第5章　隠れマルコフモデル（HMM）

隠れマルコフモデル（Hidden Markov Model: HMM）は，出力シンボルによって一意に状態遷移を決定することができない非決定性確率有限状態オートマトンとして定義されます．出力シンボル系列が与えられても状態遷移系列を一意に決めることができません，言い換えれば，観測できるのはシンボル系列だけであることから hidden（隠れ）マルコフモデルと呼ばれています．HMM は音声認識システムの音響モデル $P(\boldsymbol{O}|\boldsymbol{W})$ として用いられます．

5.1　基本 HMM の定式化

HMM は以下のパラメータで規定されます．

N モデルの状態数．一般に，状態は互いに接続されていて，任意の状態から（それ自身も含めて）他の任意の状態に遷移することができます．状態の集合を $\boldsymbol{S} \triangleq \{S_1, S_2, \ldots, S_N\}$ で表わすことにします．

M 出力シンボル種類数．モデル化する信号の離散記号を $\mathcal{V} \triangleq \{v_1, v_2, \ldots, v_M\}$ で表わします．

$\boldsymbol{A}$ 状態遷移確率 $\boldsymbol{A} = \{a_{ij}\}$．時刻 t での状態を q_t とするとき，

$$a_{ij} \triangleq P(q_{t+1} = S_j | q_t = S_i), \quad 1 \leq i, j \leq N, \quad \sum_{j=1}^{N} a_{ij} = 1.$$

$\boldsymbol{B}$ シンボル出力確率．状態 j における出力シンボルの確率分布 $\boldsymbol{B} = \{b_j(k)\}$．ここで，

$$b_j(k) \triangleq P(v_k | q_t = S_j) \quad 1 \leq j \leq N, \quad 1 \leq k \leq M.$$

$\boldsymbol{\pi}$ 初期状態確率 $\boldsymbol{\pi} = \{\pi_i\}$．

$$\pi_i \triangleq P(q_1 = S_j), \quad \sum_{i=1}^{N} \pi_i = 1, \quad 1 \leq i \leq N.$$

適当な N，M，$\boldsymbol{A}$，$\boldsymbol{B}$，および $\boldsymbol{\pi}$ が与えられれば，以下の手順を用いて，HMM を $\boldsymbol{O} = \boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T, \quad \boldsymbol{o}_t \in \mathcal{V}$ なる観測系列の生成器として用いることができます．

1. 初期状態確率分布 $\boldsymbol{\pi}$ に基づいて，初期状態 $q_1 = S_i$ を選ぶ．
2. $t = 1$ と設定する．

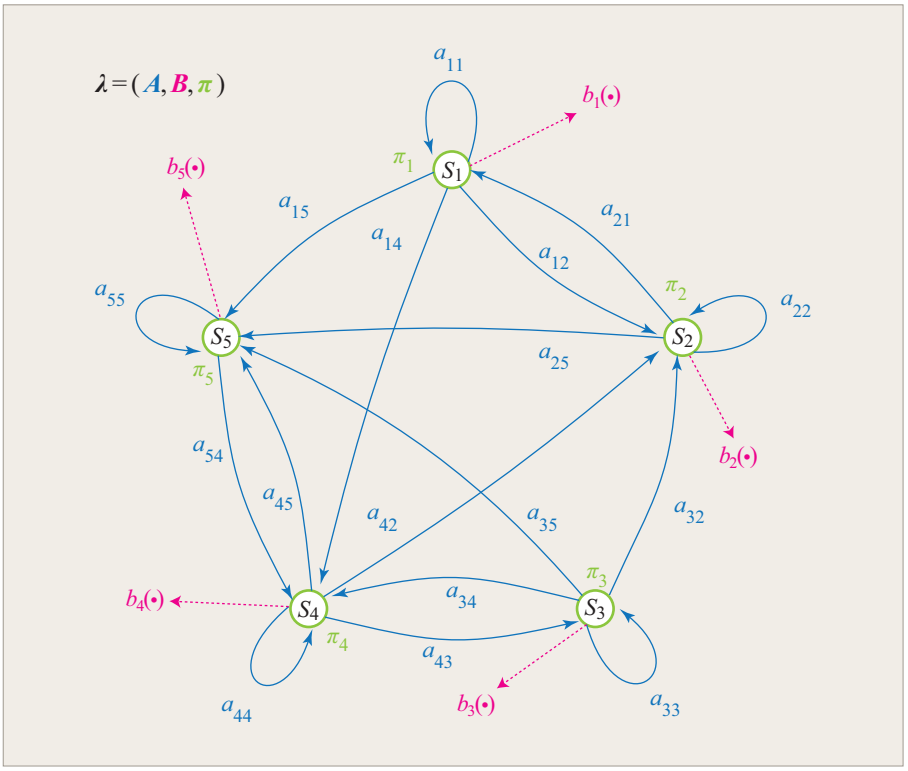

図 5.1: 5 状態の隠れマルコフモデル．確率 0 の状態遷移は描いていない．

3. 状態 S_i のシンボル出力確率分布 $b_i(k)$ にしたがって，$\boldsymbol{o}_t = v_k$ を選ぶ．

4. 状態 S_i の状態遷移確率 a_{ij} にしたがって，次の時刻の状態 $q_{t+1} = S_j$ に遷移する．

5. 時刻を 1 進める，すなわち，$t \leftarrow t+1$ とする．もし，$t < T$ ならばステップ 3 に戻り，さもなければ終了する．

以上により，HMM を規定するためには，N，M，観測系列の仕様，確率に関するパラメータ $\boldsymbol{A}$，$\boldsymbol{B}$，$\boldsymbol{\pi}$ が必要ですが．簡単のため，モデル λ のパラメータを

$$\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$$

と表わすことにします．

5.2　HMM の 3 つの問題

HMM を実際に用いるためには，次の 3 つの問題の解が必要です．

問題 1　観測系列 $\boldsymbol{O} = \boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T$ と，モデル $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ が与えられたとき，$P(\boldsymbol{O}|\lambda)$ を効率的に計算する．

問題 2　観測系列 $\boldsymbol{O} = \boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T$ と，モデル λ が与えられたとき，最適な状態遷移系列 $\boldsymbol{Q} = q_1 q_2 \cdots q_T$ を求める．

問題 3　$P(\boldsymbol{O}|\lambda)$ を最大化するモデル λ を求める．

問題 1 は，評価，つまり，モデルと観測系列が与えられたとき，その観測系列がそのモデルから生成される確率を計算する問題です．見方を変えれば，モデルが観測系列にどれだけ適合しているかの点数付けを行うことです．後者の観点で捉えると，複数のモデルの中から与えられた観測系列に最も適合するものを選ぶ（クラス分類する）ことができます．

問題 2 は，モデルの隠された部分（状態）に関することです．理論上，与えられた観測系列に対する”正しい”状態遷移系列というものを得ることはできません．しかし，適当な最適化基準を設定し，その下でもっともらしい状態系列を推定することは可能です．推定ができれば，モデルの構造を調べたり，音声のセグメンテーション[1]に応用することができます．

問題 3 は，与えられた観測系列が最もよく生成されるようにモデルのパラメータを最適化することです．このとき用いられる観測系列を**学習データ**といい，モデルパラメータを最適化することを **HMM を学習する**といいます．

以下，5.3 節では，まず，HMM が観測系列を生成する確率の計算の演算量の問題を考えます．5.3.1 節および 5.3.2 節では，問題 1 の解法である Forward アルゴリズムおよび Backward アルゴリズムを説明します．問題 2 の解法であるビタビ（Viterbi）アルゴリズムは 5.3.3 節で解説します．5.3.4 節で，問題 3 の解法である Baum-Welch アルゴリズムを説明します．

5.3 確率計算法

モデル λ が観測系列 $\boldsymbol{O} = \boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T$ を生成する確率を計算します．もっとも直接的な方法は，長さ T の全ての可能な状態遷移系列を枚挙し，各状態系列に関する確率を累積することです．その中の 1 つの状態系列を $\boldsymbol{Q} = q_1 q_2 \cdots q_T$ とします．状態系列 $\boldsymbol{Q}$ に対する観測系列 $\boldsymbol{O}$ の確率は，観測シンボルの独立性を仮定すると，

$$P(\boldsymbol{O}|\boldsymbol{Q}, \lambda) = \prod_{t=1}^{T} P(\boldsymbol{o}_t|q_t, \lambda)$$

と表わすことができます．HMM の定義より，

$$P(\boldsymbol{O}|\boldsymbol{Q}, \lambda) = b_{q_1}(\boldsymbol{o}_1) b_{q_2}(\boldsymbol{o}_2) \cdots b_{q_T}(\boldsymbol{o}_T) \tag{5.1}$$

となります．状態系列 $\boldsymbol{Q}$ の確率は，

$$P(\boldsymbol{Q}|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T} \tag{5.2}$$

となります．$\boldsymbol{O}$ と $\boldsymbol{Q}$ の同時確率は，式 5.1 と式 5.2 の積です．すなわち，

$$P(\boldsymbol{Q}, \boldsymbol{O}|\lambda) = P(\boldsymbol{O}|\boldsymbol{Q}, \lambda) P(\boldsymbol{Q}|\lambda)$$

[1] segmentation．連続音声を局所的にまとまりのある小単位に分割すること．小単位が属するカテゴリを認定することを含むのが一般的．

です．したがって，全ての可能な状態遷移系列 $\boldsymbol{Q} = q_1 \cdots q_T$ についてのこの同時確率の和は，

$$\begin{aligned} P(\boldsymbol{O}|\lambda) &= \sum_{\text{all } \boldsymbol{Q}} P(\boldsymbol{O}|\boldsymbol{Q},\lambda) P(\boldsymbol{Q}|\lambda) \\ &= \sum_{\text{all } \boldsymbol{Q}} \pi_{q_1} b_{q_1}(\boldsymbol{o}_1) a_{q_1 q_2} b_{q_2}(\boldsymbol{o}_2) \cdots a_{q_{T-1} q_T} b_{q_T}(\boldsymbol{o}_T) \end{aligned} \tag{5.3}$$

となります．

$P(\boldsymbol{O}|\lambda)$ の計算に要する演算量を見積もってみましょう．各時刻 t において到達可能な状態は N あるため，可能な状態系列は N^T 種類存在します．各状態系列の確率は $2T-1$ 回の乗算を行い，N^T 種類の確率の和を求めるために N^T-1 回の加算を要します．したがって，式 5.3 に必要な演算回数は，乗算が $N^T(2T-1)$ 回，加算が N^T-1 です．

実際に，演算回数がどれほどのものになるか計算してみます．状態数 $N=3$，観測系列長 $T=50$ の場合，乗算回数は $3^{50}\times(2\times50-1) = 7.17897988\times10^{23}\times99 \approx 7.1\times10^{25}$，加算は $3^{50}-1 = 7.17897988\times10^{23}-1 \approx 7.2\times10^{23}$ となります．この演算量は現実離れしています．音声認識で HMM を利用するためには，$P(\boldsymbol{O}|\lambda)$ の効率的な算法が必要です．

5.3.1　前向きパス（Forward）アルゴリズム

前向き（forward）変数 $\alpha_t(i)$ を定義します．

$$\alpha_t(i) \triangleq P(\boldsymbol{o}_1 \boldsymbol{o}_2 \cdots \boldsymbol{o}_t, q_t = S_i|\lambda). \tag{5.4}$$

$\alpha_t(i)$ はモデル λ が与えられたとき，時刻 t における状態が S_i である場合の部分観測系列 $\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_t$ の確率を表わしています．$\alpha_t(i)$ は帰納的に計算することが可能です．

1) 初期化

$$\alpha_1(i) = \pi_i b_i(\boldsymbol{o}_1), \quad 1 \le i \le N. \tag{5.5}$$

2) 帰納

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(\boldsymbol{o}_{t+1}), \quad 1 \le t \le T-1, \quad 1 \le j \le N. \tag{5.6}$$

3) 停止

$$P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i). \tag{5.7}$$

まず，前向き確率 $\alpha_1(i)$ を，初期状態確率 π_i と最初の観測シンボル $\boldsymbol{o}_1$ の同時確率として初期化します（式 5.5）．時刻 $t+1$ において状態 S_j には，時刻 t における N 個の状態 S_i, $1 \le i \le N$ から遷移が可能です（図 5.2（a））．前向き確率 $\alpha_t(i)$ は $\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_t$ が観測され，かつ状態が S_i にあるという同時事象の確率

なので，積 $\alpha_i(t)a_{ij}$ は $\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_t$ が観測され，かつ時刻 t での状態 S_i から時刻 $t+1$ で状態 S_j に遷移したという同時事象の確率ということになります．この積を時刻 t における可能な状態 $S_i,\ 1\le i\le N$ について足し合わせれば，時刻 t までの系列 $\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_t$ が観測されて時刻 $t+1$ に状態 S_j にある確率が求まります．この確率に状態 S_j においてシンボル $\boldsymbol{o}_{t+1}$ を観測する確率 $b_j(\boldsymbol{o}_{t+1})$ を掛けることにより，$\alpha_{t+1}(j)$ を得ます（式 5.6）．この計算は，時刻 t において全ての状態 $S_j,\ 1\le j\le N$ について行い，全ての $t=1,2,\ldots,T$ で繰り返します．そして，時刻 $t=T$ での前向き確率の和 $\sum_i^N \alpha_T(i)$ として所望の $P(\boldsymbol{O}|\lambda)$ が得られます．

$$\begin{aligned}\sum_i^N \alpha_T(i) &= \sum_i^N P(\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_T, q_T=S_i|\lambda)\\ &= P(\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_T|\lambda)\\ &= P(\boldsymbol{O}|\lambda)\end{aligned}$$

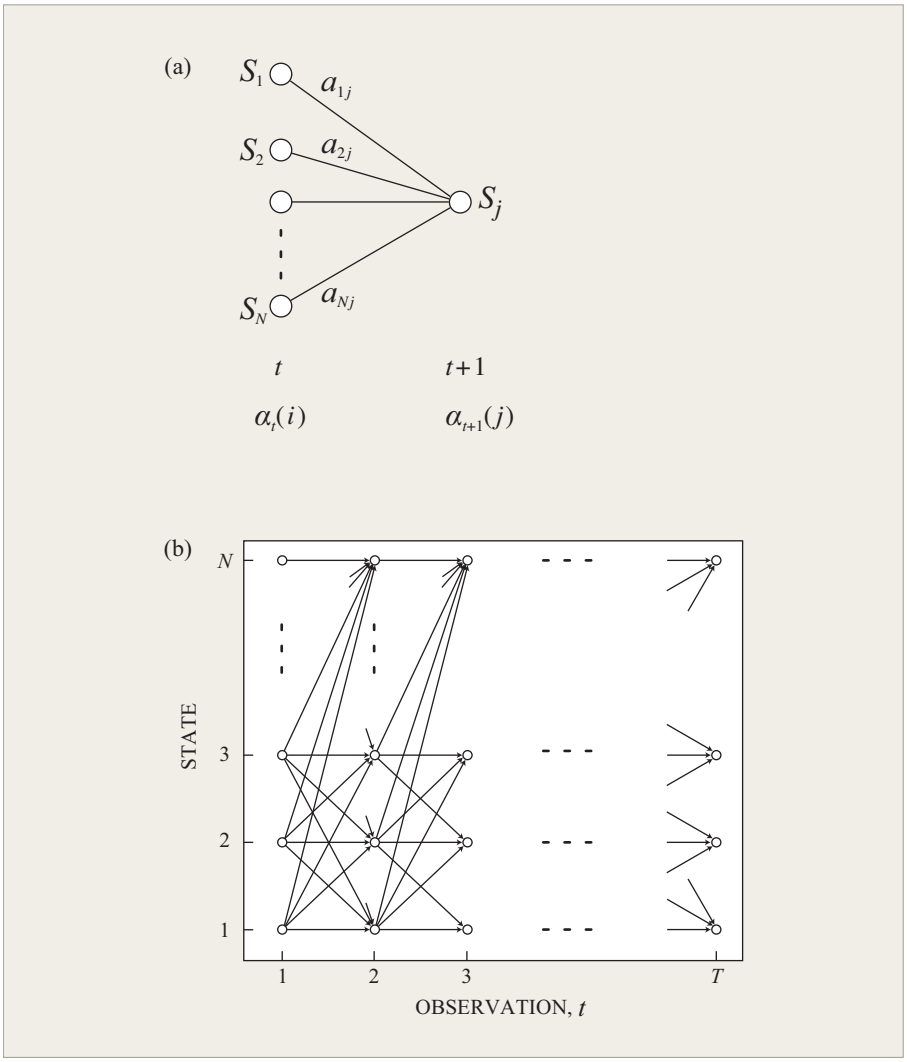

図 5.2: 前向きパス（Forward）アルゴリズム．(a) 帰納計算，(b) トレリス．

計算量の見積りは，乗算が $N(N+1)(T-1)+N$ 回，加算が $N(N-1)(T-1)$ 回となります．5.3 節と同じ例（$N=3,\ T=50$）を用いて演算量を計算して

みると，乗算回数は $3 \times (3+1) \times (50-1) + 3 = 3 \times 2 \times 49 = 591$，加算は $3 \times (3-1) \times (50-1) = 3 \times 2 \times 49 = 294$ となります．直接計算の場合と比べて，総演算量が 23 桁も削減されています！

前向き確率計算は，図 5.2（b）のような**トレリス（trellis，格子）**上で実行されていると考えると理解しやすいかもしれません．各時刻 t（横軸上の点）において N 個の状態しかないので，観測系列の長さに関係なく，その時刻までの可能な全ての状態系列が N 個のいずれかの状態（縦軸上の点）に合流します．時刻 $t=1$ において計算が必要なのは $\alpha_1(i),\ 1 \le i \le N$，時刻 $t=2,\ldots,T$ において計算が必要なのは $\alpha_t(j),\ 1 \le j \le N$ であり，それぞれは直前の N 個の値 $\alpha_{t-1}(i),\ 1 \le i \le N$ を用いれば求めることができます．このことが計算の大幅な効率化に役立っています．

5.3.2　後ろ向きパス（Backward）アルゴリズム

前向き変数と同様に，後ろ向き (backward) 変数 $\beta_t(i)$ を定義します．

$$\beta_t(i) \triangleq P(\boldsymbol{o}_{t+1}\boldsymbol{o}_{t+2}\cdots\boldsymbol{o}_T|q_t = S_i, \lambda).$$

1) 初期化

$$\beta_T(i) = 1, \quad 1 \le i \le N. \tag{5.8}$$

2) 帰納

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\boldsymbol{o}_{t+1})\beta_{t+1}(j), \quad t = T-1, T-2, \ldots, 1, \quad 1 \le i \le N. \tag{5.9}$$

3) 停止

$$P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(\boldsymbol{o}_1)\beta_1(i). \tag{5.10}$$

初期化では，時刻 T で全ての i について $\beta_T(i) = 1$ と設定します．帰納計算においては，時刻 t で状態 S_i にあり，かつ時刻 $t+1$ 以降の観測系列に対応する $\beta_t(i)$ を計算するため，時刻 $t+1$ で可能な全ての状態 S_j を考慮します（図 5.3）．すなわち，S_i から S_j への遷移確率 a_{ij}，S_j でシンボル $\boldsymbol{o}_{t+1}$ を観測する確率 $b_j(\boldsymbol{o}_{t+1})$，および状態 S_j 以降の後ろ向き変数値 $\beta_{t+1}(j)$ の積の $1 \le j \le N$ についての和を計算します．時刻 $t=1$ についての計算が済めば，アルゴリズムは終了します．Forward アルゴリズムと同様，Backward アルゴリズムの計算量は $O(N^2T)$ であり，トレリス上で計算が実行されます．

5.3.3　最適状態系列推定（Viterbi アルゴリズム）

観測系列 $\boldsymbol{O} = \boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_T$ に対する，モデル λ の最適状態遷移系列 $\boldsymbol{Q} = q_1q_2\cdots q_T$ を求める問題の解法として，ビタビ（Viterbi）アルゴリズムを説明し

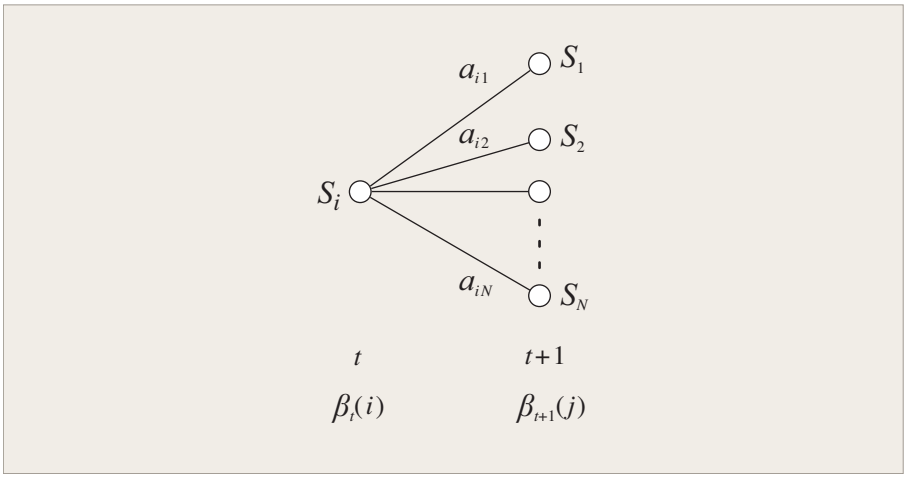

図 5.3: 後ろ向きパス（Backward）アルゴリズム

ます．まず，モデル λ について，観測系列 $\boldsymbol{O}$ が与えられたとき，時刻 t において状態 S_i にいる確率

$$\gamma_t(i) \triangleq P(q_t = S_i|\boldsymbol{O}, \lambda) \tag{5.11}$$

を定義します．前向き確率 $\alpha_t(i)$ は時刻 t において状態 S_i にあり部分観測系列 $\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_t$ を観測した確率，後ろ向き確率 $\beta_t(i)$ は時刻 t において状態 S_i にあり部分観測系列 $\boldsymbol{o}_{t+1}\boldsymbol{o}_{t+2}\cdots\boldsymbol{o}_T$ を観測する確率なので，式 5.11 は，$\alpha_t(i)$ と $\beta_t(i)$ を用いて書き表すことができます．すなわち，

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\boldsymbol{O}|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}.$$

分母の $P(\boldsymbol{O}|\lambda)$ は変数 $\gamma_t(i)$ を確率尺度にするためのものです．したがって，

$$\sum_{i=1}^{N}\gamma_t(i) = 1.$$

変数 $\gamma_t(i)$ を用いることにより，時刻 t においてもっともありうる状態 q_t を

$$q_t = \underset{1\le i\le N}{\operatorname{argmax}}\,\gamma_t(i), \quad 1 \le t \le T$$

と得ることができます．しかし，この基準で各時刻毎に求めた q_t を並べても，全体として妥当な状態系列 $\boldsymbol{Q} = q_1q_2\cdots q_T$ になるとは限りません．例えば，$a_{ij} = 0$ となる i と j が存在する場合には，実際にあり得ない状態系列になってしまう可能性があります．

このような可能性を避けるため，動的計画法（Dynamic Programming: DP）に基づいた Viterbi アルゴリズムが考案されました．与えられた観測系列 $\boldsymbol{O} = \boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_T$ に対して単一の最良状態系列 $\boldsymbol{Q} = q_1q_2\cdots q_T$ を求めるため，

$$\delta_t(i) \triangleq \max_{q_1q_2\cdots q_{t-1}} P(q_1q_2\cdots q_t = i, \boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_t|\lambda)$$

を定義します．変数 $\delta_t(i)$ は $t = 1,\ldots,t$ までのシンボルを観測して状態 S_i に至る状態系列のうち最も良い（最大の確率を与える）状態系列の確率です．$\delta_t(i)$ を

基にして，1 時刻先の δ_{t+1} は

$$\delta_{t+1}(i) = \left(\max_i \delta_t(i)a_{ij}\right) b_i(\boldsymbol{o}_{t+1}) \tag{5.12}$$

と計算することができます．最適状態系列を得るためには，各時刻 t の各状態 j おける式 5.12 を満たす状態遷移元 i を記録しておくことが必要です．この目的のために配列 $\psi_t(j)$ を用意します．以下に，Viterbi アルゴリズムの手順を示します．

1) 初期化

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(\boldsymbol{o}_1), \quad 1 \le i \le N. \\ \psi_1(i) &= 0. \end{aligned}$$

2) 帰納

$$\begin{aligned} \delta_t(j) &= \max_{1\le i\le N} \left(\delta_{t-1}(i)a_{ij}\right) b_j(\boldsymbol{o}_t), \quad 2 \le t \le T, \quad 1 \le j \le N. \\ \psi_t(j) &= \operatorname*{argmax}_{1\le i\le N} \left(\delta_{t-1}(i)a_{ij}\right), \quad 2 \le t \le T, \quad 1 \le j \le N. \end{aligned}$$

3) 停止

$$\begin{aligned} P^*(\boldsymbol{O}|\lambda) &= \max_{1\le i\le N} \delta_T(i). \\ q_T^* &= \operatorname*{argmax}_{1\le i\le N} \delta_T(i). \end{aligned} \tag{5.13}$$

4) 経路のバックトラッキング

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \ldots, 1.$$

Viterbi アルゴリズムは，Forward アルゴリズムと類似しています．Forward アルゴリズムと異なるのは，直前の状態に関する和（$\sum$）が最大値（max）に置き換わっている点です．Viterbi アルゴリズムも Forward アルゴリズムと同様，トレリス上で効率的に実行することができます．

5.3.4 パラメータ推定（Baum-Welch アルゴリズム）

観測系列 $\boldsymbol{O}$ を生成する確率が最大になるようなモデル λ のパラメータ $(\boldsymbol{A}, \boldsymbol{B}, \pi)$ を求める問題には，解析的な解が知られていません．このような問題に対しては，まず，決定すべきパラメータに適当な初期値を与え，繰り返し計算しながらパラメータの値を少しずつ修正し，最終的に局所最適解を得る手法が用いられます．Baum-Welch アルゴリズムはそのような手法の 1 つです．

まず，変数 $\xi_t(i,j)$ を定義します．これは，モデル λ と観測系列 O が与えられたとき，時刻 t で状態 S_i にあり，かつ時刻 $t+1$ で状態 S_j にある確率を表わすものとします．すなわち，

$$\xi_t(i,j) \triangleq P(q_t = S_i, q_{t+1} = S_j|\boldsymbol{O}, \lambda). \tag{5.14}$$

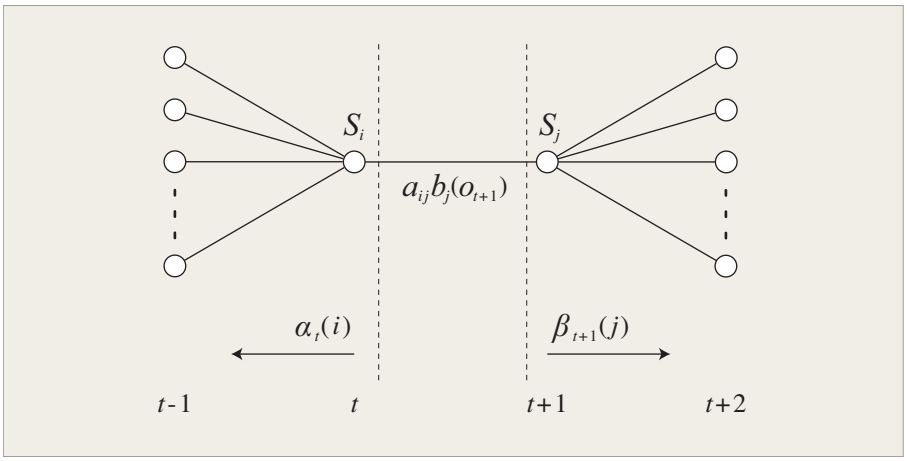

図 5.4: システムが時刻 t で状態 S_i にあり，かつ時刻 $t+1$ で状態 S_j にある同時確率を計算する．

式 5.14 の状況を図示すると，図 5.4 のようになります．前向き確率 $\alpha_t(i)$ と後ろ向き確率 $\beta_t(i)$ の定義より，$\xi_t(i,j)$ は

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(\boldsymbol{o}_{t+1})\beta_{t+1}(j)}{P(\boldsymbol{O}|\lambda)} \tag{5.15}$$

$$= \frac{\alpha_t(i)a_{ij}b_j(\boldsymbol{o}_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\boldsymbol{o}_{t+1})\beta_{t+1}(j)}. \tag{5.16}$$

と表わすことができます．ここで，分子は $P(q_t = S_i, q_{t+1} = S_j, \boldsymbol{O}|\lambda)$ であり，$P(\boldsymbol{O}|\lambda)$ で割ることにより，$P(q_t = S_i, q_{t+1} = S_j|\boldsymbol{O}, \lambda)$ を得ています．

5.3.3 節で定義した $\gamma_t(i)$ は，モデル λ について観測系列 $\boldsymbol{O}$ が与えられたとき，時刻 t において状態 S_i にいる確率なので，$\xi_t(i,j)$ の j についての和は $\gamma_t(i)$ と等しくなります．すなわち，

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j). \tag{5.17}$$

$\gamma_t(i)$ の t についての和をとれば，状態 S_i を通過する回数の期待値，別の見方をすれば状態 S_i からの状態遷移の回数の期待値が得られます．同様に，$\xi_t(i,j)$ の t についての和は，状態 S_i から S_j への遷移回数の期待値となります．すなわち，

$$\sum_{t=1}^{T-1}\gamma_t(i) = \text{状態 } S_i \text{ からの遷移回数の期待値}, \tag{5.18}$$

$$\sum_{t=1}^{T-1}\xi_t(i,j) = \text{状態 } S_i \text{ から } S_j \text{ への遷移回数の期待値}. \tag{5.19}$$

上記の式を基に，事象計数の概念を用いると，HMM のパラメータの再推定公

式を導くことができます．$\boldsymbol{A}$，$\boldsymbol{B}$，$\boldsymbol{\pi}$ の再推定公式は，

$$\tilde{\pi}_i = \text{時刻 } t=1 \text{ で状態 } S_i \text{ にいる頻度の期待値} = \gamma_1(i). \tag{5.20}$$

$$\tilde{a}_{ij} = \frac{\text{状態 } S_i \text{ から } S_j \text{ への遷移回数の期待値}}{\text{状態 } S_i \text{ からの遷移回数の期待値}} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)} \tag{5.21}$$

$$\begin{aligned}\tilde{b}_j(k) &= \frac{\text{状態 } S_j \text{ でシンボル } v_k \text{ を観測する期待値}}{\text{状態 } S_j \text{ にいる回数の期待値}} \\ &= \frac{\sum_{t=1,\ s.t.\ o_t=v_k}^{T-1}\gamma_t(j)}{\sum_{t=1}^{T-1}\gamma_t(j)}\end{aligned} \tag{5.22}$$

と導出されます．

現在のモデルを $\lambda=(\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})$ 用いて式 5.20，式 5.21，式 5.22 の右辺を計算し，左辺の値を更新されたモデル $\tilde{\lambda}=(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}},\tilde{\boldsymbol{\pi}})$ として用います．このようにして求めたモデル $\tilde{\lambda}$ はモデル λ に比べて良いモデル，すなわち，

$$P(\boldsymbol{O}|\tilde{\lambda}) > P(\boldsymbol{O}|\lambda) \tag{5.23}$$

となることが知られています．したがって，λ の代わりに $\tilde{\lambda}$ を用いて上記の手順を繰り返すことにより，観測系列 $\boldsymbol{O}$ が生成される確率は単調増加します．ある限界点に達するまで計算を繰り返すことにより，HMM パラメータの最尤推定値（の 1 つ）を得ることができます．ただし，極大値しか得られません．一般に最適値探索の空間は複雑で多くの極大値が存在するので，最大値に収束するとは限りません．

5.4　音声認識のための HMM

5.4.1　一方向性 HMM

取り扱う時系列パターンの性質に応じて，状態遷移に種々の拘束を設けることがあります．例えば，

$$a_{ij}=0, \quad j<i \tag{5.24}$$

という拘束を課した HMM を考えることができます．前の状態に戻ることが許されていないので，**一方向性モデル（Bakis モデル）**と呼ばれています．一方向性 HMM は，時間の経過とともに性質が変る音声信号のような信号をモデル化するのに適しています．状態系列は S_1 から始まり S_N で終了するので，式 5.24 に加えて，初期状態確率は，通常，

$$\pi_i = \begin{cases} 0 & i \neq 1 \\ 1 & i = 1 \end{cases} \tag{5.25}$$

に設定されます．

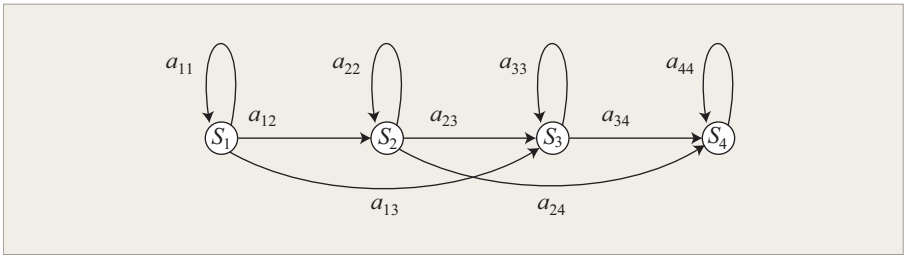

図 5.5: 一方向性 HMM．状態数は 4．2 つ先までの状態への遷移が可能．

大幅な状態の飛び越しが起きないように，状態遷移に拘束条件を加えることがあります．すなわち，

$$a_{ij} = 0, \quad j > i + \Delta \tag{5.26}$$

とします．Δ の値を 2 と設定すると図 5.5 のような 2 つより先の状態への遷移が許されない HMM になります．したがって，この HMM の状態遷移行列は，

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix} \tag{5.27}$$

となります．当然，一方向性モデルの最終状態については，

$$\begin{aligned} a_{NN} &= 1 \\ a_{Ni} &= 0, \quad i < N \end{aligned} \tag{5.28}$$

です．

一方向性 HMM を用いるときは，出力記号列 $\boldsymbol{O} = \boldsymbol{o}_1 \boldsymbol{o}_s \cdots \boldsymbol{o}_T$ が観測されたとき，時刻 T において最終状態に達していることが仮定されています．そうすると，$S_T = S_N$ という情報を用いることができるので，前向きアルゴリズム，後ろ向きアルゴリズム，Baum-Welch アルゴリズムの形が少し変ります．また，最終状態に達すると，その後の出力記号列は，他の状態における遷移確率や記号の出力確率に関する情報を全く含みません．したがって，パラメータ推定を行うときは，単一の長い観測系列パターンではなく，それぞれ初期状態から出発した HMM の出力をみなせる複数の時系列パターンが必要です．複数の学習パターンを用いてパラメータ推定を行う方法については，関連図書を参照してください．

5.4.2　連続 HMM

音声認識で音響モデル $P(\boldsymbol{O}|\lambda)$ として HMM を持ちいる場合，一般に，観測系列 $\boldsymbol{O}$ は連続値（実数ベクトル）です（図 5.6）．連続値を量子化して処理することも可能ですが，量子化処理によって情報が失われるため，性能がかなり低下する可能性があります．連続値の出力を扱う**連続 HMM** は，離散出力 HMM に比べて良い性能が期待できます．

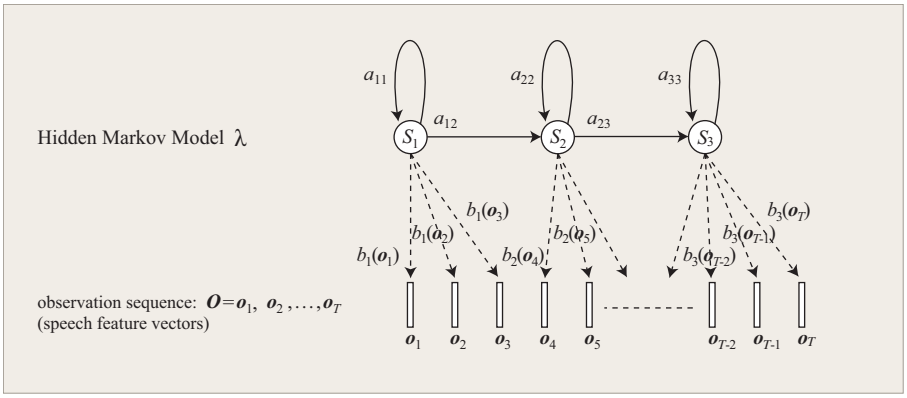

図 5.6: 音声スペクトル時系列 $\boldsymbol{O}$ を生成する HMM λ.

連続 HMM の出力を定義する確率密度関数としては，混合ガウス分布がよく用いられます．この場合，シンボル出力確率 $b_j(\boldsymbol{O})$ は，

$$b_j(\boldsymbol{o}_t) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\boldsymbol{o}_t, \mu_{jm}, \Sigma_{jm}), \quad 1 \leq j \leq N \tag{5.29}$$

と表わされます.. ここで，$\boldsymbol{o}_t$ は時刻 t の観測ベクトル，c_{jm} は状態 j の第 m 混合成分の混合係数，$\mathcal{N}$ はガウス分布，μ_{jm} と Σ_{jm} は状態 j の第 m 混合成分の平均値ベクトルと共分散行列です（図 5.4.2）．混合係数 c_{jm} は確率としての制約

$$\sum_{m=1}^{M} c_{jm} = 1, \quad 1 \leq j \leq N, \tag{5.30}$$

$$c_{jm} \geq 0, \quad 1 \leq j \leq N, 1 \leq m \leq M \tag{5.31}$$

を満たします．したがって，

$$\int_{-\infty}^{\infty} b_j(\boldsymbol{x}) d\boldsymbol{x} = 1, \quad 1 \leq j \leq N. \tag{5.32}$$

上記の混合ガウス分布は 1 次元ですが，実際の音声認識に用いるのは多次元の

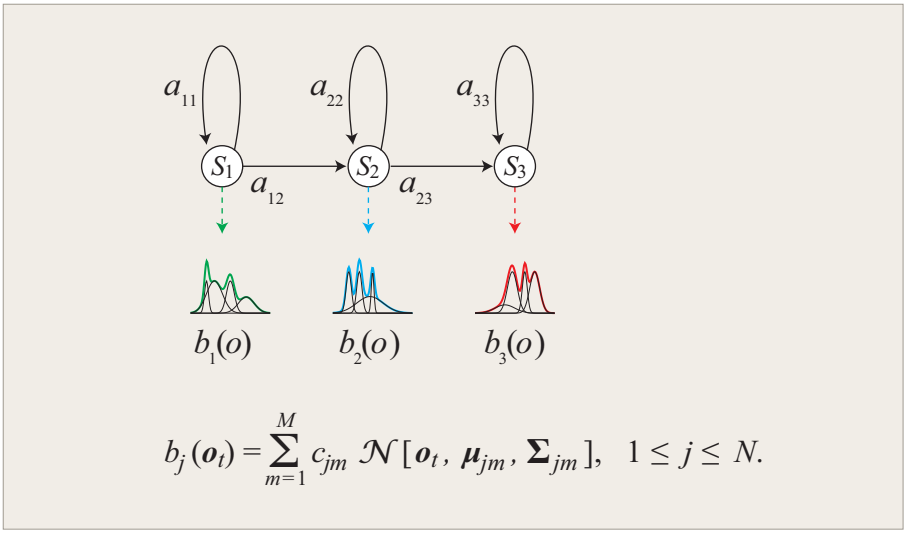

図 5.7: 出力確率密度が混合ガウス分布である連続 HMM

特徴量なので，多次元ガウス分布の確率密度関数を使います．

$$\mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_m|}} \exp\left\{-\frac{1}{2}(\boldsymbol{o}_t - \boldsymbol{\mu}_m)' \boldsymbol{\Sigma}_m^{-1} (\boldsymbol{o}_t - \boldsymbol{\mu}_m)\right\} \quad (5.33)$$

であり，D は $\boldsymbol{o}_t$ の次数，$'$ は転置，M は混合数，$\boldsymbol{\Sigma}_m^{-1}$ は $\boldsymbol{\Sigma}_m$ の逆行列，$|\boldsymbol{\Sigma}_m|$ は $\boldsymbol{\Sigma}_m$ の行列式を表わします．多次元データを利用した場合，$|\boldsymbol{\Sigma}_m|$ は共分散行列となりますが，この実験では，次元間の相関が無いと仮定した対角共分散行列を使用します．この場合，共分散行列は

$$\boldsymbol{\Sigma}_m = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ 0 & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \sigma_D^2 \end{pmatrix} \quad (5.34)$$

となり，密度関数は，

$$\mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) = \frac{1}{\sqrt{\prod_{d=1}^{D} 2\pi\sigma_d^2}} \exp\left\{-\frac{1}{2}\sum_{d=1}^{D} \frac{(\boldsymbol{o}_t^{(d)} - \boldsymbol{\mu}_m^{(d)})^2}{\sigma_d^2}\right\} \quad (5.35)$$

で与えられます．ここで，$\boldsymbol{o}_t^{(d)}$ は観測ベクトル $\boldsymbol{o}_t$ の第 d 次元，$\boldsymbol{\mu}_m^{(d)}$ は第 m 混合の平均値ベクトル $\boldsymbol{\mu}_m$ の第 d 次元です．

連続 HMM のパラメータ推定は，離散記号を出力する HMM に対する方法と同様の考え方で行うことができます．Baum-Welch の再推定公式を連続出力密度の HMM に対して適用することができます．この場合，

$$\tilde{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \gamma_t(j,k)}, \quad (5.36)$$

$$\tilde{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\boldsymbol{O}_t}{\sum_{t=1}^{T} \gamma_t(j,k)}, \quad (5.37)$$

$$\tilde{\Sigma}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)(\boldsymbol{O}_t - \boldsymbol{\mu}_{jk})(\boldsymbol{O}_t - \boldsymbol{\mu}_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)} \quad (5.38)$$

となります．ここで，$\gamma_t(j,k)$ は時刻 t において状態 S_J にある場合，$\boldsymbol{O}_t$ の第 k 番目の混合成分に対応しています．すなわち，

$$\gamma_t(j,k) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \cdot \frac{c_{jk}\mathcal{N}(\boldsymbol{O}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})}{\sum_{m=1}^{M} c_{jm}\mathcal{N}(\boldsymbol{O}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})} \quad (5.39)$$

式 5.36 の c_{jk} は，状態 S_j にあって第 k 混合成分を用いている回数の期待値と状態 S_j にある回数の期待値の比となっています．同様に，平均値ベクトル $\boldsymbol{\mu}_{jk}$ の再推定式（式 5.37）では，式 5.36 の分子を観測ベクトルで重み付けしているので，観測ベクトルのうち第 k 成分に帰する部分の割合の期待値になっています．共分散行列の再推定式（式 5.38）も同様の解釈で理解することができます．

5.4.3　スケーリング（scaling）

前向きパスアルゴリズム（5.3.1 節）や後ろ向きパスアルゴリズム（5.3.2 節）によって $\alpha_t(i)$ や $\beta_t(i)$ を求めるとき，パスが進むとともにこれらの値が徐々に小さくなり，コンピュータによる計算においてはアンダーフローが起こることが多いです．これを回避するための何らかの工夫が必要です．一つの方法は，前向きアルゴリズムで $\alpha_t(i)$ $(1 \leq j \leq N)$ が得られるたびに

$$c_t \triangleq \left[\sum_i \alpha_t(i) \right]^{-1} \tag{5.40}$$

を計算します．そして

$$\alpha_t(i) \triangleq c_t \alpha_t(i) \tag{5.41}$$

という**スケーリング（scaling）**を行います．$\beta_t(i)$ についても，後ろ向きアルゴリズムで $\beta_t(i)$ $(1 \leq j \leq N)$ が得られるたびに，上記の係数 c_t を用いて

$$\beta_t(i) \triangleq c_t \beta_t(i) \tag{5.42}$$

とします．このようにスケーリングを施した $\alpha_t(i)$，$\beta_t(i)$ を用いてパラメータの更新を行う計算法については，文献 [6] を参照してください．$P(\boldsymbol{O}|\lambda)$ そのものはアンダーフローのため計算できないことが多いのですが，$\log P(\boldsymbol{O}|\lambda)$ は

$$\log P(\boldsymbol{O}|\lambda) = -\sum_{i=1}^{T} \log c_t \tag{5.43}$$

で求められることが知られています [9]．

スケーリングはこの実習の単語 HMM の学習プログラム `train`（6.5.4 節）でつかわれています．C 言語のソースファイル `train.c` の `scale` という配列がスケーリング係数 c_t です．

5.5　学習から認識までの手順

HMM を使って時系列パターンの認識を行う手順をまとめます．

1. 認識対象を定め，それをいくつのカテゴリに分類するのかを決めます．
2. HMM の出力記号，状態数などを決めます．
3. カテゴリ毎に学習パターンを用意します．
4. HMM パラメータの初期値を設定します．
5. Baum-Welch アルゴリズムを用いて，カテゴリ毎に HMM のパラメータを推定します．
6. Forward アルゴリズム，Backward アルゴリズム，あるいは Viterbi アルゴリズムを用いて，未知パターンの出力確率をカテゴリ毎に計算し，出力確率が最大となるカテゴリを選んで認識結果とします．

5.6 実習

HMM の確率計算を計算機プログラムで実行してみましょう．状態数 $N = 4$, 出力シンボル種類数 $M = 2$, シンボル $\mathcal{V} = \{v_1 = 0, v_2 = 1\}$, 状態遷移確率は

$$\boldsymbol{A} = \begin{pmatrix} 0.2 & 0.1 & 0.3 & 0.4 \\ 0.1 & 0.5 & 0.1 & 0.3 \\ 0.3 & 0.4 & 0.2 & 0.1 \\ 0.2 & 0.1 & 0.4 & 0.3 \end{pmatrix}, \tag{5.44}$$

シンボル出力確率は

$$\boldsymbol{B} = \begin{pmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \\ 0.1 & 0.9 \\ 0.2 & 0.8 \end{pmatrix}, \tag{5.45}$$

初期状態確率が $\boldsymbol{\pi} = \{0.3, 0.4, 0.1, 0.2\}$ である図 5.8 のような HMM λ を考えます．

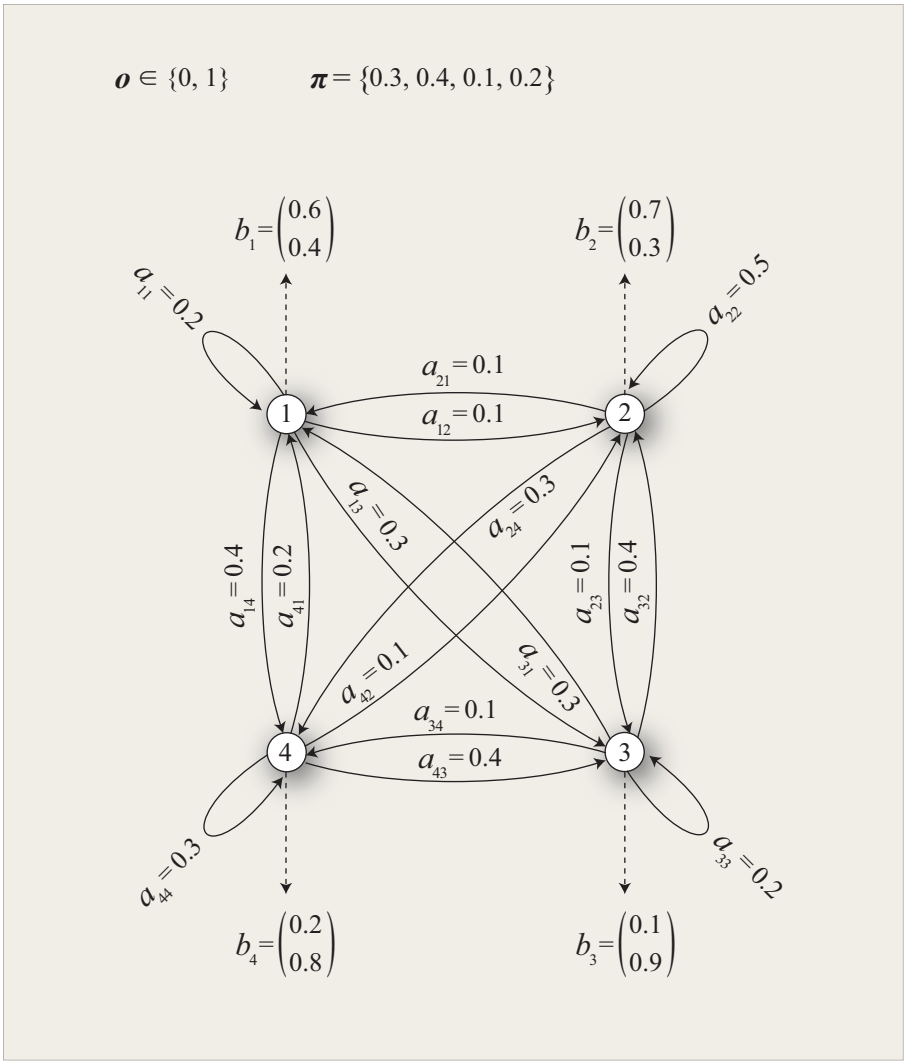

図 5.8: 4 状態離散出力 HMM

以下の説明は，計算機実験を`~/asr/drill` で行うことを前提としています．

5.6.1 Forward アルゴリズム

観測系列 $\boldsymbol{O} = 0\,1\,1\,0$ に対して，$\alpha_t(i)$ および $P(\boldsymbol{O}|\lambda)$ を Forward アルゴリズムで求めてください．

forward.c というCの言語のソースファイルがあります．ただし，肝心のコードの部分が空白になっています．穴埋め指示のある部分を埋めてソースコードを完成させてください．HMM の出力記号は $\mathcal{V} = \{v_1 = 0, v_2 = 1\}$ ですが，v_k の値が配列 `O[T]` の要素の値となっていて，シンボル出力確率の配列の添字に対応しています．C 言語の配列の添字は 0 から始まることに注意してください．完成したら，以下のコマンドを入力してコンパイルを実行します．

```
[~/asr/drill]% make drillF
```

ソースコードに C 言語の文法エラーがある場合は，その旨，メッセージが出力されるので，エラーメッセージが出なくなるまで修正してください．文法エラーがない場合は，drillF というコマンドができているので，これを入力すると，paramFB.txt から HMM のパラメータを読み込み，forward アルゴリズムが実行され，$\alpha_t(i)$ の値と $P(\boldsymbol{O}|\lambda)$ が表示されます．

```
[~/asr/drill]% drillF
```

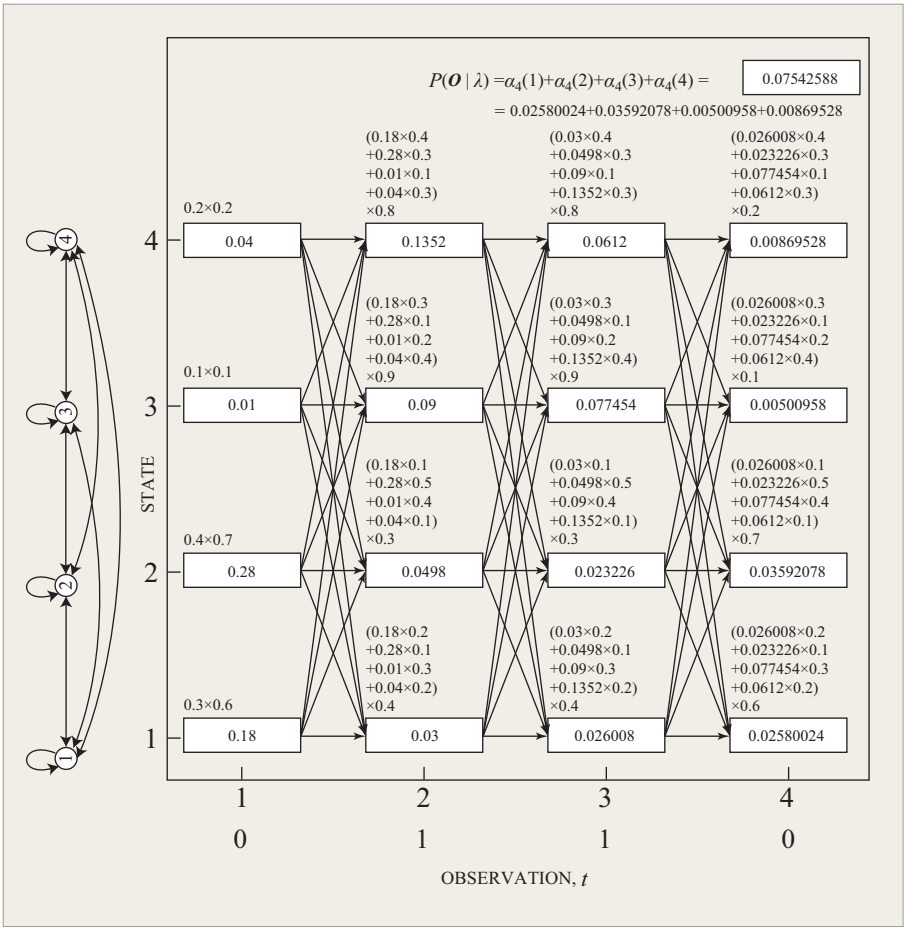

図 5.9: Forward アルゴリズム計算の過程

コマンド実行によって端末に表示される計算結果は図 5.9 と合致しているでしょうか？ レポートにはソースコードの穴埋め部分，および drillF コマンドの実行結果を載せてください．

5.6.2 Backward アルゴリズム

観測系列 $\boldsymbol{O} = 0\,1\,1\,0$ に対して，$\beta_t(i)$ および $P(\boldsymbol{O}|\lambda)$ を backward アルゴリズムで求めてください．

backward.c という C 言語のソースファイルの一部が削除されています．穴埋め指示のある部分を埋めてソースコードを完成させてください．以下のコマンドを入力してコンパイルを実行します．

```
[~/asr/drill]% make drillB
```

ソースコードに C 言語の文法エラーがある場合は，その旨，メッセージが出力されるので，エラーメッセージが出なくなるまで修正してください．文法エラーがない場合は drillB というコマンドができているので，これを入力すると，paramFB.txt から HMM のパラメータを読み込み，Backward アルゴリズムが実行され，$\beta_t(i)$ の値と $P(\boldsymbol{O}|\lambda)$ が表示されます．

```
[~/asr/drill]% drillB
```

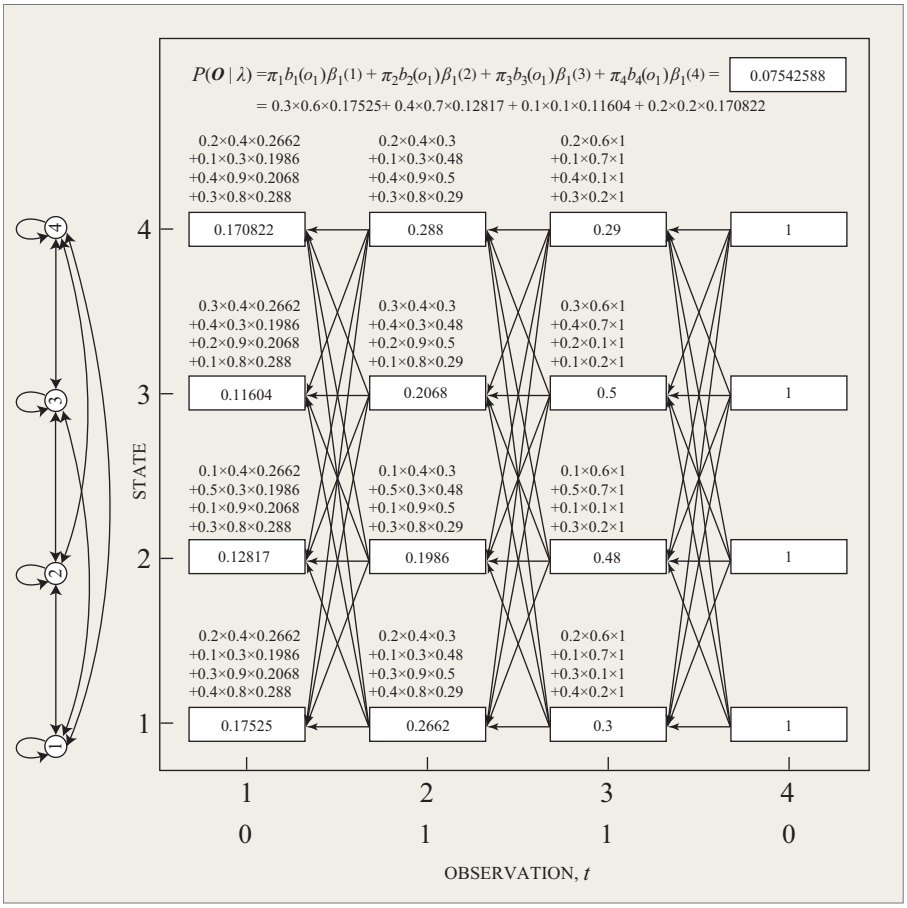

図 5.10: Backward アルゴリズム計算の過程

計算結果が図 5.10 と合致しているでしょうか？ レポートにはソースコードの穴埋め部分，および `drillB` コマンドの実行結果を載せてください．

5.6.3 Baum-Welch アルゴリズム

与えられた記号列 (離散値) から HMM パラメータの推定値を求める Baum-Welch アルゴリズムをプログラミング言語で記述しやすい形にまとめたのが 5-19 ページの Algorithm 1 です．

HMM の学習実験 [9]

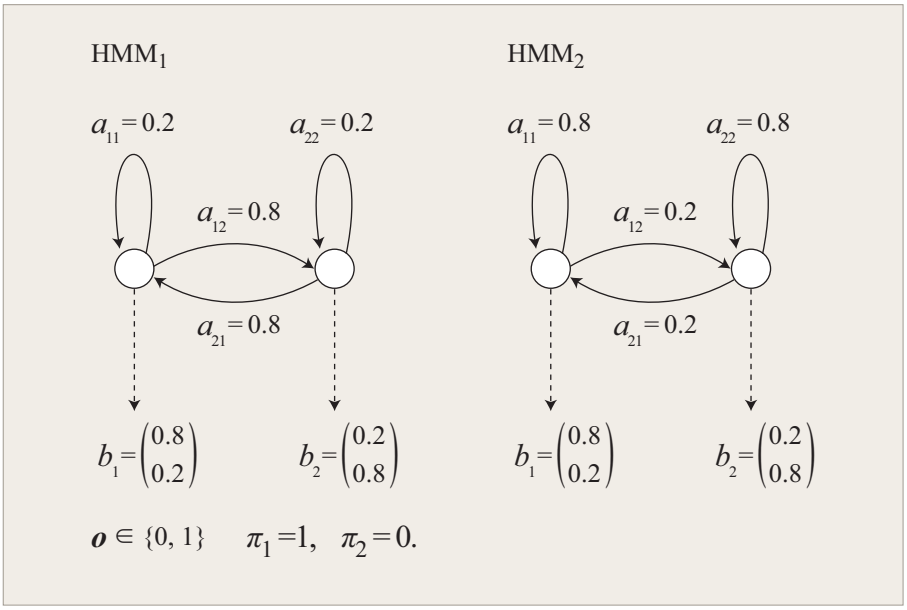

図 5.11: 2 状態 2 出力離散 HMM

状態数が 2，出力記号が 2 種類=$\{0,1\}$ の異なる HMM を HMM_1，HMM_2 とします（図 5.11）．それぞれのパラメータ λ_1 と λ_2 を

$$\lambda_1 = (\boldsymbol{\Pi}_1, \boldsymbol{A}_1, \boldsymbol{B}_1),\ \boldsymbol{\Pi}_1 = (1.0, 0.0),\ \boldsymbol{A}_1 = \begin{pmatrix} 0.2 & 0.8 \\ 0.8 & 0.2 \end{pmatrix},\ \boldsymbol{B}_1 = \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix}$$

$$\lambda_2 = (\boldsymbol{\Pi}_2, \boldsymbol{A}_2, \boldsymbol{B}_2),\ \boldsymbol{\Pi}_2 = (1.0, 0.0),\ \boldsymbol{A}_2 = \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix},\ \boldsymbol{B}_2 = \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix}$$

と設定します．HMM による記号生成のシミュレーションプログラムを用いて，$\mathrm{HMM}_1, \mathrm{HMM}_2$ からそれぞれ長さ 1000 の記号列 $\boldsymbol{O}_1 = o_1(1), o_1(2), \ldots, o_1(1000)$，$\boldsymbol{O}_2 = o_2(1), o_2(2), \ldots, o_2(1000)$ を生成します．そして，Baum-Welch アルゴリズムを用いて，記号列 $\boldsymbol{O}_1$ からパラメータ λ_1，記号列 $\boldsymbol{O}_2$ からパラメータ λ_2 を推定する実験を行います．パラメータの初期値は，

$$\boldsymbol{\Pi}_i = (1.0, 0.0),\quad \boldsymbol{A}_i = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix},\quad \boldsymbol{B}_i = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix},\quad (i = 1, 2)$$

Algorithm 1 Baum-Welch Algorithm

Baum-Welch Algorithm
$\bar{\lambda} = (\bar{\pi}_1, \ldots, \bar{\pi}_N, \bar{a}_{1,1}, \ldots, \bar{a}_{N,N}, \bar{b}_{1,1}, \ldots, \bar{b}_{N,M})$: パラメータ初期値 [入力]
$\boldsymbol{v}$: 出力記号 [入力]，　$o(1), o(2), \ldots, o(T)$: 記号列 [入力]
ϵ: 収束判定用閾値 [入力]
$\bar{\lambda}$: パラメータ推定値 [出力]
$P(\lambda^{old}) \leftarrow -\infty$: 負の十分大きな値
START: $\lambda \leftarrow \bar{\lambda}$: パラメータの更新
Forward アルゴリズムにより λ の値から $\alpha_t(i)$ $(1 \leq t \leq T,\ 1 \leq i \leq N)$ と $P(\lambda)$ を計算する．
if $\log P(\lambda) - \log P(\lambda^{old}) < \epsilon$ **then**
　$\bar{\lambda} \leftarrow \lambda$: 推定結果
　exit
else
　$P(\lambda^{old}) \leftarrow P(\lambda)$
end if
Backward アルゴリズムにより λ の値から $\beta_t(i)$ $(1 \leq t \leq T,\ 1 \leq i \leq N)$ と $P(\lambda)$ を計算する．
for $i = 1$ to N **do**

$$\bar{\pi}_i = \frac{\alpha_1(i)\beta_1(i)}{P(\lambda)}$$

end for
for $i = 1$ to N **do**
　for $j = 1$ to N **do**

$$\bar{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{i,j} b_j(o(t+1)) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)}$$

　end for
end for
for $j = 1$ to N **do**
　for $k = 1$ to M **do**

$$\bar{b}_{j,k} = \frac{\sum_{t=1}^{T} \delta_{v(k),o(t)} \alpha_t(j)\beta_t(j)}{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j)}$$

$\delta_{v(k),o(t)}$ はクロネッカーのデルタ．$\delta_{x,y} = 1\,(x = y),\ 0\,(x \neq y)$.
　end for
end for
$\bar{\lambda} = (\bar{\pi}_1, \ldots, \bar{\pi}_N, \bar{a}_{1,1}, \ldots, \bar{a}_{N,N}, \bar{b}_{1,1}, \ldots, \bar{b}_{N,M})$
goto START
end of *Baum-Welch Algorithm*

と設定します．

Baum-Welch アルゴリズムを実行したところ，HMM_1，HMM_2 について，繰り返し回数 100 回ほどでそれぞれのパラメータの推定値，

$$\tilde{\lambda}_1 = (\tilde{\boldsymbol{\Pi}}_1, \tilde{\boldsymbol{A}}_1, \tilde{\boldsymbol{B}}_1),\ \tilde{\boldsymbol{\Pi}}_1 = (1.0, 0.0),\ \tilde{\boldsymbol{A}}_1 = \begin{pmatrix} 0.23 & 0.81 \\ 0.79 & 0.23 \end{pmatrix},\ \tilde{\boldsymbol{B}}_1 = \begin{pmatrix} 0.74 & 0.27 \\ 0.27 & 0.74 \end{pmatrix}$$

$$\tilde{\lambda}_2 = (\tilde{\boldsymbol{\Pi}}_2, \tilde{\boldsymbol{A}}_2, \tilde{\boldsymbol{B}}_2),\ \tilde{\boldsymbol{\Pi}}_2 = (1.0, 0.0),\ \tilde{\boldsymbol{A}}_2 = \begin{pmatrix} 0.77 & 0.23 \\ 0.18 & 0.82 \end{pmatrix},\ \tilde{\boldsymbol{B}}_2 = \begin{pmatrix} 0.82 & 0.18 \\ 0.18 & 0.82 \end{pmatrix}$$

が得られました．

HMM による認識実験 [9]

HMM_1，HMM_2 を用いて，長さ 50 の出力記号列をそれぞれ 100 パターンずつ生成し，推定されたパラメータの値 $\tilde{\lambda}_1$，$\tilde{\lambda}_2$ を用いてこれらのパターンを認識する実験を行います．HMM_1 によって生成されるパターンをカテゴリ 1 に属するパターン，HMM_2 によって生成されるパターンをカテゴリ 2 に属するパターンと呼ぶことにします．

認識するためには，Forward アルゴリズムを用い，各パターン $\boldsymbol{O}$ に対して $P(\boldsymbol{O}|\tilde{\lambda}_1)$ と $P(\boldsymbol{O}|\tilde{\lambda}_2)$ を計算します．そして，$P(\boldsymbol{O}|\tilde{\lambda}_1) \geq P(\boldsymbol{O}|\tilde{\lambda}_2)$ ならばそのパターンはカテゴリ 1 に，$P(\boldsymbol{O}|\tilde{\lambda}_1) < P(\boldsymbol{O}|\tilde{\lambda}_2)$ ならばそのパターンはカテゴリ 2 に属していると判定します．

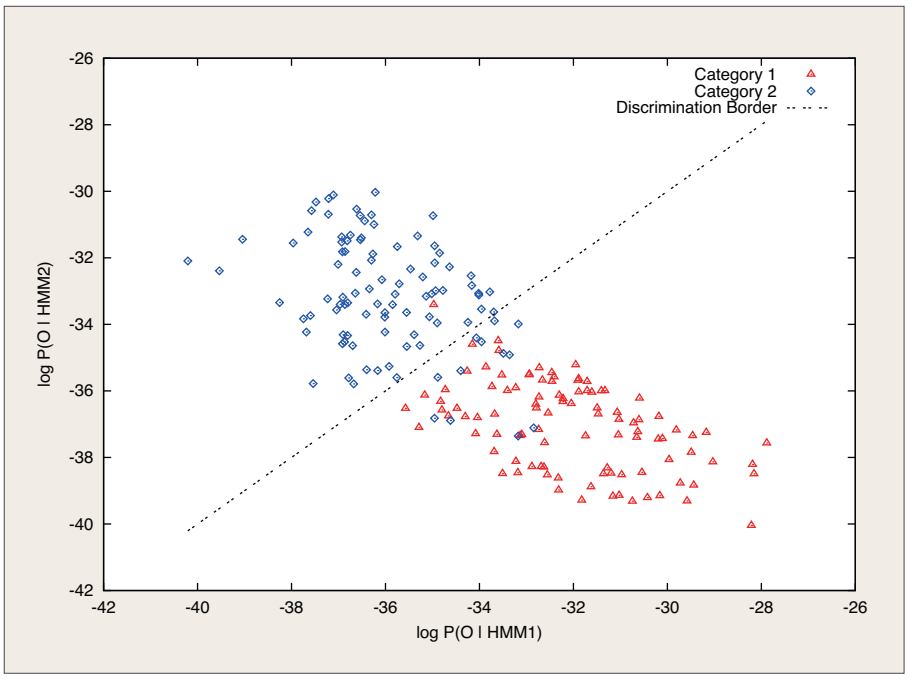

図 5.12: 尤度計算結果．カテゴリ 1 のパターン横軸は $\log P(\boldsymbol{O}|\tilde{\lambda}_1)$，縦軸は $\log P(\boldsymbol{O}|\tilde{\lambda}_2)$．△ はカテゴリ 1 のパターン，◇ はカテゴリ 2 のパターン，直線（点線）は **識別境界**，すなわち $\log P(\boldsymbol{O}|\tilde{\lambda}_1) = \log P(\boldsymbol{O}|\tilde{\lambda}_2)$ を満たす領域です．

図 5.12 はカテゴリ 1 とカテゴリ 2 に属するパターンについての認識プログラムの計算結果を図示したものです．横軸は $\log P(\boldsymbol{O}|\tilde{\lambda}_1)$，縦軸は $\log P(\boldsymbol{O}|\tilde{\lambda}_2)$ を示しています．点線で示しているのは $\log P(\boldsymbol{O}|\tilde{\lambda}_1) = \log P(\boldsymbol{O}|\tilde{\lambda}_2)$ なる識別境界です．したがって，カテゴリ 1 のパターン (△) のうち点線より下にあるパターンは正しく認識され，上にあるパターンは誤って認識されることになります．この例の場合，100 パターンのうち正しく認識されたパターンは 99 個です．カテゴリ 2 に属するパターン (◇) については，点線より上にあるパターンは正しく認

識され，下にあるパターンは誤って認識されることになります．この例の場合，100 パターンのうち正しく認識されたパターンは 88 個です．カテゴリ 1 とカテゴリ 2 の平均認識率は 93.5%となりました．

実習手順

(1) Baum-Welch アルゴリズムのコーディング `baumwelch.c` という C 言語のソースファイルがあります．Baum-Welch アルゴリズムのソースコード `baumwelch.c` が完成したら，以下のコマンドを入力してコンパイルを実行します．

```
[~/asr/drill]% make drillT
```

ソースコードに C 言語の文法エラーがある場合は，その旨メッセージが出力されるので，エラーメッセージが出なくなるまで修正してください．文法エラーがない場合は，`drillT` というコマンドができています．

(2) 学習データの生成 記号列生成のために `gen` というプログラムが用意されています．まず，

```
[~/asr/drill]% make gen
```

と入力してコンパイルしてください．HMM_1 から 1000 個の記号を生成するには，

```
[~/asr/drill]% gen 1 1000 > d1.txt
```

と入力します．$\boldsymbol{O}_1$ のサンプルが 1 行 1 個のテキスト形式で出力されるので，リダイレクションを利用して `d1.txt` に保存しています．このプログラムは実行された時刻が異なると異なる系列を生成します．したがって，このデータを用いた実験結果はこのテキストの例とは若干異なることになります．同様に，

```
[~/asr/drill]% gen 2 1000 > d2.txt
```

と入力して $\boldsymbol{O}_2$ のサンプルを 1000 個生成し，`d2.txt` に保存しています．

(3) HMM パラメータの推定 生成した $\boldsymbol{O}_1$ と $\boldsymbol{O}_2$ を用いて，HMM のパラメータを推定します．まず，HMM_1 のパラメータ推定を行いましょう．次のように入力します．

```
[~/asr/drill]% drillT d1.txt paramHMM1.txt > d1.log
```

Baum-Welch アルゴリズムにより，パラメータの推定が繰り返されます．収束条件が満たされると，学習データ `d1.txt` を用いて学習された HMM のパラメータは端末に表示されるとともに，`paramHMM1.txt` に保存されます．ファイル `d1.log` には，繰り返し計算の毎回ごとの暫定の HMM パラメータの値が記録されます．同様に HMM_2 の学習も行います．

```
[~/asr/drill]% drillT d2.txt paramHMM2.txt > d2.log
```

(4) 学習過程のグラフ化 この実習の HMM の初期パラメータは全て 0.5 ですが，Baum-Welch アルゴリズムによって再推定を繰り返すことにより，学習データの信号の生成源のパラメータに単調に近づいて行きます．その様子をグラフに描いてみましょう．次の 2 つのコマンドは，d1.log あるいは d2.log に記録された a_{11} と a_{12} の再推定ごとの値を曲線として描く gnuplot スクリプトです．

```
[~/asr/drill]% drawLC_d1
[~/asr/drill]% drawLC_d2
```

コマンドを実行すると画面にグラフが表示され，その後に端末でリターンキーを押すと表示が消え，表示されたグラフが LC_a11-a12_HMM1.png と LC_a11-a12_HMM2.png というファイルに保存されます．実際にパラメータ推定を行ってみたら，HMM_1 は 107 回，HMM_2 は 125 回で再推定が収束しました．その様子を上記コマンドでグラフにしたのが図 5.13 です．再推定回数や収束曲線は学習データによって異なりますが，おおよそこの事例のようになります．

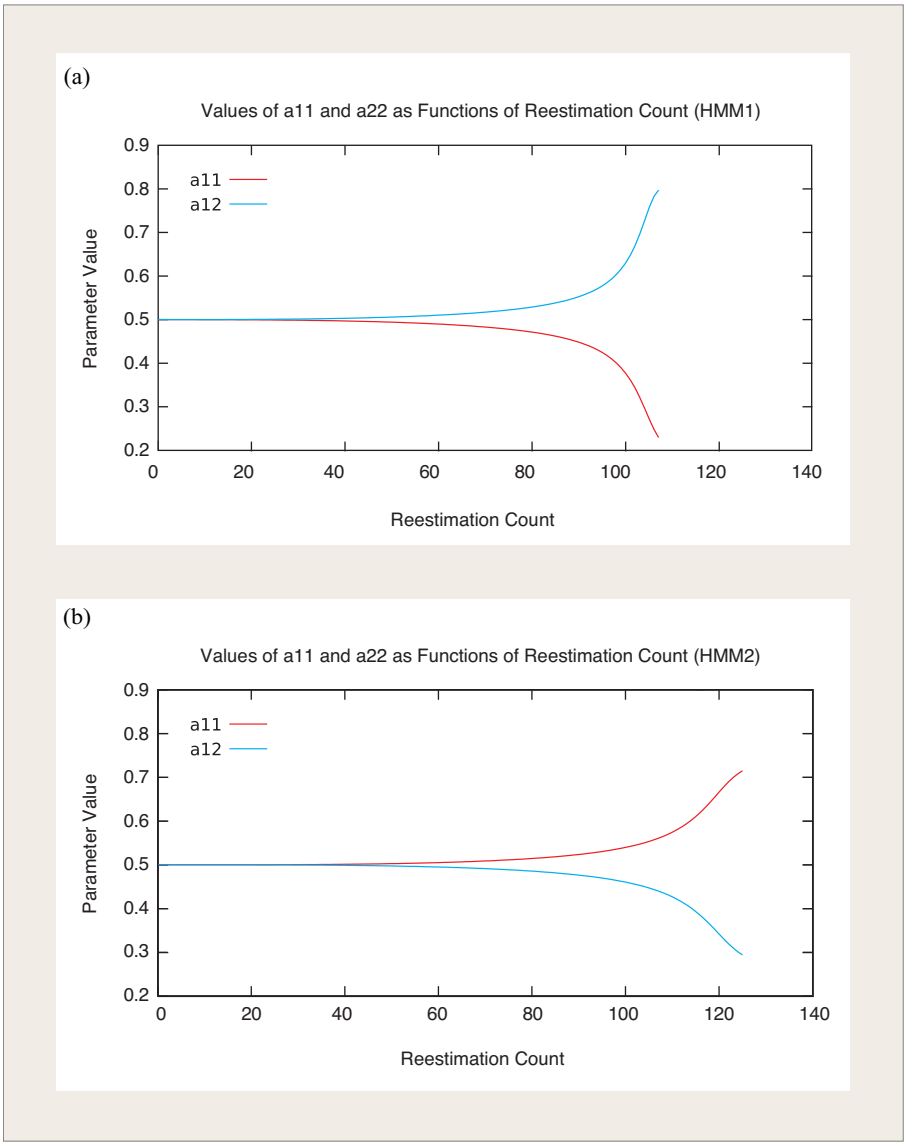

図 5.13: (a)HMM_1 と (b)HMM_2 の状態遷移確率 a_{11} と a_{12} が再推定の繰り返しにより信号源のパラメータに近づく様子．`drawLC_d1` と `drawLC_d2` を用いた eps 形式の出力を説明用に曲線の色などを修整したもの．この実行例の再推定回数は HMM_1 が 107 回，HMM_2 が 125 回であった．他のパラメータも 0.2 あるいは 0.8 に漸近する同じ軌跡を描く．

(5) 学習済みモデルの評価

(5.1) 評価データの生成

HMM_1，HMM_2 から長さ 50 の出力記号列をそれぞれ 100 パターン生成します．学習データの生成に用いたプログラム `gen` を再び使います．プログラムのあるディレクトリにある `data1` と `data2` というサブディレクトリに，HMM_1 からのパターンと HMM_2 からのパターンのファイルを保存します．まず，

```
[~/asr/drill]% genData
```

上記のコマンドを実行することにより，認識用データが生成され，そのファイル名の一覧が `data1.list` と `data2.list` に保存されます．

ファイルの行数や内容を確認してください．テキストファイルの行数を数えるには，例えば，端末に `wc -l data1.list` と入力します．

(5.2) 認識率測定

認識用データの HMM_1 と HMM_2 に対する尤度を Forward アルゴリズム (穴埋めして完成した forward 関数を使います) によって計算するプログラムを作ります．

```
[~/asr/drill]% make drillR
```

と入力してコンパイルを実行します．`drillR` は，HMM のパラメータファイルと認識を指定して実行します．HMM_1 からのパターンを認識させてみると次のようになりました．テキストの横幅に収めるためにコマンドを途中で改行して書いていますが，実際は 1 行です．

```
[~/asr/drill]% drillR paramHMM1.txt paramHMM2.txt
                                    data1.list > data1.result
n1= 94
n2= 6
```

端末に表示される数値はカテゴリ 1（`n1`) とカテゴリ 2 (`n2`) の正解数です．これはあくまでも一例です．`data1.result` というファイルには，各パターンについて，1 行に $\log P(\boldsymbol{O}|\tilde{\lambda}_1)$ と $\log P(\boldsymbol{O}|\tilde{\lambda}_2)$ が並んで記録されます．内容を確かめておいてください．`data2.list` についても，同様に認識を実行します．

```
[~/asr/drill]% drillR paramHMM1.txt paramHMM2.txt
                                    data2.list > data2.result
n1= 17
n2= 83
```

前述のように，`gen` によって生成される記号列は `gen` が実行された時刻に依存するため，推定される HMM のパラメータの推定値は若干異なったものになります．さらに，認識データも生成時刻によって異なるので，必ずしも上記の結果と同じ結果にはなりません．

学習と認識を複数回繰り返すと，学習データと認識データの確率的揺らぎにより，HMM パラメータの推定結果は揺らぎ，認識結果も毎回揺らぎます．興味があれば試してみると良いでしょう．

認識結果の表示 HMM_1 からのパターンの認識結果 `data1.result` と HMM_2 からのパターンの認識結果 `data2.result` ができたら，

```
[~/asr/drill]% drawR
```

と入力します．図 5.12 のようなグラフが表示されます（ちなみに，図 5.12 のデータは上記で得られたのとは別の試行の結果です．）．`drawR` を実行した端末で Return キーを押すと，グラフの表示は消え，`drawR.png`

という PNG 形式のファイルにグラフが保存されます．drawR は gnuplot スクリプトです．他のファイル名や形式で保存したい場合は編集して使ってください．

5.6.4 多次元ガウス確率密度関数

単語認識のための HMM の学習や認識のために必要な多次元ガウス確率密度関数のプログラミングをします．この作業は~/asr/wrecog で行います．C 言語のソースファイル program/gpdf.c があります．式 5.35 に従ってこのソースを完成させてください．実例を用いて確率密度を計算するプログラムを以下のようにして作成します．

```
[~/asr/wrecog]% make -C program drillG
```

サンプルの HMM パラメータと時系列データを用いて多次元ガウス確率密度を計算し，尤度を算出するテストプログラム drillG が出来ます．

プログラムが正しく出来ているか確認しましょう．端末に次のように入力して，計算確認用の HMM パラメータ sample/sampleG.hmm に記述されている多次元ガウス確率密度関数を用いて MFCC 時系列 sample/sampleG.mfcc の Viterbi 尤度 $\log P^*(\boldsymbol{O}|\lambda)$（式 5.13）を計算して端末に表示します．正しくできていれば次のような計算結果の数値が得られます．

```
[~/asr/wrecog]% drillG sample/sampleG.hmm sample/sampleG.mfcc
log likelihood= -3.818172e+02
```

単語認識実験には多次元ガウス確率密度関数の計算が必要です．上記と同じ結果が出たら，次のコマンドを実行してください．単語認識実験に必要な全てのプログラムのコンパイルが実行されます．

```
[~/asr/wrecog]% make -C program all
```

5.6.5 レポート（第 2 週）

1. 実習課題の目的
2. HMM 算法のプログラミングとシミュレーション
   (a) 前向きパスアルゴリズム
      i. `forward.c` の穴埋め部分.
      ii. `drillF` の実行結果.
   (b) 後ろ向きパスアルゴリズム
      i. `backward.c` の穴埋め部分.
      ii. `drillB` の実行結果.
   (c) Baum-Welch アルゴリズム
      i. `baumwelch.c` の穴埋め部分.
      ii. `drillT` による HMM_1 と HMM_2 のパラメータ推定結果.
      iii. `drawLC_d?`による HMM_1 と HMM_2 の a_{11} と a_{12} の学習曲線.
      iv. `drillR` による認識結果
         A. 認識率
         B. 尤度計算結果（図 5.12 と同形式の `drawR` の出力）.
   (d) 多次元ガウス確率密度関数
      i. `gpdf.c` の穴埋め部分.
      ii. `drillG` の実行結果.
3. 考察
4. 一般事項
   (a) 本日のポイントは何であったか？
   (b) 良くわかったこと
   (c) わからなかったこと
   (d) 要望
   (e) 感想，その他