

U N I V E R S I T É M O N T P E L L I E R I I
— Sciences et Techniques du Languedoc —
U.F.R. Sciences de Montpellier

Thèse de Doctorat de l'Université

Discipline : Informatique
Formation Doctorale : Informatique
École Doctorale : Information, Structures et Systèmes

**Modèles de Markov cachés et apprentissage
par fusions d'états : algorithmes, applications,
utilisations pour le test de circuits intégrés**

par

Laurent BRÉHÉLIN

présentée et soutenue publiquement le *11 Juin 2001* devant le jury :

M. Michel Habib, Professeur, Université Montpellier II Président
M. Olivier Gascuel, Directeur de recherche, CNRS/LIRMM Directeur de thèse
M. Dominique Cellier, Maître de conférences, Université de Rouen Rapporteur
Mme Sybille Hellebrand, Professeur, Université d'Innsbruck Rapporteur
M. Jean-François Mari, Professeur, Université de Nancy II Rapporteur
M. Christian Landrault, Directeur de recherche, CNRS/LIRMM Examinateur

Chapitre 2

Modèles de Markov cachés

2.1 Introduction

Les modèles de Markov cachés (Hidden Markov Models ou HMMs) ont été introduits par Baum et ses collaborateurs dans les années 1960-70s [Baum et al., 1970]. Cette classe de modèles est fortement apparentée aux automates probabilistes (PAs) [Casacuberta, 1990]. Un automate probabiliste est défini par une structure composée d'états et de transitions, et par un ensemble de distributions de probabilités sur les transitions. À chaque transition est associée un symbole d'un alphabet fini. Ce symbole est généré à chaque fois que la transition est empruntée. Un HMM se définit également par une structure composée d'états et de transitions et par un ensemble de distributions de probabilités sur les transitions. La génération de symboles est généralement attachée aux états, mais elle peut éventuellement être attachée aux transitions [Bahl et al., 1983]. On associe alors à chaque état (ou transition) non pas un symbole, mais une distribution de probabilités sur les symboles de l'alphabet. Une illustration de la forte parenté entre HMMs et PAs peut être trouvée dans la propriété que n'importe quel HMM peut être simulé par un PA du même nombre d'états [Abe et Warmuth, 1992]. Notons que la propriété inverse n'est pas vraie.

Le premier domaine auquel les HMMs ont été appliqués fut le traitement de la parole au début des années 1970 [Baker, 1975], [Rabiner, 1989]. Dans ce domaine, les HMMs se sont rapidement imposés comme le modèle de référence et une majeure partie des techniques d'utilisation et d'implémentation des HMMs (choix d'une structure ou d'un type de structure, problèmes des probabilités faibles, ...) ont été développés dans le cadre de ces applications. Ces techniques furent ensuite appliquées et adaptées avec succès aux problèmes de reconnaissance de textes manuscrits [Kundu et al., 1988], [Schenkel et al., 1994]) et d'analyse de séquences biologiques [Haussler et al., 1992], [Durbin et al., 1998]. Les HMMs ont également été appliqués à d'autres domaines comme la reconnaissance d'images [Povlow et Dunn, 1995], ou la modélisation d'un signal musical [Raphael, 1998]. Depuis très récemment, avec l'augmentation importante de la masse des données électroniques, une nouvelle application prometteuse semble être l'extraction d'informations à partir de données textuelles [Seymore et al., 1999], [Freitag et McCallum, 1999].

L'objectif de ce chapitre est de présenter une vue d'ensemble des HMMs. Les algo-

rithmes et les problèmes classiques de la littérature sont exposés et illustrés autant que possible par des applications réelles. La section 2.2 présente de manière formelle les modèles de Markov cachés. À la section 2.3 nous présentons les deux principaux types de problèmes qui peuvent être résolus par les HMMs, c'est-à-dire les problèmes de discrimination et les problèmes de segmentation, et nous exposons les algorithmes classiques associés à leur résolution : l'algorithme de calcul de la probabilité d'une séquence pour les applications de discrimination et l'algorithme de calcul du chemin le plus probable pour les applications de segmentation. Quelle que soit l'application traitée, un des principaux problèmes que l'on rencontre lorsqu'on utilise des HMMs est la construction d'un modèle adapté aux séquences que l'on souhaite modéliser. La section 2.4 est dédiée à ce problème d'apprentissage. Nous verrons les différents cas de figure qu'il présente, dépendant des séquences à modéliser et des connaissances *a priori* dont on dispose.

2.2 Présentation

Un HMM H peut être défini par un quadruplet $\langle \mathcal{Q}, \Sigma, \mathcal{T}, \mathcal{G} \rangle$ avec :

- \mathcal{Q} , un ensemble de N états ; \mathcal{Q} contient deux états spéciaux muets *start* et *end* qui servent respectivement à débuter et conclure une séquence.
- Σ , un alphabet de symboles ;
- $\mathcal{T} = \mathcal{Q} - \{\text{end}\} \times \mathcal{Q} - \{\text{start}\} \rightarrow [0, 1]$, une matrice indiquant les probabilités de transition d'un état à l'autre ; on note $P(s \rightarrow s')$ la probabilité de transition de l'état s vers l'état s' ;
- $\mathcal{G} = \mathcal{Q} - \{\text{start}, \text{end}\} \times \Sigma \rightarrow [0, 1]$, une matrice indiquant les probabilités de génération associées aux états ; on note $P(o|s)$ la probabilité de générer le symbole o à partir de l'état s .

La définition proposée ici n'est pas la définition originale donnée par Baum [Baum et al., 1970] dont elle diffère par l'introduction des états muets *start* et *end*. L'état *start* n'en-traine pas de modification essentielle du modèle et remplace avantageusement l'utilisation de la distribution de probabilités *a priori* associée aux états pour débuter les séquences dans la définition de Baum. L'état *end*, par contre, n'a pas d'équivalent dans la définition originale. Néanmoins, la définition présentée ici est celle habituellement utilisée dans la plupart des applications. Nous nous limiterons ici à la modélisation de séquences de symboles appartenant à un alphabet fini : un ensemble de *phonèmes* dans le cadre d'applications de reconnaissance de la parole, les quatre bases azotées (adénine, guanine, cytosine et thymine) pour la modélisation de séquences d'ADN, l'ensemble des acides aminés pour les séquences protéiques, les mots d'un certain corpus pour l'extraction d'information à partir de données textuelles, Néanmoins les HMMs peuvent également être utilisés pour la modélisation de signaux continus [Rabiner, 1989]. On définit la structure d'un HMM par l'ensemble de ses états, ainsi que par les transitions et les symboles de probabilité non nulle. La figure 2.1 représente un exemple de HMM avec sept états et onze transitions.

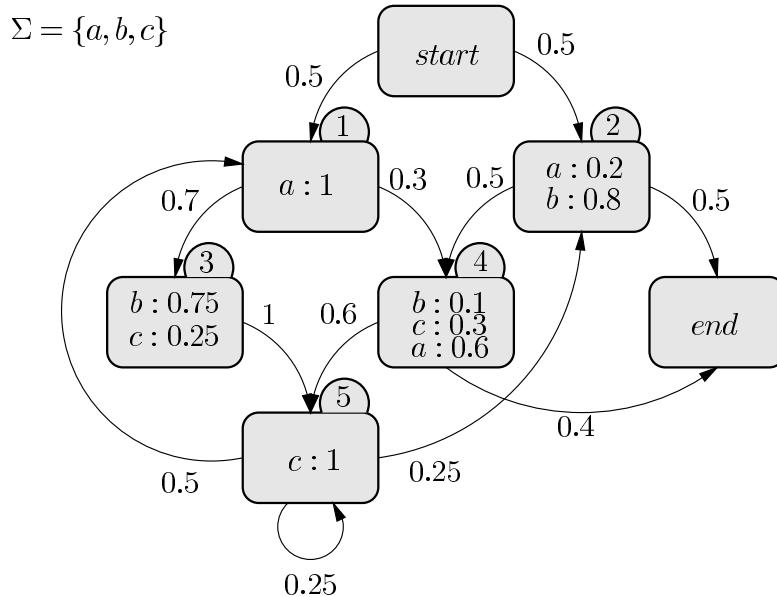


FIG. 2.1: Un exemple de HMM.

La procédure de génération d'une séquence de symboles à l'aide d'un HMM consiste à partir de l'état *start*, à se déplacer d'état en état suivant les probabilités de transition, et à générer un symbole sur chaque état rencontré en utilisant la distribution de probabilités de génération qui lui est associée. Lorsqu'un symbole a été généré, on choisit une transition sortante suivant la distribution de probabilités de transition associée à l'état courant, et la procédure est réitérée jusqu'à atteindre l'état *end*. Par exemple, la séquence *abccb* peut être générée par le HMM de la figure 2.1 en partant de l'état *start*, puis en allant sur l'état 1 puis sur l'état 3, puis 5, 5, 5, 2 et *end*. Notons que cette séquence de symboles peut également être générée en suivant la séquence d'état *start* – 1 – 4 – 5 – 5 – 5 – 2 – *end*, ou *start* – 2 – 4 – 5 – 5 – 5 – 2 – *end*. La probabilité de génération suivant la première séquence d'états – ou *chemin* – est la probabilité de passer de l'état *start* à l'état 1 (0.5), multipliée par la probabilité que l'état 1 émette le symbole *a* (1), multipliée par la probabilité de transition de 1 vers 3 (0.7), multipliée par $\dots = 6.6 \cdot 10^{-3}$. Suivant les deuxième et troisième chemins, cette probabilité est respectivement $2.3 \cdot 10^{-4}$ et $7.5 \cdot 10^{-5}$. Comme il n'y a pas d'autres chemins qui permettent de générer cette séquence, la probabilité de génération de *abccb* par le HMM est $6.6 \cdot 10^{-3} + 2.3 \cdot 10^{-4} + 7.5 \cdot 10^{-5} = 6.9 \cdot 10^{-3}$. Les HMMs définissent donc un processus stochastique non déterministe, au sens où une même séquence de symboles peut être générée de plusieurs manières différents. On comprend alors mieux le nom donné à ce modèle : le processus de génération est un processus *markovien* – la probabilité de transition vers un état ne dépend que de l'état actuel et non des états rencontrés précédemment – qui est *caché* car il est impossible de connaître le processus suivi pour la génération d'un séquence de symboles donnée.

2.3 Utilisations et algorithmes de base

Les applications classiques des HMMs sont les problèmes de discrimination et les problèmes de segmentation. On trouve dans la première catégorie les applications de reconnaissance d'un mot parmi un ensemble de mots possibles à partir d'un signal audio (reconnaissance de la parole) [Rabiner, 1989] ou d'une écriture manuscrite [Kundu et al., 1988], de reconnaissance d'une famille de protéines à partir d'une séquence d'acides aminés [Hausler et al., 1992] ... Pour ces applications, on construit généralement un HMM différent pour chaque classe à reconnaître – par exemple un HMM pour chaque mot dans le cadre de la reconnaissance de la parole –, et, comme pour les unigrams, la reconnaissance d'une séquence consiste alors à calculer la probabilité de génération de la séquence par chacun des HMM, et à assigner à cette séquence sa classe la plus probable. Ce type d'application nécessite donc un algorithme efficace pour le calcul de la probabilité de génération d'une séquence. Cet algorithme est présenté à la section 2.3.1.

Dans les applications de segmentation qui utilisent les HMMs on trouve les problèmes de découpage d'un signal musical en notes [Raphael, 1998], de localisation de régions codantes/non-codantes dans une chaîne de nucléotides [Krogh et al., 1994], d'extraction de phrases clefs d'un texte parmi un ensemble important de phrases à faible niveau informationnel [Freitag et McCallum, 1999], de découpage d'un article en différents champs (titre, date, nom de l'auteur, ...) [Seymore et al., 1999] ... On utilise pour ces applications un HMM dont les états sont *typés*. La procédure de segmentation d'une séquence de symboles consiste alors à calculer à l'intérieur du HMM le chemin qui a la probabilité maximale de générer cette séquence. On associe ensuite à chaque symbole de la séquence son type, en fonction du type de l'état correspondant dans le chemin calculé. Considérons par exemple le HMM de la figure 2.2. En plus des états *start* et *end* ce HMM est composé de quatre autres états typés ARTICLE, ADJECTIF, NOM et VERBE. Remarquons que deux mots apparaissent dans plusieurs états : "modèle", qui peut être utilisé comme adjectif et comme nom, et "classe" qui peut être utilisé comme nom et comme verbe. Considérons maintenant la phrase "Le modèle classe la séquence.". Notre objectif est de segmenter cette phrase, c'est-à-dire d'assigner à chaque mot son type. Deux chemins du HMM permettent de générer cette phrase : *start-ARTICLE-ADJECTIF-NOM-ARTICLE-NOM-end* et *start-ARTICLE-NOM-VERBE-ARTICLE-NOM-end*. La probabilité de génération suivant le premier chemin est $1.8 \cdot 10^{-7}$, tandis qu'elle est de $6.22 \cdot 10^{-6}$ suivant le second chemin. La segmentation la plus probable est donc celle induite par le second chemin. Cette segmentation nous permet alors, par exemple, de conclure que "classe" est utilisé dans cette phrase comme verbe et non comme nom, ce qui n'est *a priori* pas évident – toute considération sémantique mise à part – si l'on considère que ce mot est généralement plus souvent utilisé comme nom que comme verbe. Si l'on étudie de plus près les paramètres du HMM, on se rend compte que ce qui nous a permis d'arriver à cette conclusion est que, si ce n'était pas le cas (premier chemin), alors "modèle" serait un adjectif. Or un adjectif a peu de chance de suivre immédiatement un article (probabilité 0.1) et "modèle" n'est en définitive pas un adjectif très usité (probabilité 0.1). Ces paramètres dépendent bien évidemment du corpus de phrases étudié, et le HMM, pour être efficace, doit donc refléter le plus fidèlement possible les caractéristiques de ce corpus. C'est tout le problème

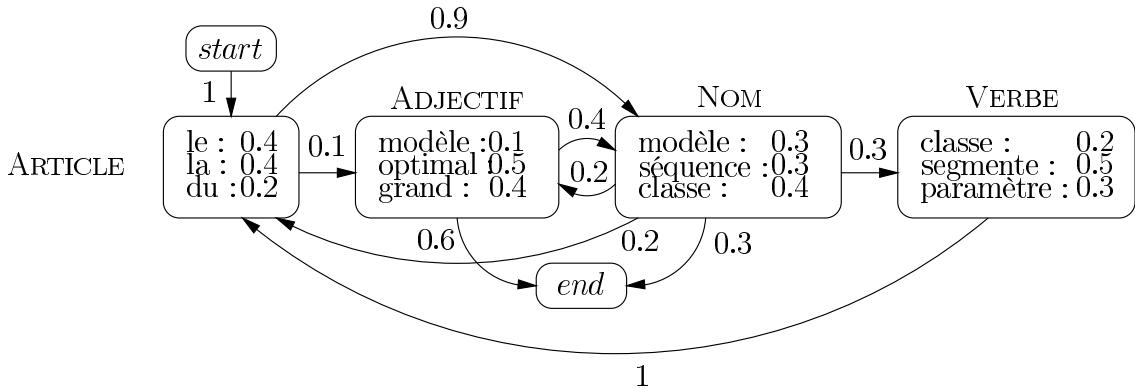


FIG. 2.2: Un HMM dont les états sont typés (ARTICLE, ADJECTIF, NOM, et VERBE) et qui peut être utilisé pour la segmentation de phrases.

du paramétrage abordé à la section 2.4. Les applications de segmentation reposent donc sur une recherche du chemin optimal et nécessitent un algorithme efficace pour résoudre ce problème. Cet algorithme est présenté à la section 2.3.2.

2.3.1 Algorithme *forward-backward*

Si l'on veut calculer la probabilité de générer la séquence de symboles $O = o_1 \dots o_T$ à l'aide du HMM $H = \langle Q, \Sigma, \mathcal{T}, \mathcal{G} \rangle$, l'approche directe consiste à calculer, comme on l'a fait dans la partie précédente, la probabilité de génération pour chaque chemin possible et faire la somme de ces probabilités. La probabilité de générer O suivant le chemin $S = s_0 s_1 \dots s_T s_{T+1}$ avec $s_0 = \text{start}$ et $s_{T+1} = \text{end}$ est

$$P(O, S) = P(s_0 \rightarrow s_1)P(o_1|s_1) \dots P(s_{T-1} \rightarrow s_T)P(o_T|s_T)P(s_T \rightarrow s_{T+1}). \quad (2.1)$$

La probabilité de générer O avec H obtenue en sommant sur l'ensemble des séquences d'états possibles est donc

$$P(O|H) = \sum_{\langle s_0 \dots s_{T+1} \rangle} P(s_0 \rightarrow s_1)P(o_1|s_1) \dots P(s_{T-1} \rightarrow s_T)P(o_T|s_T)P(s_T \rightarrow s_{T+1}), \quad (2.2)$$

avec $\langle s_0 \dots s_{T+1} \rangle$ l'ensemble des séquences d'états possibles. Si N est le nombre d'états du HMM, alors le nombre de chemins possibles pour générer une séquence de symboles de longueur T est de l'ordre de N^T . Comme, pour chaque chemin, le calcul de la formule (2.1) demande de l'ordre de T opérations, le calcul de la probabilité de génération de O par H suivant l'équation (2.2) est en $O(TN^T)$. Un rapide calcul numérique nous montre que même pour des valeurs de T et N peu élevées, l'approche directe n'est clairement pas acceptable : avec $N = 5$ et $T = 100$ le calcul de $P(O|H)$ demande approximativement 10^{72} opérations. Heureusement une procédure bien plus efficace existe pour réaliser ce calcul : l'algorithme *forward-backward* [Rabiner, 1989].

Considérons la variable *forward* $\alpha_t(s)$ définie par

$$\alpha_t(s) = P(o_1 o_2 \dots o_t, s_t = s | H),$$

qui exprime la probabilité d'avoir généré la séquence $o_1 \dots o_t$ en partant de l'état $start$ et d'être arrivé sur l'état s à l'instant t . Cette variable peut être calculée de manière récursive :

1. Initialisation. À $t = 1$ on a :

$$\alpha_1(s) = P(start \rightarrow s) \cdot P(o_1|s).$$

2. Récursion. Pour $t = 2, \dots, T$ on a :

$$\alpha_t(s) = \left(\sum_{s' \in Q} \alpha_{t-1}(s') P(s' \rightarrow s) \right) P(o_t|s). \quad (2.3)$$

Or, connaissant $\alpha_T(s)$ – c'est à dire la probabilité d'avoir généré la séquence O et d'être arrivé sur l'état s – pour tout $s \in Q$, le calcul de $P(O|H)$ est immédiat :

$$P(O|H) = \sum_{s \in Q} \alpha_T(s) P(s \rightarrow end). \quad (2.4)$$

On peut alors proposer une version complète de l'algorithme de programmation dynamique *forward* (voir algorithme 1).

Algorithme 1: ALGORITHME FORWARD

Données : $O = o_1 \dots o_T$ une séquence;
 H un HMM.

```

pour chaque  $s \in Q$  faire
   $\alpha_1(s) = P(start \rightarrow s) \cdot P(o_1|s)$  ;
pour chaque  $t = 2$  à  $T$  faire
  pour chaque  $s \in Q$  faire
    Calculer et stocker la valeur de  $\alpha_t(s)$  à l'aide de la formule (2.3);
Calculer  $P(O|H)$  à l'aide de la formule (2.4);
```

Examinons la complexité de cet algorithme. La phase d'initialisation requiert une opération pour chaque état du HMM, donc au total $O(N)$ opérations. Pour la phase de récursion, pour chaque instant et chaque état, on réalise $O(N)$ opérations. Sommé sur l'ensemble des états et la totalité des instants, la phase de récursion requiert donc $O(N^2T)$ opérations. Une fois les $\alpha_T(s)$ calculés, le calcul de $P(O|H)$ suivant la formule (2.4) demande une opération pour chaque état, donc au total $O(N)$ opérations. Le calcul de $P(O|H)$ à l'aide de l'algorithme *forward* ne requiert donc au total que $O(N^2T)$ opérations. En reprenant la même application numérique que précédemment ($N = 5$ et $T = 100$), le calcul de $P(O|H)$ suivant cet algorithme nécessite approximativement 3000 opérations.

Cet algorithme est appelé *forward* car la récursion est réalisée en avant : on calcule tout d'abord la probabilité de générer le premier symbole de la séquence, puis à chaque étape de la récursion on rajoute un symbole et on réitère la procédure jusqu'à ce qu'on

ait calculé la probabilité de génération de la séquence entière. Bien que moins naturel, un algorithme similaire, l'algorithme *backward*, peut être utilisé pour réaliser ce calcul à l'envers. On utilise alors la variable *backward* $\beta_t(s)$:

$$\beta_t(s) = P(o_{t+1}o_{t+2}\cdots o_T | s_t = s, H),$$

qui exprime la probabilité de générer la séquence $o_{t+1}o_{t+2}\dots o_T$ en partant de l'état s et d'arriver sur l'état *end*. La procédure de récursion est alors la suivante :

1. Initialisation. À $t = T$ on a :

$$\beta_T(s) = P(s \rightarrow \text{end}).$$

2. Récursion. Pour $t = T - 1, \dots, 1$ on a :

$$\beta_t(s) = \left(\sum_{s' \in \mathcal{Q}} P(s \rightarrow s') P(o_{t+1}|s') \beta_{t+1}(s') \right). \quad (2.5)$$

Et, connaissant $\beta_1(s)$ – c'est à dire la probabilité de générer la séquence O en partant de l'état s et d'arriver sur l'état *end* – pour chaque état s , le calcul de $P(O|H)$ peut alors être réalisé suivant la formule

$$P(O|H) = \sum_{s \in \mathcal{Q}} P(\text{start} \rightarrow s) P(o_1|s) \beta_1(s). \quad (2.6)$$

L'algorithme *backward* est illustré par l'algorithme 2. Sa complexité est la même que celle de l'algorithme *forward*, c'est-à-dire $O(N^2T)$.

Algorithme 2: ALGORITHME BACKWARD

Données : $O = o_1 \dots o_T$ une séquence;
 H un HMM.

```

pour chaque  $s \in \mathcal{Q}$  faire
   $\beta_T(s) = P(s \rightarrow \text{end})$  ;
pour chaque  $t = T - 1$  à 1 faire
  pour chaque  $s \in \mathcal{Q}$  faire
    Calculer et stocker la valeur de  $\beta_t(s)$  à l'aide de la formule (2.5);
  Calculer  $P(O|H)$  à l'aide de la formule (2.6);

```

2.3.2 Algorithme de Viterbi

Le problème que l'on se pose ici est de trouver, étant donnés une séquence de symboles $O = o_1 \dots o_T$ et un HMM $H = \langle \mathcal{Q}, \Sigma, \mathcal{T}, \mathcal{G} \rangle$, la séquence d'états du HMM qui a la probabilité maximale de générer O . Ce qui nous préoccupe n'est pas la valeur de la probabilité maximale mais le chemin – appelé *chemin de Viterbi* et noté S^* – qui permet

de générer la séquence O avec cette probabilité. De manière similaire à l'approche utilisée pour le calcul de $P(O|H)$, l'approche directe pour résoudre ce problème consiste à calculer la probabilité de génération suivant tous les chemins possibles et de choisir parmi ces chemins celui qui a la probabilité la plus élevée. Cette approche a, comme pour le calcul de $P(O|H)$, une complexité en $O(TN^T)$ et est donc également inapplicable. L'algorithme de Viterbi [Rabiner, 1989] est un algorithme de programmation dynamique très similaire à l'algorithme *forward* et qui permet de résoudre efficacement ce problème.

Considérons la variable $\delta_t(s)$ définie par

$$\delta_t(s) = \max_{s_0 \cdots s_{t-1}} P(s_0 \cdots s_t = s, o_1 \cdots o_t | H),$$

et qui exprime la probabilité maximale de générer la séquence $o_1 \cdots o_t$ suivant un unique chemin partant de l'état *start* et arrivant sur l'état s à l'instant t . De la même manière que pour $\alpha_t(s)$, cette variable peut être calculée de manière récursive :

1. Initialisation. À $t = 1$ on a :

$$\delta_1(s) = P(\text{start} \rightarrow s)P(o_1|s).$$

2. Récursion. Pour $t = 2, \dots, T$ on a :

$$\delta_t(s) = \max_{s' \in \mathcal{Q}} (\delta_{t-1}(s')P(s' \rightarrow s)) P(o_t|s). \quad (2.7)$$

Alors, connaissant $\delta_T(s)$ pour tous les états $s \in \mathcal{Q}$, on peut calculer la probabilité maximale de générer O avec H suivant un simple chemin (on désignera cette probabilité par $P(O, S^*|H)$) en appliquant la formule :

$$P(O, S^*|H) = \max_{s \in \mathcal{Q}} (\delta_T(s)P(s \rightarrow \text{end})).$$

Malheureusement, ce n'est pas la valeur de cette probabilité qui nous intéresse mais réellement le chemin qui permet de générer O avec cette probabilité. On doit donc, à chaque étape t de la récursion et pour chaque état s , mémoriser l'état s' qui maximise l'équation (2.7). Désignons par $\psi_t(s)$ cet état. Alors :

1. À $t = 1$ on a :

$$\psi_1(s) = \text{start}.$$

2. Pour $t = 2, \dots, T$ on a :

$$\psi_t(s) = \operatorname{argmax}_{s' \in \mathcal{Q}} (\delta_{t-1}(s')P(s' \rightarrow s)). \quad (2.8)$$

Une fois les variables $\delta_t(s)$ et $\psi_t(s)$ calculées pour chaque étape de la récursion et pour chaque état, il ne reste plus qu'à lancer une procédure récursive de rétro-propagation pour "dérouler" le chemin de Viterbi $S^* = \text{start}, s_1^*, \dots, s_T^*, \text{end}$ en partant de l'état *end* :

1. Initialisation. À $t = T$:

$$s_T^* = \operatorname{argmax}_{s \in \mathcal{Q}} (\delta_T(s)P(s \rightarrow \text{end})). \quad (2.9)$$

2. Rétro-propagation. Pour $t = T - 1, T - 2, \dots, 0$:

$$s_t^* = \psi_t(s_{t+1}^*). \quad (2.10)$$

On peut alors proposer une version complète de l'algorithme de Viterbi (voir algorithme 3). On notera que, mise à part la phase de rétro-propagation, l'algorithme de Viterbi est très similaire à l'algorithme *forward*. La principale différence résulte de la maximisation des probabilités attachées aux états précédents (formule (2.7)) au lieu du calcul de la somme de ces probabilités (formule (2.3)) dans le cas de l'algorithme *forward*. La phase de rétro-propagation étant en $O(T)$, la complexité de l'algorithme de Viterbi est donc la même que celle de l'algorithme *forward*, c'est-à-dire $O(N^2T)$.

Algorithme 3: ALGORITHME DE VITERBI

Données : $O = o_1 \dots o_T$ une séquence;
 H un HMM.

```

pour chaque  $s \in Q$  faire
   $\delta_1(s) = P(\text{start} \rightarrow s) \cdot P(o_1|s)$  ;
   $\psi_1(s) = \text{start}$  ;
pour chaque  $t = 2$  à  $T$  faire
  pour chaque  $s \in Q$  faire
    Calculer et stocker la valeur de  $\delta_t(s)$  à l'aide de la formule (2.7);
    Calculer et stocker la valeur de  $\psi_t(s)$  à l'aide de la formule (2.8);
  Déterminer l'état  $s_T^*$  à l'aide de la formule (2.9);
pour chaque  $t = T - 1$  à  $0$  faire
   $\lfloor$  Déterminer l'état  $s_t^*$  à l'aide de la formule (2.10);
```

2.4 Apprentissage

On a vu à la partie précédente les deux algorithmes classiques utilisés dans le cadre des applications de reconnaissance et de segmentation utilisant les HMMs. Ces deux algorithmes supposent que l'on dispose d'un HMM construit et paramétré de manière à modéliser de façon satisfaisante les séquences que l'on souhaite traiter. La question qui nous préoccupe dans cette partie est celle de la construction d'un tel HMM. Dans le cas le plus favorable, le HMM recherché peut être construit directement à partir des connaissances *a priori* dont on dispose sur le domaine. Ce cas est, par exemple celui de Krogh *et al.* dans leur article sur la modélisation de séquences d'ADN de *E.coli* [Krogh et al., 1994]. Dans cet article, ils utilisent les HMMs pour segmenter les séquences d'ADN en régions codantes et non-codantes. Les séquences manipulées sont des séquences d'acides nucléiques et un certain nombre de connaissances – telles que la structure générale du HMM et la fréquence des acides nucléiques en fonction de leur rôle dans les séquences – leur permettent de construire facilement leur modèle. Malheureusement ce cas de figure

ne se rencontre que rarement, et, dans la plupart des applications, le ou les HMMs désirés doivent être construits à l'aide d'un algorithme d'apprentissage.

On peut distinguer dans le problème de l'apprentissage d'un HMM deux cas de figure distincts, suivant que la structure – le nombre d'états du HMM et les transitions et émissions autorisées – est connue ou ne l'est pas. Lorsque la structure est connue, le problème se réduit à un problème d'*entraînement* consistant à estimer les paramètres numériques – les distributions de probabilités de transition et de génération – de manière à expliquer au mieux les séquences d'apprentissage. Ce problème (considéré comme NP-difficile [Abe et Warmuth, 1992]) est celui traité à la section 2.4.1. Pour certaines applications, on ne dispose pas de connaissances suffisantes pour inférer naturellement la structure du HMM. L'apprentissage devient alors encore plus difficile. Il ne suffit plus de paramétriser une structure mais il faut également déduire cette structure des exemples fournis. Ce problème fait l'objet de la première partie de cette thèse. Il est exposé à la section 2.4.2 et sera plus largement développé dans les chapitres suivants.

2.4.1 Apprentissage quand la structure est connue

Nous nous intéressons dans cette partie à l'apprentissage – ou entraînement – d'un HMM à partir d'une structure connue et d'un ensemble de séquences d'apprentissage. Une approche possible est, suivant le principe du maximum de vraisemblance, de chercher les paramètres $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ du HMM qui maximisent la probabilité de génération des séquences d'apprentissage. Soit $\mathcal{O} = \{O^1, \dots, O^K\}$ l'ensemble des séquences d'apprentissage. On suppose que ces séquences sont indépendantes et donc que la probabilité de générer l'ensemble d'apprentissage est simplement le produit des probabilités de génération de chacune des séquences. Notre but est alors de trouver les paramètres $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ de H qui maximisent

$$P(\mathcal{O}|H) = \prod_{k=1}^K P(O^k|H). \quad (2.11)$$

Une approche alternative au principe du maximum de vraisemblance est de chercher à maximiser la probabilité de génération des séquences d'apprentissage suivant leur chemin de Viterbi. Le but est alors de trouver les paramètres $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ de H qui maximisent

$$P(\mathcal{O}, \mathcal{S}^*|H) = \prod_{k=1}^K P(O^k, S^{k*}|H), \quad (2.12)$$

avec S^{k*} le chemin de Viterbi de la séquence O^k dans H , et $\mathcal{S}^* = \{S^{1*}, \dots, S^{K*}\}$. Les paramètres ne sont donc pas estimés en maximisant la *vraie* probabilité de génération des séquences d'apprentissage mais la probabilité de génération suivant les chemins les plus probables. Cette approche peut sembler moins rigoureuse que l'approche basée sur le principe du maximum de vraisemblance, mais elle est très utilisée en pratique et possède quelques arguments en sa faveur. D'abord on peut se dire que si le HMM recherché est

destiné à être appliqué à des problèmes de segmentation *via* la recherche des chemins de Viterbi, il semble judicieux de l'entraîner de cette manière. Ensuite on remarque dans la pratique que la probabilité de génération d'un séquence de symboles suivant son chemin de Viterbi est en général beaucoup plus élevée que suivant n'importe quel autre chemin. Cette observation a conduit à l'hypothèse, appelée *hypothèse de Viterbi*, que tous les chemins, excepté le chemin de Viterbi, ont une probabilité nulle ou négligeable d'engendrer la séquence. Sous cette hypothèse, la probabilité de génération d'une séquence O^k par un HMM H peut être approximée par la probabilité de génération de O^k suivant son chemin de Viterbi dans H . On a alors :

$$P(\mathcal{O}|H) \approx P(\mathcal{O}, \mathcal{S}^*|H),$$

et maximiser l'équation (2.12) revient à maximiser une approximation de l'équation (2.11). Le dernier argument en faveur de cette méthode d'entraînement est que l'on peut, dans le cas de figure particulier où les chemins de Viterbi sont connus, trouver la solution optimale au sens de la maximisation de l'équation (2.12) (ce cas est plus amplement développé à la section 2.4.1.1), alors que l'on ne connaît aucune méthode optimale pour la maximisation de l'équation (2.11). Abe et Warmuth [Abe et Warmuth, 1992] ont étudié la question de savoir s'il existe un algorithme d'entraînement efficace d'un automate probabiliste, dans le cadre d'une adaptation naturelle du paradigme de *PAC-learning* introduit par Valiant [Valiant, 1984]. Leur modèle est inspiré du modèle d'apprentissage non supervisé de Laird [Laird, 1988]. La question est de savoir, pour un ensemble d'exemples de taille *réduite*, s'il existe un algorithme qui converge, avec une *forte probabilité*, vers une solution *optimale*. Dans ce cadre, ils prouvent que la classe des automates probabilistes n'est pas entraînable en temps polynômial à la taille de l'alphabet, à moins que $RP = NP$. Pour les HMMs, la question reste ouverte à notre connaissance, mais on peut raisonnablement penser que le problème est aussi difficile.

Suivant la méthode d'estimation des paramètres choisie, il existe deux heuristiques permettant l'entraînement des HMMs. Ces deux algorithmes, très similaires, sont tous deux issus d'une méthode générale – les algorithmes d'*Expectation-Maximization* (EM)¹ [Dempster et al., 1977] – servant à l'estimation des paramètres d'une grande famille de modèles probabilistes. Pour des raisons de simplicité, nous commencerons par exposer l'algorithme adapté à la maximisation de l'équation (2.12) (section 2.4.1.1), et nous verrons à la section 2.4.1.2 l'algorithme servant à maximiser l'équation (2.11).

2.4.1.1 Entrainement de Viterbi

Comme évoqué dans la partie précédente, il existe un cas où il est possible de paramétriser la structure du HMM de manière à maximiser l'équation (2.12) de façon optimale. Ce cas est celui rencontré lorsque l'on connaît les chemins de Viterbi des séquences d'apprentissage dans la structure. En effet, on peut alors associer à chaque état, chaque transition

¹En fait, l'algorithme d'entraînement adapté à la maximisation de l'équation (2.12) n'est pas à proprement parlé un algorithme de type *EM*, mais une adaptation de cet algorithme qui remplace la phase *E* par la recherche des chemins de Viterbi.

et chaque symbole attaché aux états du HMM le nombre de fois où ils sont utilisés pour générer ces séquences. Soit n_s , $n_{s \rightarrow s'}$ et n_s^o respectivement le nombre de fois où l'état s est utilisé, le nombre de fois où la transition $s \rightarrow s'$ est utilisée et le nombre de fois où le symbole o est généré par l'état s dans les chemins de Viterbi. Alors la formule (2.12) peut être réécrite

$$P(\mathcal{O}, \mathcal{S}^* | H) = \prod_{s \in Q} \left(\prod_{o \in \Sigma} P(o|s)^{n_s^o} \prod_{s' \in Q} P(s \rightarrow s')^{n_{s \rightarrow s'}} \right). \quad (2.13)$$

Maximiser cette formule revient à maximiser indépendamment chacun de ses sous-produits. On peut donc estimer les paramètres \mathcal{T} en maximisant l'expression $\prod_{s' \in Q} P(s \rightarrow s')^{n_{s \rightarrow s'}}$ et les paramètres \mathcal{G} en maximisant $\prod_{o \in \Sigma} P(o|s)^{n_s^o}$. Les estimateurs de \mathcal{T} et \mathcal{G} sont donc respectivement

$$\hat{P}(s \rightarrow s') = \frac{n_{s \rightarrow s'}}{n_s} \quad (2.14)$$

et

$$\hat{P}(o|s) = \frac{n_s^o}{n_s}. \quad (2.15)$$

Cette méthode d'estimation n'est possible que dans le cas favorable où les chemins de Viterbi sont connus. Lorsque ce n'est pas le cas, le problème est évidemment plus difficile et l'algorithme d'entraînement de Viterbi peut alors être une solution.

L'algorithme d'entraînement de Viterbi (voir algorithme 4) est un algorithme de réestimation itératif. Il consiste, à partir d'un paramétrage initial du HMM, à calculer les chemins de Viterbi des séquences d'apprentissage à l'aide de l'algorithme de Viterbi décrit à la section 2.3.2. Les chemins de Viterbi sont utilisés pour calculer, comme on l'a fait ci-dessus, le nombre de fois où chaque transition, chaque état et chaque symbole attaché aux états est utilisé. On réestime alors à l'aide des formules (2.14) et (2.15) les paramètres du HMM, on reparamètre la structure à l'aide de ces estimations et on réitère la procédure jusqu'à obtention de la stabilité.

On peut montrer que la valeur de $P(\mathcal{O}, \mathcal{S}^* | H)$ augmente à chaque itération et que l'algorithme converge vers un optimum local [Juang et Rabiner, 1990]. Malheureusement, il existe généralement un grand nombre d'optima locaux et le paramétrage obtenu par ce processus dépend fortement du paramétrage initial choisi.

Éaminons la complexité de cet algorithme. À chaque tour, il faut calculer le chemin de Viterbi de chacune des K séquences d'apprentissage. Si on note T la taille totale des K séquences, ce calcul se fait en $O(N^2T)$. La réestimation des paramètres du HMM à l'aide des formules (2.14) et (2.15) se fait en $O(N^2)$. En bornant le nombre de tours de l'algorithme par une constante L (en pratique, une dizaine de tours suffit généralement à assurer la convergence de l'algorithme), la complexité totale de l'algorithme d'entraînement de Viterbi est donc $O(LN^2T)$.

Algorithme 4: ALGORITHME D'ENTRAÎNEMENT DE VITERBI

Données : $\mathcal{O} = \{O^1, \dots, O^K\}$;
H une structure de HMM.

Choisir un paramétrage initial $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ de *H*;

répéter

Initialiser toutes les variables n_s , $n_{s \rightarrow s'}$ et n_s^o à 0;

pour chaque $O^k \in \mathcal{O}$ **faire**

Calculer le chemin de Viterbi de O^k dans *H*;

Ajouter aux variables n_s , $n_{s \rightarrow s'}$ et n_s^o la contribution de O^k ;

Réestimer les paramètres λ de *H* en utilisant les formules (2.14) et (2.15);

jusqu'à stabilité du paramétrage;

2.4.1.2 Entrainement de Baum-Welch

L'algorithme d'entraînement de Baum-Welch estime les paramètres $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ du HMM en maximisant l'équation (2.11). C'est un algorithme de réestimation itératif qui fonctionne sur le même principe que l'algorithme d'entraînement de Viterbi. Dans ce dernier, on compte le nombre de fois où chaque état, chaque transition et chaque symbole attaché aux état est utilisé dans les chemins de Viterbi. Ici on veut maximiser les probabilités de génération réelles, et non celles des chemins les plus probables. On va donc associer au états, aux transitions et aux symboles le nombre de fois où ils sont utilisés pour toutes les séquences et *tous* les chemins susceptibles de générer les séquences, pondéré par la probabilité de chacun de ces chemins. Ces comptes pondérés sont alors utilisés pour réestimer les paramètres du modèle de la même manière que pour l'algorithme d'entraînement de Viterbi.

De manière plus formelle, considérons pour commencer une unique séquence d'observations $O = o_1 \dots o_T$. Notre but est de trouver les paramètres $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ qui maximisent la probabilité $P(O|H)$ de générer *O* avec *H*. Nous verrons ensuite comment étendre l'algorithme à un ensemble de séquences d'observations. Soit

$$\xi_t(s, s') = P(s_t = s, s_{t+1} = s' | O, H)$$

la probabilité qu'en générant *O* avec *H* on passe par l'état *s* à l'instant *t* et par l'état *s'* à l'instant *t + 1*. Cette équation peut se réécrire :

$$\xi_t(s, s') = \frac{P(s_t = s, s_{t+1} = s', O | H)}{P(O | H)},$$

et en utilisant les variables *forward* et *backward* introduites à la section 2.3.1 :

$$\xi_t(s, s') = \frac{\alpha_t(s) P(s \rightarrow s') P(o_{t+1} | s') \beta_{t+1}(s')}{P(O | H)}. \quad (2.16)$$

Considérons également la variable $\gamma_t(s)$:

$$\gamma_t(s) = P(s_t = s | O, H),$$

qui exprime la probabilité qu'en générant O avec H on se trouve sur l'état s à l'instant t . Exprimée en fonction de $\xi_t(s, s')$, $\gamma_t(s)$ s'écrit :

$$\gamma_t(s) = \sum_{s' \in Q} \xi_t(s, s'). \quad (2.17)$$

Si on somme $\gamma_t(s)$ sur l'ensemble des instants t , on obtient une quantité que l'on peut interpréter comme l'espérance du nombre de fois où l'état s est utilisé pour générer la séquence O . De même, si on somme $\xi_t(s, s')$ sur l'ensemble des instants t , on obtient une quantité que l'on peut interpréter comme l'espérance du nombre de fois où la transition $s \rightarrow s'$ est utilisée pour générer la séquence O . On a donc :

$$\sum_{t=0}^T \gamma_t(s) = \text{espérance du nombre de passages dans } s, \quad (2.18)$$

$$\sum_{t=0}^T \xi_t(s, s') = \text{espérance du nombre de transitions de } s \text{ vers } s', \quad (2.19)$$

et de manière similaire à (2.18) on a également :

$$\sum_{t=1 / o_t=o}^T \gamma_t(s) = \text{espérance du nombre de générations de } o \text{ dans } s. \quad (2.20)$$

En combinant les formules (2.18), (2.19) et (2.20) on peut alors proposer un ensemble d'estimateurs des paramètres $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ du HMM :

$$\begin{aligned} \hat{P}(s \rightarrow s') &= \frac{\text{espérance du nombre de transitions de } s \text{ vers } s'}{\text{espérance du nombre de passages dans } s} \\ &= \frac{\sum_{t=0}^T \xi_t(s, s')}{\sum_{t=0}^T \gamma_t(s)} \end{aligned}$$

$$\begin{aligned} \hat{P}(o|s) &= \frac{\text{espérance du nombre de générations de } o \text{ dans } s}{\text{espérance du nombre de passages dans } s} \\ &= \frac{\sum_{t=1 / o_t=o}^T \gamma_t(s)}{\sum_{t=0}^T \gamma_t(s)} \end{aligned}$$

Dans le cas où l'on dispose d'un ensemble de séquences d'observations $\mathcal{O} = \{O^1, \dots, O^K\}$, ces estimateurs se généralisent simplement de la manière suivante :

$$\hat{P}(s \rightarrow s') = \frac{\sum_{k=1}^K \sum_{t=0}^T \xi_t^k(s, s')}{\sum_{k=1}^K \sum_{t=0}^T \gamma_t^k(s)} \quad (2.21)$$

$$\hat{P}(o|s) = \frac{\sum_{k=1}^K \sum_{t=1 / o_t=o}^T \gamma_t^k(s)}{\sum_{k=1}^K \sum_{t=0}^T \gamma_t^k(s)} \quad (2.22)$$

On peut alors proposer une version complète de l'algorithme de Baum-Welch (voir algorithme 5). Comme pour l'algorithme d'entraînement de Viterbi, on peut prouver [Baum et al., 1970] que la probabilité de génération des séquences $P(\mathcal{O}|H)$ augmente à chaque itération et que l'algorithme converge vers un optimum local. Néanmoins, dans ce cas là, l'espace de recherche est un espace continu (on n'utilise pas des comptes discrets comme pour l'algorithme d'entraînement de Viterbi, mais l'espérance de ces comptes), et l'optimum local n'est donc en réalité jamais atteint [Durbin et al., 1998]. Il est alors nécessaire de définir un critère d'arrêt, par exemple en stoppant la procédure lorsque l'augmentation de $P(\mathcal{O}|H)$ est inférieure à un certain seuil, ou lorsque le nombre d'itérations effectuées est jugé suffisamment important.

Éaminons la complexité de cet algorithme. Le calcul des variables $\alpha_t^k(s)$ à l'aide de l'algorithme *forward* est en $O(N^2T)$. Celui des $\beta_t^k(s)$ à l'aide de l'algorithme *backward* est également en $O(N^2T)$. Le calcul des $\xi_t^k(s, s')$ à l'aide de la formule (2.16) se fait en $O(N^2)$. Le calcul des $\gamma_t^k(s)$ à l'aide de la formule (2.17) se fait également en $O(N^2)$. La complexité totale du calcul des expressions (2.18), (2.19) et (2.20) est $O(N^2T)$. Chacune de ces opérations est à faire à chaque tour de l'algorithme et pour chaque séquence de \mathcal{O} . En bornant le nombre de tours par une constante L , et en notant T la taille totale des K séquences d'apprentissage, on obtient une complexité totale de l'algorithme d'entraînement de $O(LN^2T)$.

Algorithme 5: ALGORITHME D'ENTRAÎNEMENT DE BAUM-WELCH

Données : $\mathcal{O} = \{O^1, \dots, O^n\}$;
 H une structure de HMM.

Choisir un paramétrage initial $\lambda = \langle \mathcal{T}, \mathcal{G} \rangle$ de H ;

répéter

pour chaque $O^k \in \mathcal{O}$ **faire**

Calculer les valeurs des variables $\alpha_t^k(s)$ à l'aide de l'algorithme *forward*;
 Calculer les valeurs des variables $\beta_t^k(s)$ à l'aide de l'algorithme *backward*;
 Calculer les valeurs des variables $\xi_t^k(s, s')$ à l'aide de la formule (2.16);
 Calculer les valeurs des variables $\gamma_t^k(s)$ à l'aide de la formule (2.17);
 Calculer et stocker les valeurs des expressions (2.18), (2.19) et (2.20);

Réestimer les paramètres λ de H en utilisant les formules (2.21) et (2.22);

jusqu'à critère d'arrêt;

2.4.2 Apprentissage quand la structure est inconnue

On a vu à la partie précédente les algorithmes d'entraînement classiques à partir d'une structure connue. Pour certaines applications, néanmoins, aucune connaissance ne permet d'inférer naturellement cette structure (on en verra par exemple un cas dans la partie II de ce document). D'autre part, comme le soulignent Freitag et McCallum [Freitag et McCallum, 2000], la construction "à la main" de HMMs n'est pas aisée. Enfin, il est à noter que l'intuition humaine ne correspond pas toujours à la structure qui tire le