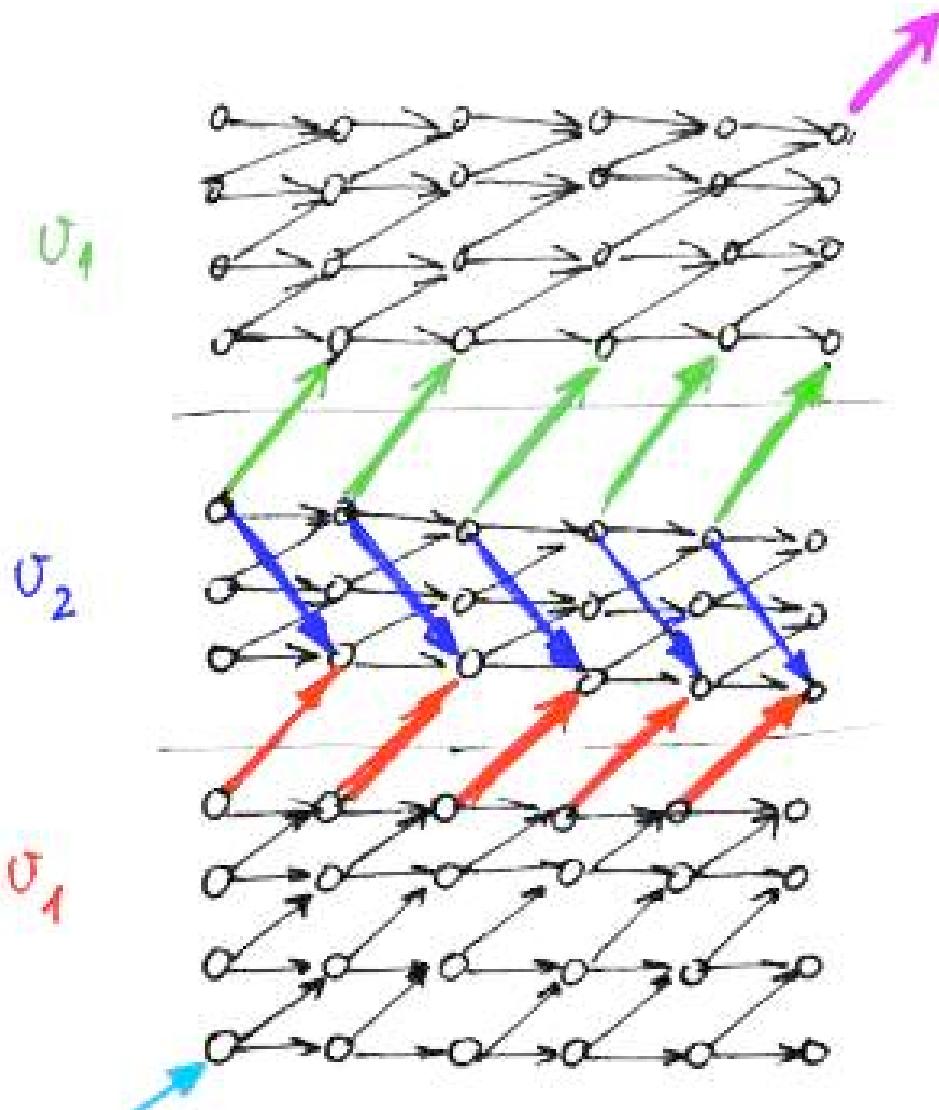
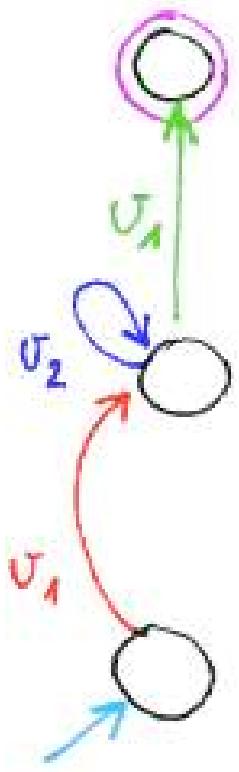
Estableciendo una analogía con el Trellis del alineamiento temporal, en éste se aprecian dos tipos distintos de transiciones (producciones): las transiciones intra-palabra y las transiciones inter-palabras.

Gramáticas regulares

El modelo de dígitos conectados de la anterior sección no es el único con el que podemos modelar pronunciaciones como secuencias de palabras.

Podemos definir Autómatas Finitos (AF) que indiquen cómo se unen las palabras para formar frases. El algoritmo de Un Paso es fácilmente extensible para trabajar con AF y puntuación de Viterbi (en cuyo caso se denomina “algoritmo de Viterbi”):

- en el eje vertical se dispone **un HMM por cada arco** (aunque dos o más correspondan a la misma palabra, se replican cuantas veces sea preciso),
- los **saltos inter-palabras** permiten saltar del final de un “arco” (palabra) al inicio de otro siempre que estén **unidos por un estado** del AF.



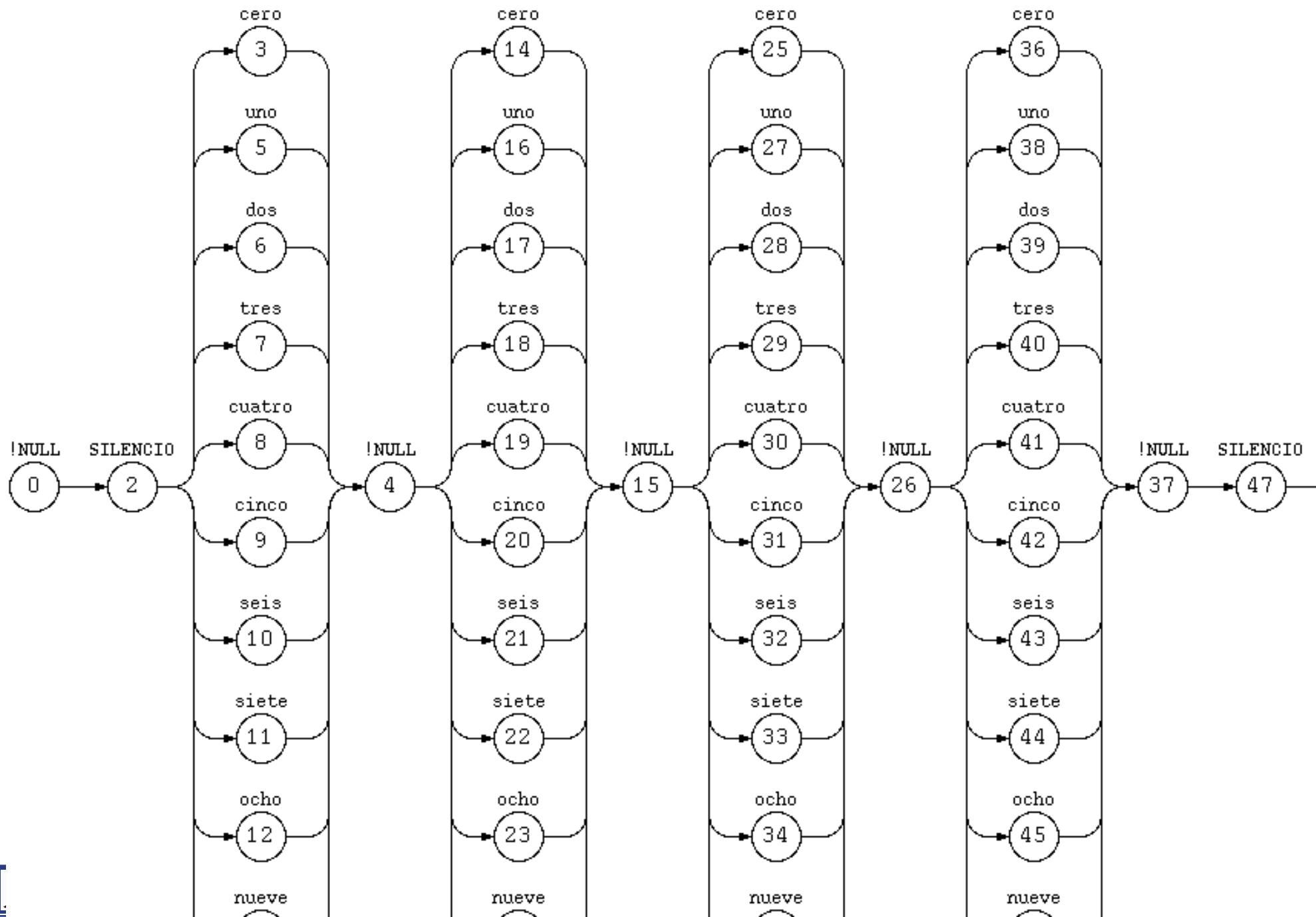
Trellis para un autómata.

Ejemplo: reconocimiento de extensiones telefónicas con 4 dígitos.

digitos4.gram

```
$numero = cero | uno | dos | tres | cuatro |  
         cinco | seis | siete | ocho | nueve ;  
(SILENCIO $numero $numero $numero $numero SILENCIO)
```

Podemos representar gráficamente el autómata así:



Hemos supuesto que los AF emiten “palabras” en los estados, y no en los arcos. Ciertos estados son no emisores (“emiten” la palabra especial !NULL). Estos AF son equivalentes a AF convencionales con λ -producciones.

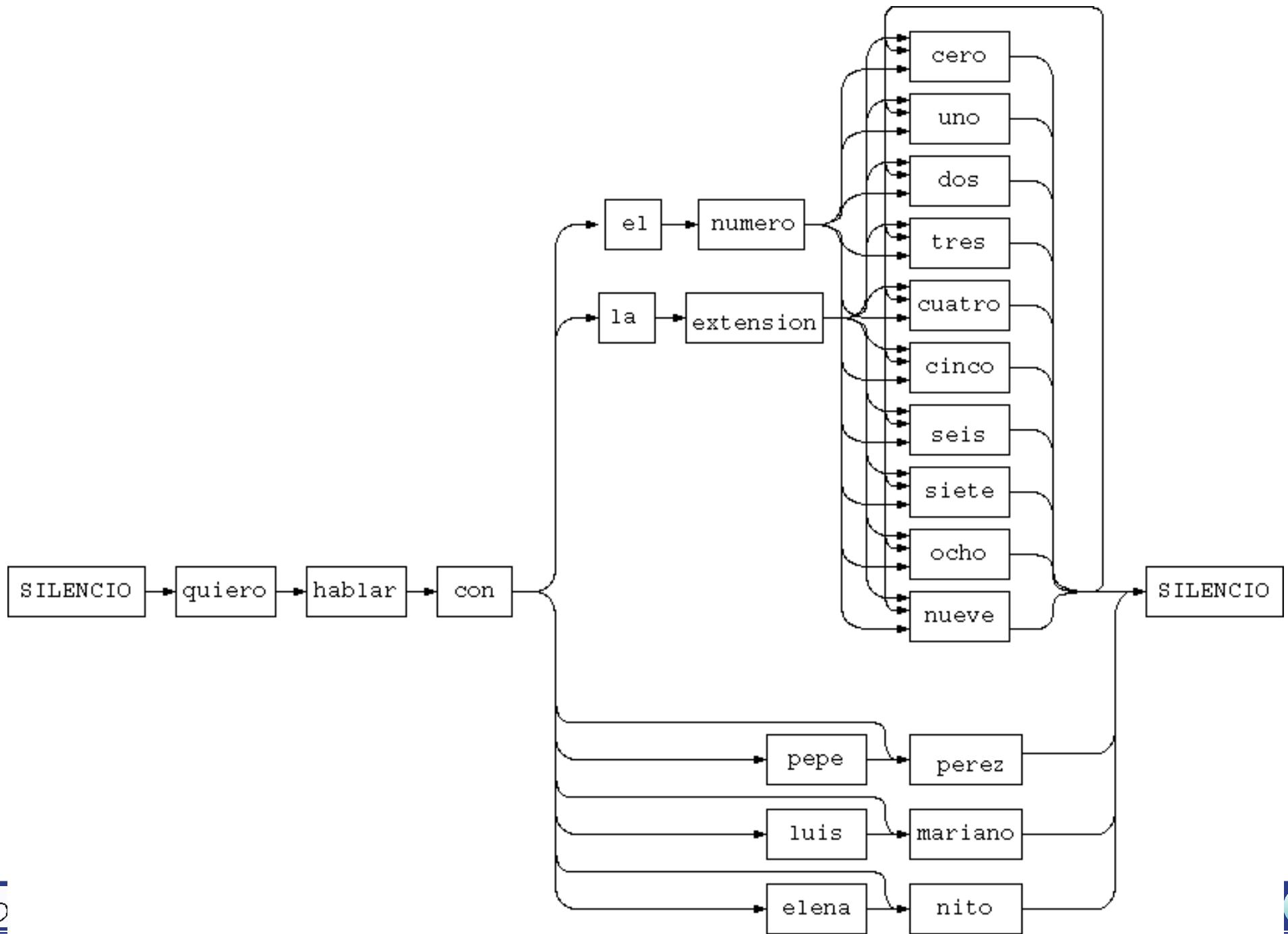
Un ejemplo:

Operadores regulares:

- |: alternativas.
- []: opcionalidad (0 o 1 vez).
- { }: clausura (0 o más veces).
- < >: clausura positiva (1 o más veces).

telefono.gram

```
$numero = cero | uno | dos | tres | cuatro |
          cinco | seis | siete | ocho | nueve ;
$nombre = [ pepe ] perez |
          [ luis ] mariano |
          [ elena ] nito ;
(SILENCIO
quiero hablar con ( $nombre |
          (el numero | la extension) < $numero >
SILENCIO )
```



Entrenamiento de modelos para palabras conectadas

Se puede efectuar el entrenamiento con palabras aisladas y el reconocimiento con palabras conectadas.

Es posible entrenar también con palabras conectadas. Es necesario disponer de un corpus de entrenamiento y sus transcripciones fonéticas (es lo habitual).

Al entrenar, se compone un modelo integrado para cada pronunciación que resulta de concatenar los modelos fonéticos presentes en la frase pronunciada.

Básicamente, es el mismo procedimiento que ya seguimos para entrenar a partir de palabras aisladas.

Un ejemplo completo

Vamos a desarrollar un reconocedor para una centralita telefónica. Para ello, puedes utilizar el toolkit Sphinx o el toolkit HTK.

Gramática de la tarea

Empezamos por definir una gramática para la tarea.

telefono.gram

```
$nombremasc = pepe | juan | josé | luis | víctor | andrés | guillermo | paulo |  
antonio | pablo | raul | germán | javier | david | miguel | carlos |  
mario | felipe | manuel | manolo | mariano | nacho | ignacio | jorge ;  
  
$nombrefem = maría | ana | luisa | isabel | paz | antonia | lidón | lledó ;  
  
$nombre = ( $nombremasc | $nombrefem ) ;  
  
$apellido = martínez | lópez | jiménez | marzal | vilá | peris | castellanos |  
aibar | prat | castaño | valls | amengual | montoliu | sanz | gómez | aliaga |  
fabregat | porcar | varó | pelayo | toledo | lobo | climent | ventura |  
garcía | ibáñez | palomar | lllorens | vilar | zarco | yagüe | llopis |  
espósito | badenas | monfort | granada | vizcaíno | iborra ;  
  
$persona = ( [ el señor ] ($nombremasc [ $apellido ] | [$nombremasc] $apellido ) |  
[ la señora ] ($nombrefem [ $apellido ] | [$nombrefem] $apellido ) ) ;  
  
$digito = cero | uno | dos | tres | cuatro | cinco | seis | siete | ocho | nueve ;
```

```
$numero = $digito [ $digito [ $digito [ $digito ] ] ] ;  
  
$extension = ( el número | la extensión | el | el teléfono ) $numero;  
  
$pedirhablar = ( deseo | desearía | me gustaría | quisiera | quiero | quería )  
                hablar ;  
  
$pedirpasar = póngame | me pone | me pasa | páseme | puede ponerme |  
               puede pasarme | me pasaría ;  
  
$pedir = $pedirhablar | $pedirpasar ;  
  
$pedirextension = $pedir con $extension;  
$pedirpersona = $pedir con $persona | se puede poner $persona |  
                $pedir con el secretario de $persona | $pedir con la secretaria de $persona |  
                $pedir con el jefe de $persona | $pedir con la jefa de $persona |  
                $pedir con el despacho de $persona | $pedirpasar con la extensión de $persona ;
```

```
$peticion = $pedirextension | $pedirpersona;
```

```
(SILENCIO ( por favor $peticion | $peticion [por favor] ) SILENCIO)
```

Diccionario de la tarea El diccionario contiene la secuencia de fonemas para cada palabra de la tarea. Podemos obtener dichas secuencias con `ort2fon.py`.

— telefono.dict —

SILENCIO [] sil
aibar a i b a r
aliaga a l j a g a
amengual a m e n g w a l
ana a n a
andrés a n d r e s
antonia a n t o n j a
antonio a n t o n j o
badenas b a d e n a s
carlos k a r l o s
castaño k a s t a h o
castellanos k a s t e H a n
cero z e r o

cinco z i n k o
climent k l i m e n t
con k o n
cuatro k w a t r o
david d a b i d
de d e
desearía d e s e a r i a
deseo d e s e o
despacho d e s p a c o
dos d o s
el e l
espósito e s p o s i t o
extensión e s t e n s j o n

fabregat f a b r e g
favor f a b o r
felipe f e l i p e
gómez g o m e z
garcía g a r z i a
germán x e r m a n
granada g r a n a d a
guillermo g i H e r m
gustaría g u s t a r
hablar a b l a r
ibáñez i b a h e z
iborra i b o @ a
ignacio i g n a z j o

isabel i s a b e l
javier x a b j e r
jefa x e f a
jefe x e f e
jiménez x i m e n e z
jorge x o r x e
josé x o s e
juan x w a n
lópez l o p e z
la l a
lidón l i d o n
lledó H e d o
llopis H o p i s
llorens H o r e n s
lobo l o b o
luis l w i s
luisa l w i s a
manolo m a n o l o
manuel m a n w e l

maría m a r i a
mariano m a r j a n o
mario m a r j o
martínez m a r t i n e z
marzal m a r z a l
me m e
miguel m i g e l
monfort m o n f o r t
montoliu m o n t o l j u
número n u m e r o
nacho n a c o
nueve n w e b e
ocho o c o
páseme p a s e m e
póngame p o n g a m e
pablo p a b l o
palomar p a l o m a r
pasa p a s a
pasaría p a s a r i a

pasarme p a s a r m e
paulo p a u l o
paz p a z
pelayo p e l a y o
pepe p e p e
peris p e r i s
pone p o n e
poner p o n e r
ponerme p o n e r m e
por p o r
porcar p o r k a r
prat p r a t
puede p w e d e
quería k e r i a
quiero k j e r o
quisiera k i s j e r
raul @ a u l
sanz s a n z
se s e

señor s e h o r

señora s e h o r a

secretaria s e k r e t a r

secretario s e k r e t a r

seis s e i s

siete s j e t e

teléfono t e l e f o n o

toledo t o l e d o

tres t r e s

juno u n o

ívíctor b i k t o r

valls b a H s

varó b a r o

ventura b e n t u r a

vilá b i l a

vilar b i l a r

vizcaíno b i z k a i n o

yagüe y a g w e

zarco z a r k o

Sesión de adquisición de voz para entrenamiento Ahora vamos a crear un fichero con frases de la tarea para entrenar los modelos. Podemos escribir las frases a mano o usar una herramienta para generar las frases a partir de la gramática.

Por ejemplo, el fichero **frases** contiene 150 frases.

frases

1. desearía hablar con sanz por favor
2. desearía hablar con jiménez
3. quiero hablar con el secretario de nacho porcar
4. póngame con vilá por favor
5. quisiera hablar con espósito
- ...
150. póngame con valls

Nota: las frases obtenidas se han generado siguiendo una distribución de probabilidad asociada al AF subyacente (que es, en el fondo, un HMM). Es posible que no se generen las frases como tú esperas. Puede que se generen muchas más peticiones para hablar con personas (hay 8 “ramas” que piden hablar con personas) que para conectar con una extensión numérica (sólo hay una rama dedicada a ello). Puedes, hasta cierto punto, controlar y corregir este problema replicando producciones. Por ejemplo, la producción

```
$peticion = $pedirextension | $pedirpersona ;
```

puedes definirla como

```
$peticion = $pedirextension |  
          $pedirpersona ;
```

Ahora debes adquirir las frases para utilizar en entrenamiento.

Sesión de adquisición de voz para evaluación Hemos de repetir el proceso anterior para adquirir, digamos, 50 nuevas frases con las que efectuar el proceso de evaluación.

Parametrización

Es habitual trabajar con $13 \times 3 = 39$ parámetros:

- 12 cepstrales más energía,
- velocidad,
- aceleración.

Preparar la sesión de entrenamiento

Los pasos a seguir son:

- Crear un fichero con la transcripción de cada una de las frases, tanto para las de entrenamiento como para las de test. Por ejemplo, `entrenamiento.mlf`

```
"test001.txt"
```

```
me
```

```
pone
```

```
con
```

```
el
```

```
despacho
```

```
de
```

```
antonia
```

```
aliaga
```

```
por
```

```
favor
```

```
.
```

```
...
```

- Ahora creamos una lista con las unidades básicas: los fonemas.

@
H
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p

r
s
t
u
w
x
y
z
sil

- Ahora hemos de generar la transcripción fonética de cada frase de entrenamiento. Además, podemos insertar silencios iniciales y finales. Por ejemplo, transcripcion.mlf

```
"training001.txt"
```

```
sil
```

```
p
```

```
a
```

```
s
```

```
e
```

```
m
```

```
e
```

```
k
```

```
o
```

```
n
```

```
e
```

```
l
```

```
x
```

```
e
```

```
f
```

```
e
```

d
e
l
o
p
e
z
p
o
r
f
a
b
o
r
sil
. . . .

Definición del modelo prototípico Se puede definir un modelo lineal de 3 estados emisores.

Inicialización de las probabilidades de emisión Se deben inicializar todos los modelos.

Iteraciones de entrenamiento Ya podemos empezar a iterar.

Evaluación Para evaluar la calidad de los modelos entrenados has de efectuar reconocimiento y obtener estadísticas de aciertos.

Líneas de mejora

- Parametrizaciones alternativas.
- Introducir pausas cortas en la gramática en aquellos puntos en los que es habitual detenerse (conviene crear un modelo especial para ese tipo de silencios).
- Modelar las distribuciones de probabilidad de emisión con mixturas de Gaussianas.
- Utilizar HMMs de diferente estructura para según qué fonemas.
- Usar fonemas con contexto.
- Enriquecer el juego de fonemas (considerando, por ejemplo, el uso de vocales tónicas).
- Enriquecer los modelos de pronunciación de palabra con pronunciaciones múltiples cuando convenga.
- Considerar que cepstrales, velocidad y aceleración son flujos diferentes.
- Inicializar el entrenamiento con la ayuda de una segmentación manual de un subconjunto de los datos de entrenamiento.
- ...

Un trabajo para el curso Un trabajo del curso puede consistir en construir un sistema de reconocimiento de palabras conectadas para esta tarea. La máxima calificación está garantizada para aquel que consiga la mayor tasa de aciertos a nivel de palabra. En caso de que dos o más estudiantes obtengan tasas de error muy próximas (pongamos que con una diferencia igual o inferior al 1 %) se valorará la velocidad del sistema.

Aceleración del cálculo

El tiempo necesario para efectuar reconocimiento crece con el número de arcos del autómata con el que representamos la gramática de la aplicación y puede llegar a ser prohibitivo, aun para vocabularios de tamaño moderado (una misma palabra puede formar parte de dos o más arcos).

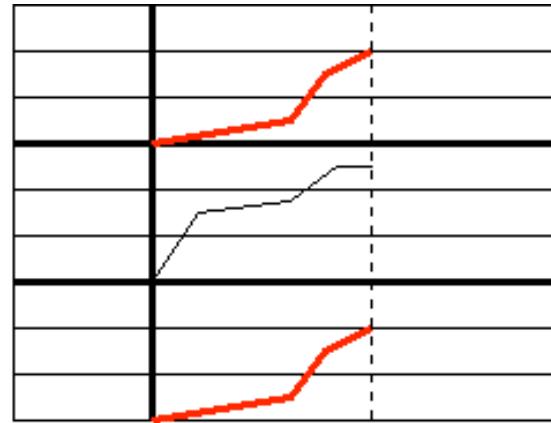
Es necesario recurrir a técnicas de aceleración del cálculo que lleguen incluso a sacrificar la corrección de la maximización, siempre que proporcionen una buena aproximación al resultado.

Árbol de prefijos

Muchos cálculos se efectúan más de una vez.

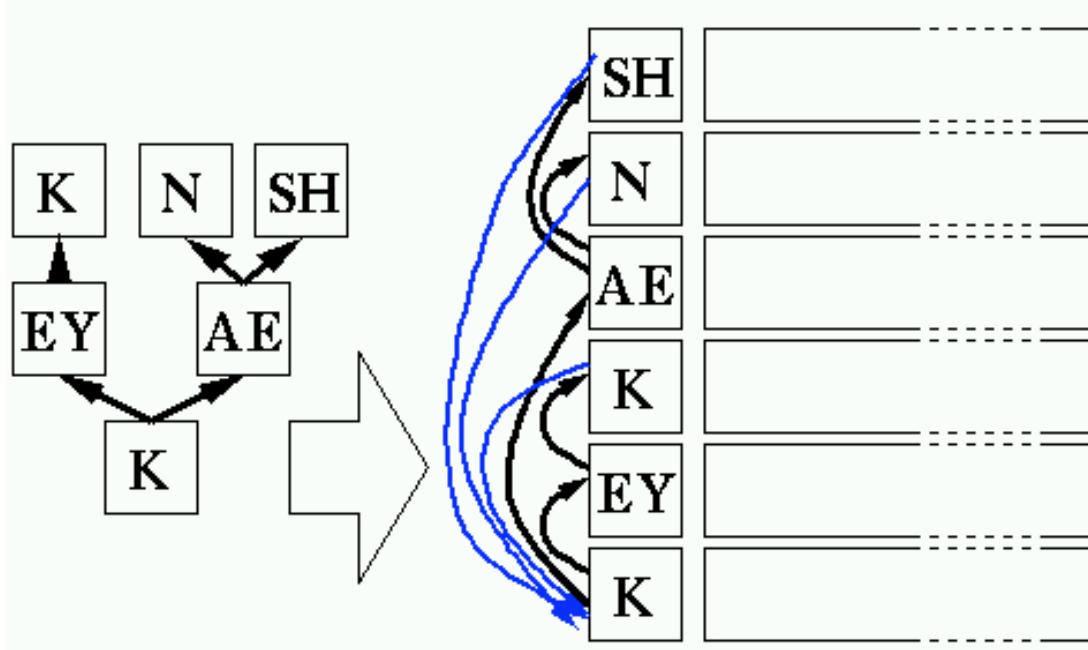
Ejemplo: “can” y “cash” empiezan con los mismos fonemas.

N	
AE	$p(AEl t)$
K	
K	
EY	
K	
SH	
AE	$p(AEl t)$
K	



La puntuación de Viterbi en los caminos rojos se calcula dos veces.

Se puede organizar el espacio de búsqueda para que los prefijos comunes se “fundan” con un árbol de prefijos:



Poda de estados

En el Trellis, cada estado visitado en el instante t da lugar a una serie de estados “candidato” en $t + 1$ de los que sólo sobreviven algunos.

Hay dos posibilidades:

- **Beam-Search:** Limitar los supervivientes en $t + 1$ a aquellos que caen dentro de cierto umbral del que mejor puntuación tiene en $t + 1$, por ejemplo, los que, para un factor B dado, tienen

$$\phi_j(t + 1) \geq b \cdot \max_{j'} \phi_{j'}(t + 1)$$

Inconvenientes:

- Puede dar lugar a un gran número de estados activos en zonas en las que las puntuaciones son similares.
- Es posible que no sobrevivan suficientes estados como para completar con éxito el análisis.
- Limitar los supervivientes a un número fijo de estados.

Inconveniente: requiere ordenar, para cada evento acústico, el conjunto de estados activos

para seleccionar los K mejores.

Otras técnicas

Existen otras técnicas de aceleración del cálculo.

- Técnicas multi-pasada.

Usan técnicas de exploración A^* para acelerar el cálculo. Es posible efectuar una pasada “backward” (o “forward”) inicialmente que sea rápida (sin gramática, por ejemplo), y usar la puntuación para obtener cotas que aceleren el cálculo al expandir estados.

- Técnicas de anticipación.

Técnicas que predicen la verosimilitud del fonema/palabra actual y lo eliminan por completo cuando conviene.

- Simplificación de cálculos de probabilidad de emisión.

Buena parte del tiempo de cálculo se dedica al cálculo de la probabilidad de emisión, especialmente cuando se trabaja con mixturas de Gaussianas.

Cuando modelamos b con mixturas de Gaussianas es posible que algunas aporten poca probabilidad a la emisión de un símbolo concreto, así que conviene disponer de técnicas que seleccionen rápidamente las Gaussianas que sí participan significativamente.

• ...

Bibliografía

- Lawrence Rabiner, Biing-Hwang Juang: *Fundamentals of speech recognition*. Prentice Hall. 1993.
- Steve Young et al.: *The HTK book (for HTK Version 3.1)*. Accesible en <http://htk.eng.ca>
- Frederick Jelinek: *Statistical Methods for Speech Recognition*. The MIT Press. 1998.
- Kai-Fu Lee: *Automatic speech recognition: The development of the Sphinx system*. Kluwer Academic. 1989.