MODULE MESEN

Méthodes Structurelles et Neuronales

C. Faure

A. Grumbach

L. Likforman-Sulem

M. Sigelle

Brique RDF - Reconnaissance des Formes

Ecole nationale supérieure des télécommunications

édition 2005-2006

AVERTISSEMENT

Ce document constitue le support de cours du module Mesen. Il consiste en une présentation des méthodes structurelles (chaînes, graphes, grammaires), markoviennes et neuronales. Ce support de cours vient en complément du polycopié du module ADES présentant le volet statistique de la reconnaissance des formes.

Ce document est organisé de la façon suivante. Les chapitres 1, 2 et 3 sont consacrés aux méthodes structurelles. Celles ci représentent les formes non plus sous forme de vecteur comme dans l'approche statistique mais sous une forme structurée et/ou séquentielle dont la représentation la plus simple est la chaîne. D'autres représentations sous forme d'arbre ou de graphe appartiennent aussi à cette catégorie. Les méthodes de décision associées sont notamment la comparaison de ces structures par programmation dynamique ou leur acceptation par un automate représentatif d'une grammaire de formes. Le chapitre 4 est consacré aux méthodes Markoviennes : chaînes de Markov simples et cachées. Ces méthodes s'appliquent à des formes dont on observe les caractéristiques séquentiellement dans le temps ou dans l'espace. L'apprentissage et la décision font appel à la théorie des probabilités pour estimer les paramètres des modèles représentés sous forme d'automates probabilistes. Enfin, le chapitre 5 est consacré aux méthodes à base de réseaux neuronaux dont les représentants les plus connus sont le perceptron et les réseaux multi-couches. Les règles d'apprentissage basées sur des algorithmes de descente de gradient calculent les poids des connexions entre les différentes couches du réseau.

Laurence LIKFORMAN-SULEM

coordonnateur de la brique RDF

TABLE DES MATIERES

CHAPITRE 1 : INTRODUCTION AUX METHODES STRUCTURELLES DE LA RECONNAISSANCE DES FORMES

Claudie Faure

La reconnaissance de formes est un ensemble de méthodes dont le but essentiel est d'évaluer la ressemblance :

- entre deux formes, la forme de référence et la forme inconnue (on parle alors d'appariement, en anglais : *pattern matching*),
- une forme inconnue et une classe de formes de référence, la valeur de la ressemblance permet de décider si la forme inconnue possède les caractéristiques de la classe et peut être considérée comme un de ses exemplaires,
- une forme inconnue à un ensemble de classes, la forme inconnue est assignée à la classe pour laquelle la ressemblance est la plus forte.

La diversité des méthodes s'explique par la recherche de performances toujours meilleures (!) et le fait que les informations et les modes de décision pertinents pour résoudre un problème de RdF sont très variés.

Les méthodes orientées vers un fort pouvoir d'explicitation des informations de structure peuvent se regrouper sous la rubrique de méthodes structurelles. Elles englobent les méthodes syntaxiques, ancrées dans les langages formels, elles peuvent aussi prendre le nom de méthodes descriptives, voire même linguistiques. La notion de structure, quoique sujette à de nombreuses définitions, peut se ramener à l'existence d'un tout (la structure, c'est-à-dire la forme) décomposable en parties et de relations entre ces parties. Les méthodes structurelles faisant explicitement apparaître les parties qui composent la forme et leurs relations sont particulièrement appropriées pour traiter des structures visuelles où les composantes sont liées par des relations spatiales.

Du point de vue des formalismes de représentation, les méthodes structurelles adoptent préférentiellement des représentations de type : séquence de symboles, graphe ou grammaire. Les symboles des séquences, les nœuds des graphes et le vocabulaire des grammaires sont associés à des composantes de la forme. Les relations entre ces composantes sont explicitées par l'ordre des symboles dans les séquences, les arcs des graphes et les règles de production des grammaires.

Les avantages des méthodes structurelles portent tout d'abord sur l'explication des informations de structure, avec comme conséquence de pouvoir intégrer directement des connaissances, fournies par des experts ou les conventions d'un domaine, dans les formalismes de description (ce qui est particulièrement utile pour traiter des signaux du biomédical ou des dessins techniques) et de pouvoir construire des références (formes ou classes) avec peu d'exemples. Ces avantages compensent leur faiblesse du point de vue de

l'apprentissage à partir d'exemples, tâche pour laquelle les méthodes statistiques sont mieux armées. L'apprentissage des informations de structure est néanmoins un problème général qui a conduit à des méthodes hybrides exploitant le pouvoir d'apprentissage des méthodes statistiques et la représentation explicite des informations de structure. Les hybridations les plus remarquables sont les méthodes qui se rattachent aux modèles markoviens, et certains réseaux connexionnistes où l'architecture a priori du réseau est définie pour permettre des étapes d'extraction d'informations locales spécifiques et de combinaison de ces informations. Le cours donnera d'autres exemples de processus d'apprentissage et de reconnaissance hybrides.

L'adéquation des méthodes structurelles à traiter des données visuelles n'a de sens que si les informations explicitées dans les représentations sont pertinentes pour mesurer des ressemblances de formes. La question du choix de représentations pertinentes est à l'origine de nombreux travaux qui visent à représenter les données brutes issues des capteurs de manière à éliminer les informations parasites (celles qui ne contribuent pas à résoudre le problème de RdF) des informations significatives. Une grande partie du cours est consacrée à la construction de ces représentations.

Ce cours a pour objectif d'introduire la reconnaissance structurelle, il a aussi pour objectif :

- d'apporter un complément à d'autres méthodes de la RdF, en particulier pour ce qui concerne l'extraction automatique des caractéristiques des formes qui peuvent alimenter divers formalismes de représentation des formes,
- de présenter des formalismes et des algorithmes qui sont utilisés dans plusieurs domaines, comme la comparaison de séquences de symboles ou les heuristiques de résolution de problème,
- d'aborder la reconnaissance de formes en termes de processus de traitement ce qui la rapproche de l'intelligence artificielle pour ce qui concerne les stratégies d'analyse et l'usage de connaissances.

Ce polycopié est en construction, il accompagnera le cours en ligne. Dans son état actuel, il présente les représentations à base de chaînes et les méthodes syntaxiques.

CHAPITRE 2 : REPRESENTATIONS A BASE DE CHAÎNES

Laurence Likforman-Sulem

1. Présentation

Nous présentons dans ce chapitre, les méthodes de représentation et de reconnaissance basée sur les chaînes de symboles. Ces méthodes appartiennent à la famille des méthodes structurelles de reconnaissance. Dans ce cadre, une forme est représentée par une chaîne ordonnée de symboles. Cette suite est appelée « phrase » dans la terminologie de la théorie du langage. Les symboles utilisés définissent le codage. En pratique, ces symboles représentent des primitives composant la forme ou des directions le long du contour (codage de Freeman) d'une forme bidimensionnelle.
Pour reconnaître une forme, on la compare à des formes prototypes, par des algorithmes de comparaison de chaînes. Ces algorithmes donnent généralement une mesure de ressemblance (la distance) entre la forme testée et les formes prototypes : distance de Levenshtein notamment. On peut aussi chercher l'inclusion de la forme testée dans les formes prototypes (algorithme de Pratt)

Sur l'exemple ci dessous, nous voulons représenter une courbe par une suite ordonnée de symboles. Il faut donc coder la forme par un ensemble de symboles, appelé alphabet : {0,1,2,3,4,5,6,7}. Chaque symbole de cet alphabet représente une direction dans un espace discret. Ce codage est le codage de Freeman. La courbe est alors représentée par la phrase « 10701112 » composée des lettres de cet alphabet.

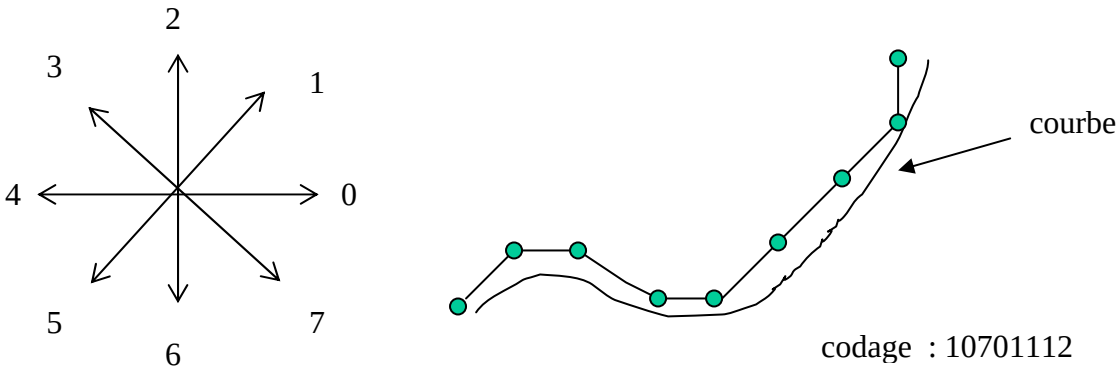


Figure 1 : codage de Freeman et codage d'une courbe mono-dimensionnelle

2. Structures de chaînes : définitions

Revenons sur la définition des chaînes qui se base sur la théorie du langage. Cette théorie partage certains de ses fondements avec ceux des grammaires formelles (cf. chapitre 2) : alphabet , langage, phrase.

• **Alphabet**: ensemble fini X d'éléments appelés lettres, notés a, b, c...

• **Phrase** (sur X): suite ordonnée d'éléments de X représentée par une simple juxtaposition (concaténation) des éléments.

Exemple: alphabet $X = \{a,b,c\}$
phrase sur X: $x = bcaab$

• **Longueur d'une phrase x:** nombre d'éléments qu'elle comporte, notée $|x|$.

• **Phrase vide** notée λ: suite de lettres de longueur nulle.

• **Notation** $x = aaa....a = a^n$

• **L'ensemble des phrases** sur X est noté X^*, **l'ensemble des phrases non vides** X^+.

• y est une **sous-phrase** de $x \in X^*$ si $\exists\ u \in X^*$ et $v \in X^*$, $x = u\ y\ v$
u est le préfixe, v le suffixe.

• opération de **concaténation**

$x, y \in X^*$ avec $x = x_1.....x_n$
$y = y_1.....y_m$

$z = xy = x_1.....x_n y_1.....y_m \in X^*$

La concaténation est une opération interne, associative, non commutative dont l'élément neutre est λ.

3. Comparaison de chaînes

La reconnaissance consiste à comparer une forme test avec une ou plusieurs formes prototype décrites dans le même formalisme de chaîne.
Plusieurs algorithmes de comparaison existent : recherche de sous-chaînes, distance entre chaînes (distance de Hamming, distance d'édition, comparaison dynamique).

3.1 Distance de Hamming

Cette distance permet de comparer deux chaînes de même longueur. En effet, la distance de Hamming consiste compter le nombre de différences entre les deux chaînes construites sur l'alphabet X={0,1}.

Par définition :

$$d(x,y) = Card\left\{ i \in [1,n] \mid x_i = y_i \right\}$$

La distance de Hamming peut être calculée facilement par soustraction des deux chaînes.

x = 1 1 0 1 0 0 1
y = 1 1 0 0 1 0 1
x - y = 0 0 0 1 1 0 0 $\Rightarrow d_h(x,y) = 2.$

on peut vérifier que d(x,y)=d(y,x) et que cette distance vérifie l'inégalité triangulaire.

3.2 recherche d'inclusion : algorithme de Morris et Pratt

Problème: une chaîne y est-elle une sous-chaîne de x?

Le but est d'extraire des parties de forme ressemblant à un prototype ("Pattern Matching").
exemple:

x = a b a a b a a b b
y = b a a b $\quad y \subset x$

Algorithme de Morris et Pratt:

Soit $y = b_1.....b_l$ la sous-phrase à trouver dans $x = a_1.....a_n$ avec $l \leq n$.
x et y sont des phrases du même alphabet.

Principe: manipuler un pointeur P dont la valeur indique la longueur du plus long préfixe de y reconnu comme sous-phrase de x.

On considère l'élément a_1 de x.
Si $a_1 = b_1$ alors P = 1 sinon P reste à 0, on introduit a_2.
I- Après avoir introduit $a_1.....a_k$, le préfixe $b_1.....b_j$ de y est identifié.
alors P vaut j.
II- Si $a_{k+1} = b_{j+1}$ alors $P \leftarrow P + 1$ sinon donner à P une valeur $i < j$ telle que:
1) $b_1.....b_i$ est un suffixe de $a_1.....a_k$
2) $b_1.....b_i$ est le plus long préfixe de $b_1.....b_l = y$ à vérifier 1)

A l'issue de II, on retourne à l'étape I avec P=j=i puis on compare b_{i+1} avec a_{k+1} (étape II)

Exemple: on veut comparer des chaînes formées sur l'alphabet {0, 1} en informatique.
Soient les deux chaînes x et y :

x = 0 10 0 1 0 0 1 0 0 1 1

y = 0 0 1 0 0 1 1

On a déjà introduit a1..ak et P=j=6

a_1_____a_k | … 0 1 0 0 1 0 0 1 | 0 0 1 1

b_1_____b_j | … 0 0 1 0 0 1 | 1

On introduit a_{k+1} différent de b_{j+1}. On décale la chaîne b de telle sorte que $b_1 \ldots b_i$ est un suffixe de $a_1..a_k$, (donc aussi de $b_1..b_j$) et préfixe de $b_1..b_l$ (donc aussi de $b_1..b_j$). La chaîne est ici décaléevers la droite pour que P=3 < 6.

a_1_____ a_k |… 0 10 0 1 0 0 1 | 0 0 1 1

b_1_____b_i |… 0 0 1 | 0 0 1 1

L'indice i ne dépend que de y et de j: i = f(j) ⇒ connaissant j, on peut en déduire i par une fonction de choix f, ne dépendant que de y. Par exemple :
si j=5, alors $b_1..b_j$=<u>00</u>1<u>00</u> et on aura i=2 car $b_1..b_i$=00 est le plus grand suffixe de $b_1..b_j$ à être préfixe de $b_1.. b_{\ell}$

On obtient:

j	1	2	3	4	5	6	7
i=f(j)	0	1	0	1	2	3	0

<u>Algorithme de calcul de f:</u>

f(1) ← 0
Pour j = 2 à l
u ← f(j-1)
Tant que $b_j \neq b_{u+1}$ et u > 0 faire u ← f(u)
Si $b_j \neq b_{u+1}$ et u = 0
alors f(j) ← 0
sinon f(j) ← u + 1

Le plus grand suffixe de b_1.....b_j qui soit préfixe de b_1..... b_{ℓ}contient u+1 éléments.

la chaîne y peut alors être représentée par un graphe dont les nœuds sont les valeurs de P et les transitions portent les symboles de y. On rajoute des transitions correspondant à la fonction f, plus une boucle sur l'état 0. La sous-chaîne y sera incluse dans une chaîne x si on peut parcourir le graphe en lisant la chaîne x du nœud initial 0 au nœud final 7.

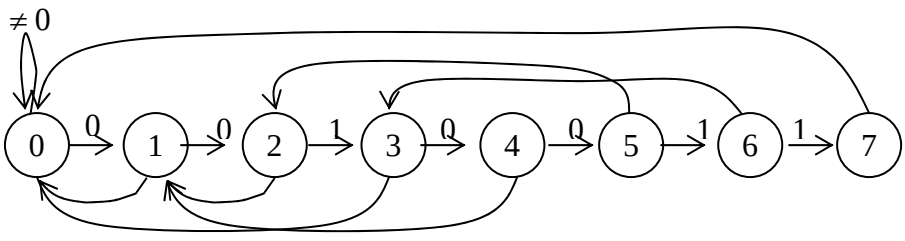
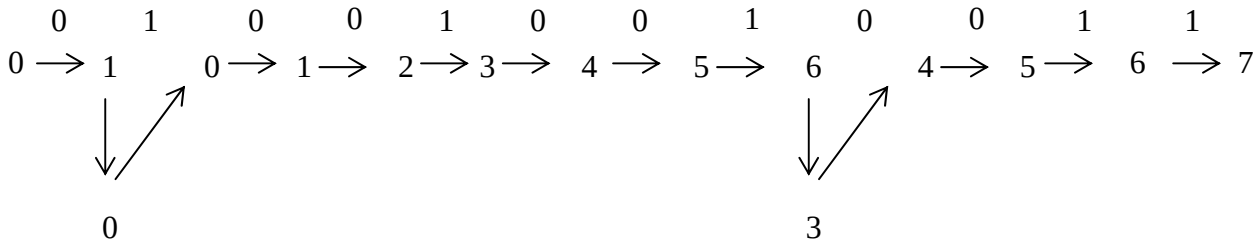


Figure 2 : graphe construit à partir de la chaîne y et de la fonction f

On peut vérifier que y est incluse dans x=10010010011. Quand on lit x, on passe par les nœuds suivants :



4. Distance d'édition

4.1 Définition

La distance d'édition (distance de Levenshtein) entre deux chaînes est la somme cumulée des coûts de transformation d'une chaîne en une autre. Certaines opérations seulement sont autorisées (insertion, substitution, destruction). L'algorithme de Wagner et Fisher permet de calculer cette distance.

Les trois opérations élémentaires possibles pour passer d'une chaîne à l'autre sont associées chacune à un coût:

- Substitution: $a \rightarrow b$ coût $\gamma(a,b)$
- Insertion: $\lambda \rightarrow b$ coût $\gamma(\lambda ,b)$
- Destruction: $a \rightarrow \lambda$ coût $\gamma(a,\lambda)$

Exemple:
Si l'on veut passer de la phrase *x = a b d* en y= *a c b* , une suite de transformations élémentaires peut être :
on détruit d : $(d \rightarrow \lambda)$ on substitue a en c $(a \rightarrow c)$ on insère a : $(\lambda \rightarrow a)$
abd → *ab* → *cb* → *acb*

Le coût de cette transformation est la somme des coûts des transformations élémentaires :
$\gamma(d,\lambda)+ \gamma(a,c) +\gamma(\lambda, a)$

une autre transformation possible est: on insère c : $(\lambda \rightarrow c)$ on détruit d : $(d \rightarrow \lambda)$
acbd → *acb*

Le coût de la transformation est alors : $\gamma(\lambda, c)+ \gamma(d,\lambda)$

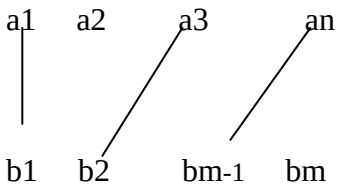
On définit la **distance d'édition** $\delta\ (\ x,\ y)$ comme le coût correspondant à la suite des transformations la moins coûteuse pour passer de x en y.

On introduit les deux contraintes suivantes
- pour les chaînes réduites à des symboles et pour éviter une explosion combinatoire :
  $\delta(\ a,\ b) = \gamma(a,b)$
- inégalité triangulaire pour γ : $\gamma(a,b) + \gamma(b,c) \geq \gamma(a,c)$

Si le terme de « distance » est utilisé, la distance d'édition n'a pas toujours les propriétés d'une distance, en particulier elle n'est pas commutative si $\gamma(\ a,\ b)$ est différent de $\gamma(\ b,a)$.

4.2 Trace

On représente la transformation appelée trace, par un schéma. En voici un exemple :



- substitution entre a_1 et b_1
- destruction de a_2
- substitution entre a_3 et b_2
- substitution entre a_n et b_{m-1}
- .....
- insertion de b_m

La trace utilise des traits correspondants aux opérations de substitution. Les symboles non reliés correspondent suivant le cas à une insertion ou une destruction. On impose les contraintes suivantes à la trace :

- deux liens ne peuvent se croiser,
- deux liens ne partent ou n'arrivent pas à la même lettre.

La dernière condition signifie que l'on ne peut pas fusionner (merge) ou fractionner (split) des symboles (ou primitive). Cette contrainte sera levée dans le cas de la programmation dynamique.

Propriété fondamentale:

Le coût d'une transformation de coût minimal entre deux phrases est égal au coût d'une trace de coût minimal entre ces deux phrases.

4.3 Algorithme de Wagner et Fisher

Soient deux chaînes $X= a_1 .....a_N$, $Y= b_1.....b_M$

On cherche la distance d'édition entre X et Y. Celle-ci est donnée par δ (X,Y)=D(N,M)
D étant un tableau de N+1 lignes et M+1 colonnes où i représente l'indice de ligne et j l'indice de colonne. Ce tableau est rempli séquentiellement de gauche à droite et de haut en bas.

L'algorithme est optimisé pas à pas, c'est à dire pour n'importe quelles sous chaînes.
$D(i, j) = \delta (x(i) , y(j))$ (transformation de $a_1.....a_i$ en $b_1.....b_j$)
avec : $x(i) = a_1 .....a_i$
$y(j) = b_1.....b_j$
Algorithme de Wagner et Fisher:

D(0, 0) = 0
Pour i := 1 à N /* 1ere colonne */
D(i, 0) = D(i - 1, 0) + γ (a_i, λ)
Pour j := 1 à M faire /* 1ere ligne */
D(0, j) = D(0, j-1) + γ (λ, b_j)
Pour i :=1 à N faire
Pour j :=1 à M faire
m_1 = D(i - 1, j - 1) + γ (a_i, b_j) /* substitution */
m_2 = D(i - 1, j) + γ (a_i, λ) /* destruction */
m_3 = D(i , j - 1) + γ (λ, b_j) /* insertion */
D(i, j) = Min(m_1, m_2, m_3)
δ (x, y) = D(N, M) /* distance d'édition*/

La complexité de cette algorithme est en O(NM).

Soit l'exemple simple suivant où tous les coûts sont égaux à 1 : $\gamma(\alpha, \beta) = 1$ pour $\alpha \neq \beta$
X = { a , .., z}
x = h a l l t e i r
y = h a u t e u r

La chaîne X correspond par exemple à une chaîne reconnue par un système de reconnaissance de caractères OCR (Optical Character Recognition), tandis que la chaîne Y correspond à un mot du dictionnaire. On remplit le tableau suivant en commençant par la première ligne, la première colonne. Puis en progressant de la gauche vers la droite, de haut en bas :

chaîne du dictionnaire

X \ Y	λ	h	a	u	t	e	u	r	s
λ	0	1	2	3	4	5	6	7	8
h	1	0	1	2	3	4	5	6	7
a	2	1	0	1	2	3	4	5	6
l	3	2	1	1	2	3	4	5	6
l	4	3	2	2	2	3	4	5	6
t	5	4	3	3	2	3	4	5	6
e	6	5	4	4	3	2	3	4	5
i	7	6	5	5	4	3	3	4	5
r	8	7	6	6	5	4	4	3	4

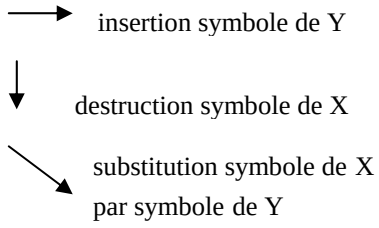
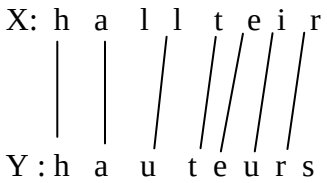


Table I : illustration de l'algorithme de Wagner et Fisher pour le calcul de δ(X,Y)

La distance d'édition est la valeur inscrite dans la dernière case D(N,M) : $\delta(x,y)=3$

Pour connaître la suite de transformations élémentaires correspondant au chemin le moins coûteux, il faut retenir en mémoire pour case (i,j) le passage vers cette case du tableau. Parmi les trois passages possibles : {(i - 1, j - 1), (i , j - 1), (i - 1, j) }un seul correspond au coût le plus faible. Une fois la dernière case calculée, on trouve le chemin optimal par retour-arrière (backtraking).

Il peut y avoir plusieurs chemins optimaux possible. Nous en indiquons un en gras en Table I dont la trace correspondante est :



Les coûts de substitution peuvent être ajustés en fonction de la matrice de confusion de l'Ocr. En effet , le coût $\gamma(i,u)$ sera d'autant plus faible que l'Ocr aura plus tendance à reconnaître les u comme des i.

5. Couplage de formes par Programmation Dynamique

5.1 Présentation

Les méthodes de programmation dynamiques peuvent aussi utiliser des fonctions de déformation (warping function). Ce procédé est très général et permet l'appariement non linéaire entre signaux (parole, courbes, etc…). En effet, les signaux de parole, ou d' écriture (exemple : les signatures) peuvent être sujettes à de variations temporelles ou spatiales, intra-scripteur ou locuteur. Les séquences ou primitives extraites ne sont généralement pas mises en correspondance linéairement, excepté si les signaux sont identiques.

L'objectif de la fonction de déformation est de mettre en correspondance les points de $X=a_1..a_i…a_I$ avec ceux de $Y=b_1…b_j…b_J$. Les contraintes étant pour la courbe de couplage de commencer par C1=(1,1) et de terminer par CK=(I,J). Il ne peut aussi y avoir de retour arrière ni de descente de la courbe. On peut introduire de plus des contraintes pour réduire l'espace de recherche et donc le temps de calcul.

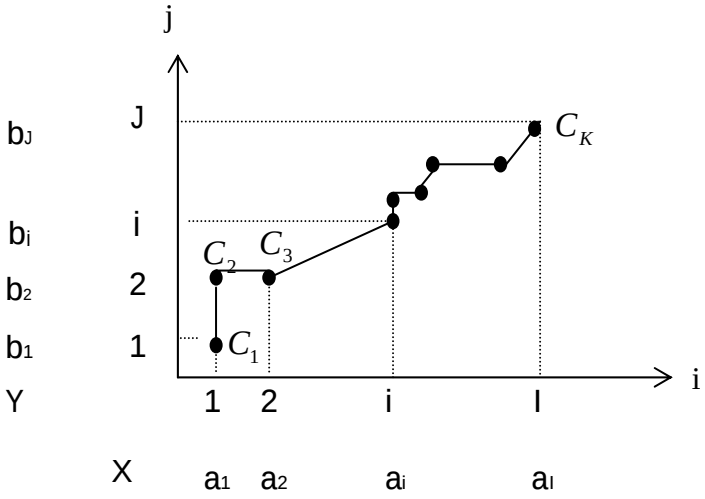


Figure 3 : Courbe de couplage entre deux chaînes X et Y. La courbe est déterminée par l'ensemble des points Ck.

Les possibilités de couplage sont plus souples : possibilités de fusionner ou de scinder des primitives ou symboles.

$$X = \text{a a b c b c}$$
$$\text{Y} = \text{a b b b c c}$$

L'ensemble des points C1, C2, …, Ck,…, CK définissent la fonction de déformation F. avec $C_k=(i(k), j(k))$.
La distance entre les formes X et Y est alors définie par :

$$D(X,Y) = \frac{1}{N} \min_{F} \left[\sum_{k=1}^{K} d(C_k) w(k) \right]$$
$$d(C_k) = d(a_{i(k)}, b_{j(k)}) \ \ et \ \ \ N = I + J$$

Cette formulation comporte un terme de poids w(k) associé à chaque point C_k, et un terme global de normalisation en 1/N qui compense l'effet lié aux longueurs des séquences.

5.2 Algorithme

L'algorithme de calcul de distance est de la forme :

C_1=(1,1) ; D(1,1)=2d(a1,b1)
Pour j :=1 à J
Pour i :=1 à I
$m_1 \leftarrow D(i-1, j) + d(a_i, b_j)$
$m_2 \leftarrow D(i-1, j-1) + 2d(a_i, b_j)$
$m_3 \leftarrow D(i, j-1) + d(a_i, b_j)$
$D(i, j) = \text{Min}(m_1, m_2, m_3)$

$$D(X,Y) = \frac{1}{N} D(I,J)$$

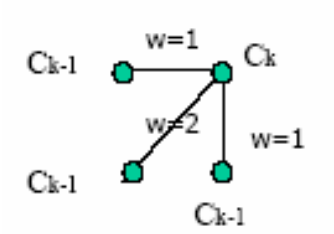


Figure 4 : chemins possibles entre deux points consécutifs de la courbe F

Le tableau D(i,j) se remplit ici ligne par ligne de gauche à droite et de bas en haut. Les poids w ont pour forme w =i(k)-i(k-1)+j(k)-j(k-1). La valeur de w est donc 2 ou 1 suivant la distance entre Ck-1 et Ck.

Une autre forme de l'algorithme modifiant la pente de la courbe F modifie la partie centrale de l'algorithme de la façon suivante [Sakoe]:

$m_1 \leftarrow D(i-1, j-2) + 2d(a_i, b_{j-1}) + d(a_i, b_j)$
$m_2 \leftarrow D(i-1, j-1) + 2d(a_i, b_j)$
$m_3 \leftarrow D((i-2, j-1) + 2d(a_{i-1}, b_j) + d(a_i, b_j)$
$D(i, j) = \text{Min}(m_1, m_2, m_3)$

$$D(X,Y) = \frac{1}{N} D(I,J)$$

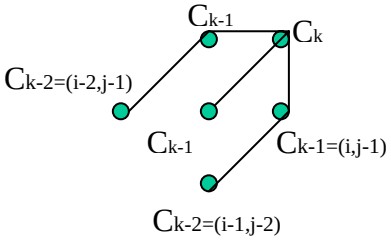


Figure 5 : chemins possibles entre trois points consécutifs de la courbe F

La fenêtre de recherche peut être réduite en imposant les contraintes suivantes sur les indices : $j - r \le i \le j + r$

D’autres formulations de l’algorithme sont possibles :
- on peut introduire des contraintes sur la pente de F [Sakoe]. Les poids w sont modifiés en conséquence.
- on peut introduire des coûts de division d’une primitive en un paire de primitive ou de fusion de paires de primitives en une primitive unique.

Bibliographie:

J. Lopez-Krahe, polycopié de reconnaissance des formes, chapitre 5, 1993.

K.S. Fu "Syntactic Pattern Recognition, Applications", Springer Verlag, New York 1977.

L. Miclet , "Méthodes structurelles pour la reconnaissance des formes"
Eyrolles Collection CNET-ENST, 1984.

A. Belaïd, Y. Belaïd , "Reconnaissance des formes", Inter Editions, 1992.

R.O. Duda, P.E. Hart, Stork , "Pattern Classification and Scene Analysis", Wiley 2003.

H. Sakoe, S. Chiba, Dynamic Programming Algorithm Optimization for spoken word recognition, IEEE trans. Acoustic, Speech and Signal Processing, Vol 26, no 1, 1978.

CHAPITRE 3 : METHODES SYNTAXIQUES

Claudie Faure

1. Méthodes syntaxiques

Les méthodes syntaxiques pour la reconnaissance de formes datent des années 70. Pendant plus de dix ans, ce domaine a été dominé par les travaux de King-Sun FU et de son équipe, il a laissé de nombreuses publications dont deux livres consacrés à ces méthodes [FU 74 ; FU 82]. Elles sont directement issues de la théorie des langages formels.

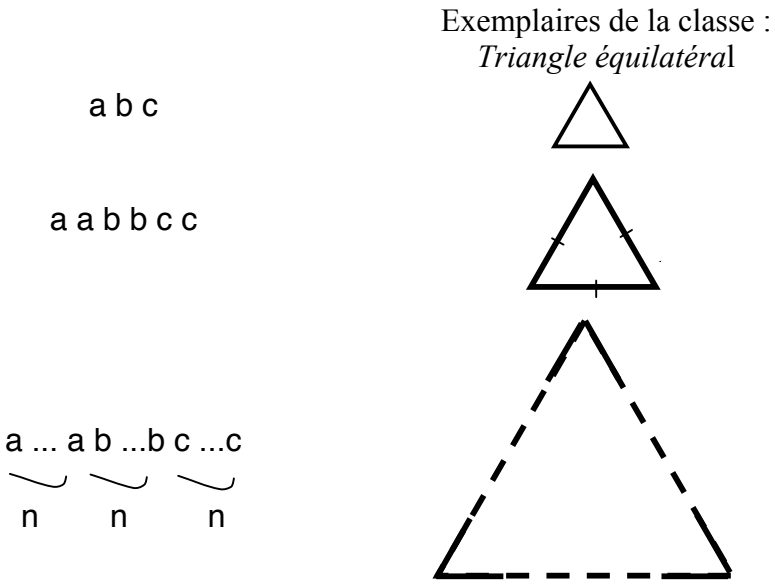
Un langage formel est un ensemble de phrases écrites sur un alphabet de symboles, la structure de ces phrases est déterminée par une grammaire G, soit L(G) le langage qui est engendré par G. Un automate associé à la grammaire G permet d'analyser la syntaxe d'une une phrase x, si x est syntaxiquement correcte, alors x appartient à L(G).

On retrouve dans la définition des langages formels celle des langages de programmation où le vocabulaire est défini à priori ainsi que les règles qui contraignent la syntaxe des "phrases" du code que des automates vérifient au moment de la compilation. Les langages de programmation constituent une facette des langages formels, c'est à la linguistique, autour de Chomsky, que l'on doit des contributions fondamentales. C'est maintenant un domaine de recherche bien établi et cette brève présentation, orientée vers la reconnaissance des formes, n'en donne qu'un très modeste aperçu.

Les méthodes syntaxiques reposent sur une hypothèse forte : les formes qui appartiennent à la même classe présentent une structure commune qui peut être représentée par une grammaire. Un parallèle est fait entre langage formel et *langage de forme* de la manière suivante :

Langage formel	**Langage de forme**
alphabet de symboles	primitives de décomposition
syntaxe du langage / grammaire	structure propre à la classe / grammaire
analyse syntaxique	assignation de la forme à la classe

Exemple	Représentations symboliques	Formes primitives
	a	
	b	
	c	



Le langage de la classe *Triangle équilatéral* : $L_\Delta = \{ x \mid x = a^n b^n c^n, n > 0\}$

❑

L'hypothèse forte qui sous-tend les méthodes syntaxiques permet de plonger le problème de la reconnaissance dans un formalisme rigoureux et d'exploiter des outils d'apprentissage et d'analyse existants. Leur efficacité sera d'autant plus grande que les formes seront conformes à des définitions a priori qui précisent leur structure (c'est le cas des figures géométriques ou de certains des dessins techniques). Les formes "naturelles" seront beaucoup plus difficiles à assimiler à des structures formelles. On verra que les méthodes et formalismes de la théorie des langages vont évoluer pour s'adapter aux problèmes posés par la reconnaissance de formes. Malgré cette limitation, une introduction aux langages formels s'impose ne serait-ce que parce qu'ils interviennent directement dans des problèmes de reconnaissance pour des données textuelles et constituent la référence sous-jacente aux méthodes structurelles.

2. Langage formel

$\mathcal{A}$: ensemble des symboles de l'alphabet, par exemple : $\mathcal{A} = \{ a, b, c \}$

Un mot (ou une phrase) est une séquence finie d'éléments de $\mathcal{A}$, par exemple : x = 'bacaba' où le premier élément est une occurrence de b, le second une occurrence de a ...

λ est le mot vide.

$| x |$ = le degré ou la longueur d'une séquence est le nombre de symboles qui la compose.

$\mathcal{A}^*$ est l'ensemble de toutes les séquences finies de symboles écrites sur $\mathcal{A}$, incluant λ.
$\mathcal{A}^+ = \mathcal{A}^* - \lambda$

La concaténation des séquences x et y est la séquence z = xy

La concaténation est une loi de composition interne de $\mathcal{A}^*$, partout définie et associative, elle n'est pas commutative, λ est son élément neutre. Muni de cette loi, $\mathcal{A}^*$ est un semi-groupe ou monoïde. Les éléments de $\mathcal{A}$ sont les générateurs de $\mathcal{A}^*$.

Définition :

On appelle *langage formel*, défini sur l'ensemble $\mathcal{A}$, tout sous ensemble de $\mathcal{A}^*$.

3. Grammaire formelle

Définition :

Une grammaire G est définie par :

- un symbole de départ : S
- un vocabulaire terminal : V_T
- un vocabulaire auxiliaire : V_N
- des règles de production : P

Le vocabulaire total est noté V, c'est l'union de V_N et V_T.

V* est l'ensemble des mots construits sur le vocabulaire V, incluant le mot vide λ.

Notations :

Les règles de production sont des règles de réécriture.

P : $\alpha \rightarrow \beta$ signifie "α se réécrit β"

Soit V le vocabulaire total d'une grammaire G. Si η et γ sont deux mots de V*, tels que l'on puisse dériver γ à partir de η en appliquant 0, 1 ou plusieurs règles de production, alors on peut écrire :

$$\eta \underset{G}{\overset{*}{\Rightarrow}} \gamma$$

G engendre les séquences de symboles syntaxiquement correctes qui appartiennent au langage L(G).

Les éléments de L(G) sont toutes les séquences x écrites sur l'alphabet terminal V_T qui peuvent être obtenues à partir du symbole de départ S en appliquant 0, 1 ou plusieurs règles de production :

$$L(G) = \{x \text{ t.q. } x \in V_T^* \text{ et } S \underset{G}{\overset{*}{\Rightarrow}} x\}$$

Exemple

Soit G une grammaire définie par :

$V_T = \{a,b\}$ $\quad V_N = \varnothing$

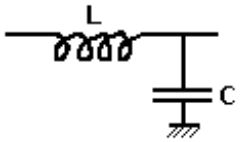
P1: $S \rightarrow aS$ $\quad$ P2 : $S \rightarrow b$

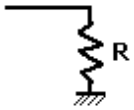
Soit x = aaab, x est dérivé de S par application des règles de production :

$$S \overset{(1)}{\Rightarrow} aS \overset{(1)}{\Rightarrow} aaS \overset{(1)}{\Rightarrow} aaaS \overset{(2)}{\Rightarrow} aaab$$

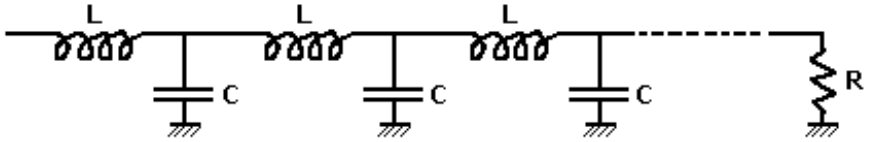
$$S \overset{*}{\underset{G}{\Longrightarrow}} aaab \in L(G)$$

Le langage L(G) est de la forme $\{ x \mid x = a^n b, n \geq 0\}$

Si a représente : 

et b : 

alors L a pour expression graphique :



❑

4. Types de grammaires

Les grammaires sont classées suivant quatre types qui sont déterminés par la forme des règles de production.

Type 0 : Aucune contrainte sur les règles de production.

Type 1 : Grammaire contextuelle

P: $\xi_1 A \xi_2 \rightarrow \xi_1 \beta \xi_2$ $\qquad \beta, \xi_1, \xi_2 \in V^*, \qquad A \in V_N$

Exemple

Soit la grammaire de la classe *Triangle équilatéral* qui engendre le langage :
$L_\Delta = \{ a^n b^n c^n, n \geq 1\}$
$V_T = \{a, b, c\}$ $\qquad V_N = \{ S, A, B, C\}$
P : (1) $S \rightarrow aSBC$
(2) $S \rightarrow aBC$
(3) $CB \rightarrow BC$
(4) $aB \rightarrow ab$
(5) $bB \rightarrow bb$
(6) $bC \rightarrow bc$
(7) $cC \rightarrow cc$

⇨Remarque
La règle 3 est obtenue par :
$CB \rightarrow CC$ ($\xi_1 B \rightarrow \xi_1 C$) et $CC \rightarrow BC$ ($C\xi_2 \rightarrow B\xi_2$)

❑

Type 2 : Grammaire à contexte libre (ou algébrique)

$P : A \rightarrow \beta \qquad A \in V_N, \qquad \beta \in V^*$

Exemple

$L = \{a^n b^n \mid n \geq 1\}$
P : $S \rightarrow aSb$
$S \rightarrow ab$

❑

Type 3 : Grammaire régulière

$P : A \rightarrow aB$ ou $A \rightarrow b \qquad a, b \in V_T, \qquad A \in V_N$

Exemple

$L = \{a^m b^n \mid m, n \geq 1\}$

P :	$S \rightarrow aA$	$B \rightarrow bB$
	$A \rightarrow aA$	$B \rightarrow b$
	$A \rightarrow bB$	$A \rightarrow a$

❑

Il existe une relation d'inclusion entre les langages engendrés par les 4 types de grammaires : un langage engendré par une grammaire de type i peut être engendré par des grammaires de type < i.

Par la suite, nous ne parlerons que des grammaires de type 2 et 3 dans la mesure où elles sont les plus utilisées en RdF mais aussi pour les langages de programmation. Les autres grammaires ont des règles de production qui sont peu contraintes, voire pas du tout, ce qui augmente beaucoup la complexité des analyseurs syntaxiques. En RdF, il est pratiquement impossible de définir (ou d'apprendre) de telles grammaires.

5. Grammaires et automates

Une grammaire engendre un langage L(G). Pour savoir si une séquence de symboles x appartient à L(G), il faut vérifier que x est syntaxiquement correcte. L'analyse syntaxique de x est faite par un automate. A chaque grammaire correspond un automate défini à partir des règles de production de la grammaire. A chaque type de grammaire, est associé un type d'automate.

Pour les grammaires à contexte libre, l'automate associé est de type automate à pile de mémoire. Il ne sera pas décrit ici. Il intervient dans la compilation de langage de programmation comme Pascal ou C. Il existe des outils très utiles qui construisent l'automate associé à une grammaire de type 2. Le plus répandu est YACC (et LEX) qui permet de définir une grammaire par un lexique (le vocabulaire terminal) et des règles de production. L'analyseur est ensuite construit automatiquement. YACC et LEX sont disponibles depuis des

années sous UNIX et sont toujours très utilisés, par exemple, pour définir et analyser les expressions linguistiques qui forment un langage de commande.

Le type d'automate associé aux grammaires régulières est un automate à états finis.

Définition :

Un automate à états finis A est défini par :

$A = (\Sigma, Q, \delta, q_0, F)$

- Σ : vocabulaire d'entrée
- Q : ensemble fini d'états
- δ : applications de $Q \times \Sigma \rightarrow Q$
- q_0 : état initial
- $F \in Q$: ensemble des états finaux

Fonctionnement de l'automate :

L'automate démarre dans l'état initial q_0, les symboles sont lus de la gauche à la droite. Quand un symbole d'entrée est lu, l'automate cherche à appliquer une règle de transition d'états δ. Si une règle est applicable, l'automate change d'état, sinon il se bloque.

Les règles de transition d'états sont de la forme : $\delta (q_i, a) = q_{i+1}$ (l'automate est dans l'état q_i, le symbole a est lu en entrée, alors l'automate passe dans l'état q_{i+1}).

La séquence de symbole est syntaxiquement correcte quand le dernier symbole de la séquence a été lu et que l'automate est dans un état F.

On illustre par un exemple la construction de l'automate à états finis associé à une grammaire régulière :

Soit G, la grammaire régulière qui engendre $L = \{a^n b \mid n \geq 1\}$

$V_T = \{a, b\}$ $\quad V_N = \{A, B\}$

P : $S \rightarrow aA$

$A \rightarrow aA$

$A \rightarrow b$

Soit A l'automate à états finis qui reconnaît les phrases du langage $L = \{a^n b \mid n \geq 1\}$

$\Sigma = V_T = \{a, b\}$

$Q = V_N \cup \{T\}$ $\quad$ {T} = toutes les phrases de L(G)

$q_0 = S$

Si P contient $S \rightarrow \lambda$, alors $F = \{S, T\}$ sinon $F = \{T\}$

Les règles de transitions d'états de l'automate se construisent à partir des règles de production de la grammaire suivant les règles de traduction suivantes :

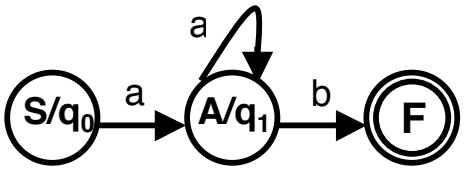
- si P s'écrit : $B \rightarrow a$ avec $B \in V_N$ et $a \in V_T$
  alors la règle de transition associée s'écrit : $\delta(B, a) = T$
- si P s'écrit : $B \rightarrow aC$ avec $B, C \in V_N$ et $a \in V_T$
  alors la règle de transition associée s'écrit : $\delta(B, a) = C$

De plus, on doit avoir : $\delta(T, a) = \varnothing$ $\quad \forall a \in V_T$

Dans cet exemple, F se réduit à un seul état et les transitions sont données par :

$\delta (q_0, a) = q_1$
$\delta (q_1, a) = q_1$
$\delta (q_1, b) = F$

La représentation graphique ci-dessous exprime à la fois les règles de production de la grammaire et le fonctionnement de l'automate :



6. Quelques propriétés des grammaires

Récursivité

Une grammaire G est récursive s'il existe au moins une règle de production de la forme :

$A_i \rightarrow \alpha_i A_i \beta_i \qquad A_i \in V_N \qquad \alpha_i, \beta_i \in V^*$

Equivalence

G_A et G_B sont équivalentes si elles engendrent le même langage, $L(G_A) = L(G_B)$.

Ambiguïté

Si une séquence de symboles appartenant à L(G) peut être dérivée de S par deux suites de règles de production différentes, alors la grammaire G est ambiguë.

7. Grammaire stochastique

Définition

Soit une grammaire stochastique Gs : (S, V_T, V_N, Ps).

Les règles de production sont associées à des probabilités de réécriture p_{ij} :

$$Ps : \alpha_i \xrightarrow{p_{ij}} \beta_{ij} \qquad \alpha_i \in V^* V_N V^*, \beta_{ij} \in V^*$$

Si n_i est le nombre de règles qui ont α_i comme prémisse : $\sum_{j=1}^{n_i} p_{ij=1}$

Soit $x \in L(Gs) : S \overset{*}{\underset{G}{\Rightarrow}} x$, la probabilité de x est le produit des probabilités des règles Ps utilisées pour dériver x à partir de S.

8. Inférence

L'inférence est le nom qui désigne l'apprentissage à partir d'exemples dans les langages formels. Le but est donc, d'utiliser l'information apportée par un ensemble fini d'exemplaires d'une classe pour apprendre ce qui est caractéristique de la classe, par des méthodes de généralisation, et ceci afin de pouvoir reconnaître de nouveaux exemplaires.

Le problème de l'inférence :

> Soit I un échantillon positif sur un alphabet $\mathcal{A}$, c'est-à-dire un ensemble de séquences de symboles qui appartiennent à un langage L(G), G inconnue.
> Comment construire une grammaire G telle que $I \in L(G)$?

Remarque : on peut aussi disposer d'un échantillon négatif de séquences de symboles n'appartenant pas à L(G).

La grammaire solution n'est pas unique, un ensemble fini de phrases ne définit pas de manière unique un langage, il appartient à une infinité de langages. Pour un échantillon d'inférence I = {aabb, aaaabbbb, aaaaaabbbbbb}, on peut donner quelques exemples de langages engendrés par des grammaires telles que : $I \in L(G)$:

$L_1 = \{x \mid x=a^{2n}b^{2n}, n>0\}$ avec $V_T = \{a, b\}$
$L_2 = \{x \mid x=a^{n}b^{n}, n>0\}$ avec $V_T = \{a, b\}$
$L_3 = \{x \mid x=a^{m}b^{n}, m, n>0\}$ avec $V_T = \{a, b\}$
$L_4 = \{x \mid x=g^{m}h^{n}, m, n>0\}$ avec $V_T = \{a, b, c, d\}$ et $g, h \in V_T$
etc.

Afin d'éliminer des grammaires peu intéressantes, on restreint l'espace des solutions en imposant que I soit structurellement complet.

Définition

I est structurellement complet si :

- G est solution du problème d'inférence, $I \in L(G)$
- l'alphabet $\mathcal{A}$ est l'ensemble des terminaux de G ($\mathcal{A} = V_T$)
- toutes les règles de P sont utilisées au moins une fois pour engendrer les éléments de I.

Les méthodes d'inférence grammaticale développées dans le domaine des langages formels sont souvent inadéquates pour apprendre des structures de formes complexes mais elles peuvent être utiles pour des formes très contraintes, comme les figures géométriques, ou des parties de formes dont la structure est plus simple que celle de la forme globale.

On considère le cas des grammaires régulières pour introduire des définitions et le principe de base de l'inférence.

9. Inférence des grammaires régulières

Définition

On appelle grammaire universelle, la grammaire G_u qui peut engendrer tous les mots écrits sur un vocabulaire terminal V_T.

$L(G_u) = V_T^*$

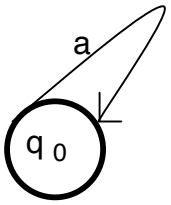
Elle est associée à l'automate universel A_u :

$|Q| = 1$

$Q = q_0$

$F = q_0$

$\forall\ a \in \Sigma,\ \delta(q_0, a) = q_0$



Définition

Soit I un ensemble fini de mots écrits sur l'alphabet V_T, la grammaire canonique G_c est telle que : $L(G_c) = I$.

Il existe plusieurs grammaires canoniques correspondant à I. La grammaire canonique maximale est construite de la manière suivante :

$I = \{x_1, ..., x_N\}$

$x_i = a_{i1}, ..., a_{il_i}$

$S \rightarrow a_{i1}A_{i1}$

$A_{i1} \rightarrow a_{i2}A_{i2}$

.

.

.

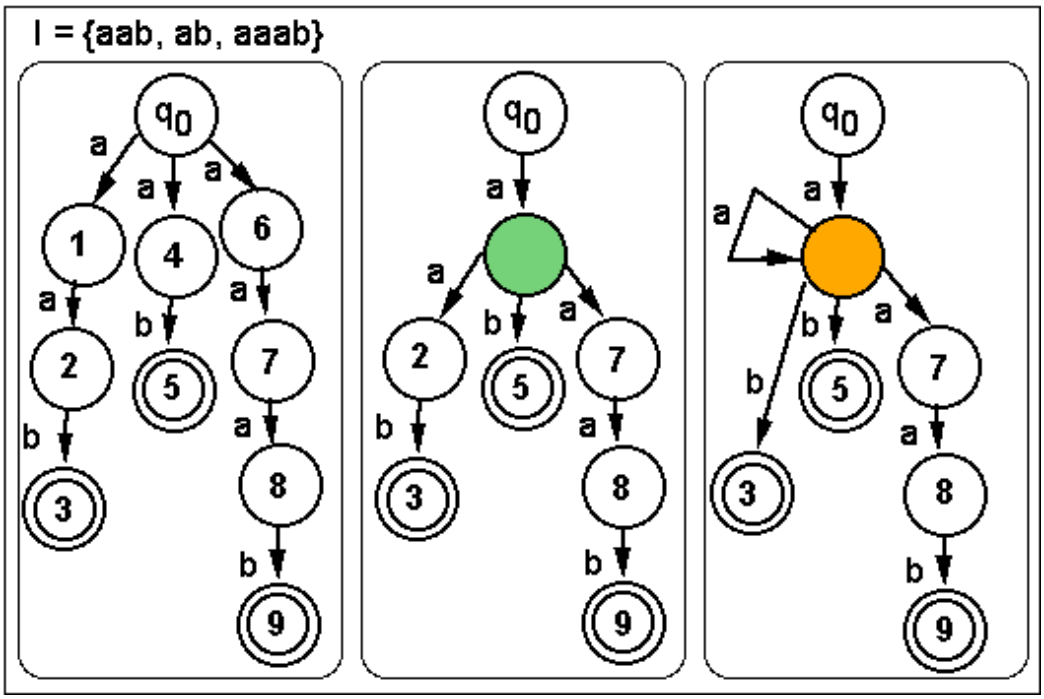
$A_{il_{i-1}} \rightarrow a_{il_i}$

même chose pour tous les mots x_i.

La grammaire canonique maximale est représentée graphiquement par la figure (a) de l'exemple suivant. On peut obtenir une grammaire canonique plus petite en fusionnant des règles de production, ce qui graphiquement revient à grouper des états, voir la figure (b). Le groupement d'états permet de produire des grammaires qui engendrent toujours I mais aussi d'autres mots, comme le montre la figure (c) de l'exemple.

Le groupement de plusieurs états en un seul état se fait en transformant les arcs entrant et sortant des états à regrouper en arcs entrant et sortant du nouvel état ou en transitions internes au nouvel état. La fusion de l'état grisé de la figure (b) avec l'état 2 conduit à une transition interne à l'état grisé de la figure (c) qui se produit à la lecture du symbole 'a'. Cette fusion permet d'accepter un nombre quelconque de 'a' à la suite du premier 'a', ce qui augmente le nombre de mots engendrés par la grammaire.

Exemple



a b c

❑

L'inférence repose sur le regroupement d'états. Le problème de l'inférence se ramène à trouver une partition de l'ensemble des états de la grammaire canonique et à associer un seul état à chaque élément de la partition. Il existe plusieurs méthodes, qui ne sont pas présentées ici, dont certaines sont décrites de manière très pédagogique dans [Miclet 84] et [Cornuéjols, Miclet 02].

10. L'inférence de formes

On donne ici un exemple qui est caractéristique de la manière dont la RdF s'est appropriée le problème de l'inférence. Cet exemple est celui des *Pattern Gramma*rs de Evans []. L'idée est de construire une grammaire de description de configurations spatiales à l'aide de règles heuristiques.

Pattern Grammar :

- un ensemble de primitives,
- un ensemble de prédicats qui décrivent les relations entre primitives et de conditions de validité,
- un ensemble de règles de production.

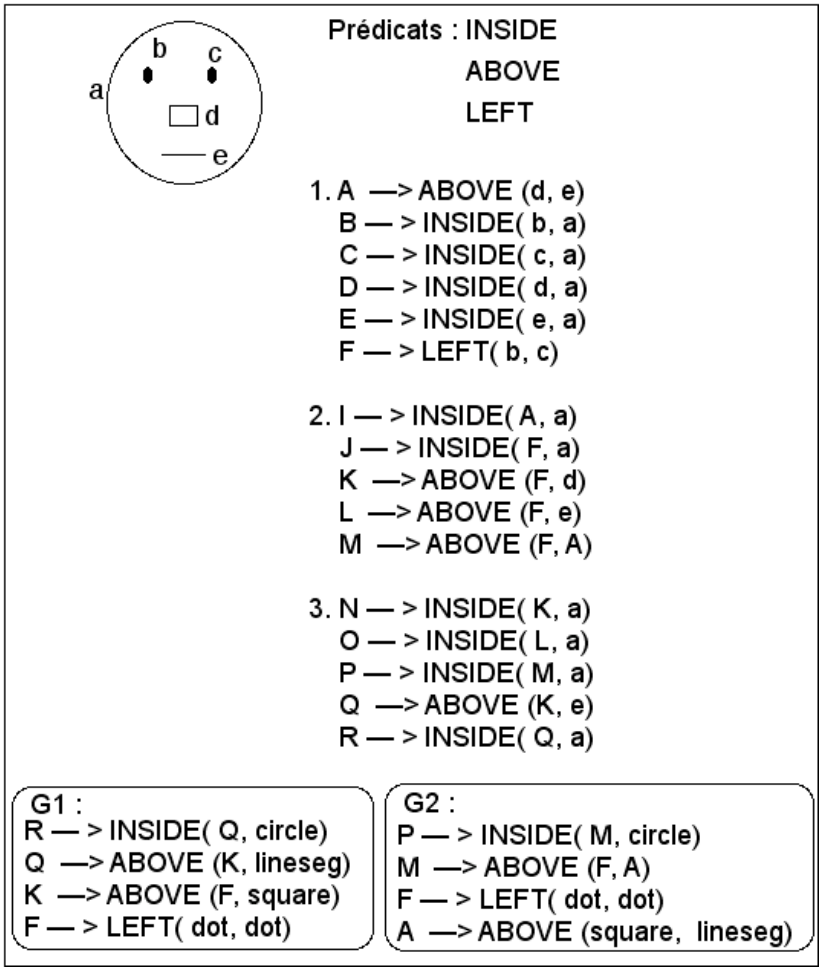
Méthode :

- Analyser la liste courante des composantes de la configuration.
- Trouver les couples de composantes qui vérifient les prédicats. Créer une nouvelle composante et une règle de production si le prédicat est vérifié.
- Quand aucun prédicat ne peut plus être vérifié sur la liste courante, mettre à jour la liste des composantes en incluant celles qui ont été créées.

Succès :

Plus de création d'éléments possible et toutes les parties constituantes de la configuration apparaissent au moins dans une règle de production.

Exemple



❑

11. Analyse syntaxique

Soit G (S, V_T, V_N, P) et x une séquence de symboles.

Si et seulement si x appartient à L(G) alors il est possible de construire l'arbre de dérivation.

Un arbre de dérivation a pour sommet le symbole de départ S de G, les feuilles sont des symboles de V_T et les nœuds intermédiaires des symboles de V_N.

L'arbre peut se construire de la racine aux feuilles (analyse descendante) ou des feuilles à la racine (analyse ascendante). Les méthodes de construction ne sont pas décrites ici.

Exemple

Règles de production :

$S \xrightarrow{1} T$ $\quad T \xrightarrow{3} T\ x\ |$ $\quad | \xrightarrow{5} a$ $\quad | \xrightarrow{7} c$

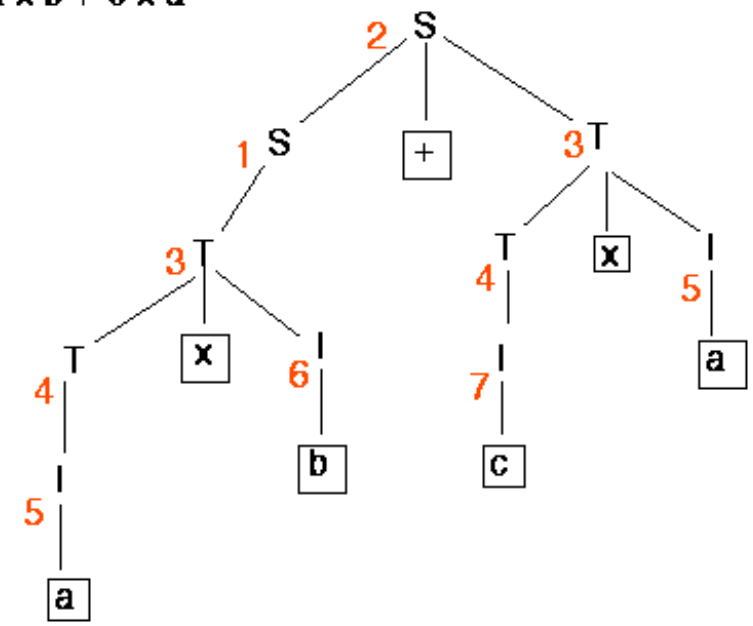
$S \xrightarrow{2} S + T$ $\quad T \xrightarrow{4} |$ $\quad | \xrightarrow{6} b$

Soit la phrase : x = 'a x b + c x a'

$S \overset{2}{\Rightarrow} S + T \overset{1}{\Rightarrow} T + T \overset{3}{\Rightarrow} T\ x\ | + T \overset{4}{\Rightarrow} |\ x\ | + T \overset{3}{\Rightarrow} |\ x\ T + T\ x\ |$

$\overset{4}{\Rightarrow} |\ x\ | + |\ x\ | \overset{5}{\Rightarrow} a\ x\ | + |\ x\ | \overset{6}{\Rightarrow} a\ x\ b + |\ x\ | \overset{7}{\Rightarrow} a\ x\ b + c\ x\ |$

$\overset{5}{\Rightarrow} a\ x\ b + c\ x\ a$



❑

12. Analyseurs tolérants

On montre ici le type de compromis entre grammaire formelle et reconnaissance des formes. Contrairement aux séquences de symboles des langages formels, les représentations des formes sont souvent bruitées. On peut limiter ce bruit par des prétraitements mais il est pratiquement impossible d'éliminer toutes les petites variations locales non significatives. Les analyseurs classiques rejettent systématiquement les séquences qui ne sont pas syntaxiquement correctes. La RdF a apporté sa contribution à l'analyse syntaxique par des analyseurs tolérants dans le but d'accepter des séquences de symboles qui ne sont pas syntaxiquement correctes mais qui restent proches des mots du langage engendré par la grammaire.

L'analyseur de Wagner intègre l'algorithme de comparaison de séquences de symboles pour rechercher le mot de L(G) le plus proche de la séquence à analyser. La distance entre ce mot de L(G) et la séquence permettra de décider si cette séquence appartient à L(G).

Fonctionnement

α : phrase à analyser : $\alpha_1\ \alpha_2 \ldots \alpha_n$

A : automate à états finis qui accepte L(G)

On introduit F(j, S) = distance d'édition minimale entre $\alpha_1\ \alpha_2 \ldots \alpha_j$ et β

β fait évoluer l'automate A de l'état initial à l'état S.

Quand j = n et S = un état final de A, F donne la distance minimale entre la séquence α et un mot de L(G).

Soit $\mathcal{F}$ l'ensemble des états finaux :

$F(|\alpha|, R) = \underset{S\in\mathcal{F}}{\text{Min}}\ F(|\alpha|, S)$ = distance d'édition minimale entre α et L(G).

Le calcul de $F(|\alpha|, R)$ se fait par programmation dynamique.

$$F(j, S) = \underset{T}{\text{Min}}\{F(j-1, T) + V(T, S, \alpha_j)\}$$

V (T,S, αj) : nombre minimum d'opérations d'édition qui changent le caractère αj en un mot w(T,S) qui fait évoluer l'automate A de l'état T à l'état S.

V (T,S, αj) = P(T,S) - L(T,S, αj)

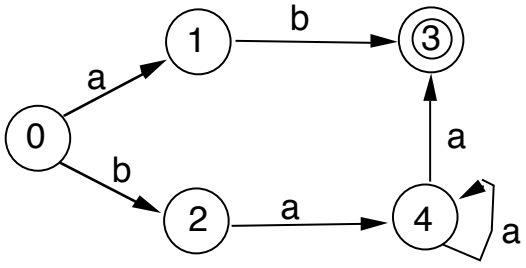
- P(T,S) = nombre d'arcs minimum entre T et S,
  si il n'y a pas de chemin entre T et S alors : P(T,S) = 0 et V (T,S, αj) =1
- L(T,S, αj) = 1 si αj apparaît au moins une fois sur le chemin le plus court entre T et S, sinon L(T,S, αj) = 0.

Exemple

On doit calculer :

$$F(j, S) = \underset{T}{\text{Min}}\{F(j-1, T) + V(T, S, \alpha_j)\}$$

ce qui nécessite le calcul du coût des transitions entre chaque état T et l'état S quand le symbole lu est αj. On donne des exemples de calculs.



F(0,S) = 0 si S est l'état initial

∞ autrement

cas : $\alpha j = b$, T=1, S=3

$F(j-1,T) + V(T,S,\alpha j) = F(j-1,1) + V(1,3,b)$
$= F(j-1,1) + P(1,3) - L(1,3,b)$
$= F(j-1,1) + 1 - 1 \Rightarrow V=0$ coût = $\gamma(b,b) = 0$

cas : $\alpha j = a$, T=1, S=3

$F(j-1,T) + V(T,S,\alpha j) = F(j-1,1) + V(1,3,a)$
$= F(j-1,1) + P(1,3) - L(1,3,a)$
$= F(j-1,1) + 1 - 0 \Rightarrow V=1$ coût = $\gamma(b,a)=1$

cas : $\alpha j = a$, T=1, S=4

$F(j-1,T) + V(T,S,\alpha j) = F(j-1,1) + V(1,4,b)$
$= F(j-1,1) + P(1,4) - L(1,4,a)$
$= F(j-1,1) + (0) \Rightarrow V=1$ coût = $\gamma(\lambda,a) = 1$

cas : $\alpha j = a$, T=0, S=3

$F(j-1,T) + V(T,S,\alpha j) = F(j-1,0) + V(0,3,b)$
$= F(j-1,0) + P(0,3) - L(0,3,a)$
$= F(j-1,0) + 2 - 1 \Rightarrow V=1$ coût = $\gamma(b,\lambda) = 1$

❑

Une autre façon de tolérer des erreurs syntaxiques est la solution proposée par Thomasson []. Les opérations d'édition sont introduites au niveau des règles de production de la grammaire initiale G, il en résulte une augmentation du nombre de règles. La grammaire augmentée G' définit un langage augmenté L(G'), l'analyseur associé à G' vérifie la correction syntaxique stricte. Les poids sur les règles de production renseignent sur le nombre d'erreurs d'édition que comporte la séquence analysée.

Les opérations sur les règles de production :

- CHn : substitution de n symboles terminaux
- DEn : élimination de n symboles terminaux
- INn : insertion de n symboles terminaux

Exemple

$V_T = \{ a, b\}$

P dans G	Opérations (avec coûts associés)	P dans G'
A —> aBb	CH1	A —> bBb
		A —> aBa
	CH2	A —> bBa
	DE1	A —> Bb
		A —> aB
	DE2	A —> B
	IN1	A —> aaBb
		A —> baBb
		A —> abBb
		A —> aBbb

A —> aBba
A —> aBab
A —> aaBb

etc.

❑

13. Réseau de transitions augmentées

Les automates à états finis présentent l'avantage d'être faciles à programmer et à visualiser graphiquement. Malheureusement, ils restent limités à l'analyse des structures définies avec peu d'information contextuelle. Ils ne peuvent pas faire une analyse syntaxique de $L=\{a^n b^n, n>0\}$ où le nombre de 'b' doit être égal au nombre de 'a'. L'absence de mémoire des automates à états finis est une limitation. Afin de préserver les avantages de ces automates et d'étendre leur domaine d'utilisation, plusieurs propositions ont été faites pour les adapter aux besoins de la reconnaissance. On décrit ici les réseaux de transitions augmentées (plus connus sous le nom de ATN) que Woods [1970] a défini pour la reconnaissance du langage naturel. Cet exemple illustre un des premiers travaux sur la construction d'analyseurs ancrés sur les automates à états finis, il est une référence pour les travaux qui ont suivi.

Définition

ATN : ensemble fini d'automates à états finis et de mémoires.
Cinq catégories d'arcs :

- CAT c : la transition sur l'état fin d'arc est réalisée si 'c' est lu + HOLD qui mémorise 'c' dans une liste.
- PUSH (q'_0), $q'_0 \in Q_0$: l'état fin d'arc est mémorisé dans une pile, transition sur q'_0.
- POP : la transition se fait sur l'état en tête de pile qui est éliminé de la pile. Si pile vide alors : état final.
- VIR c : transition sur l'état fin d'arc si 'c' est dans la liste 'HOLD'.
- JUMP : transition sur l'état fin d'arc si les conditions spécifiées sur l'arc sont vérifiées, aucun symbole lu en entrée.

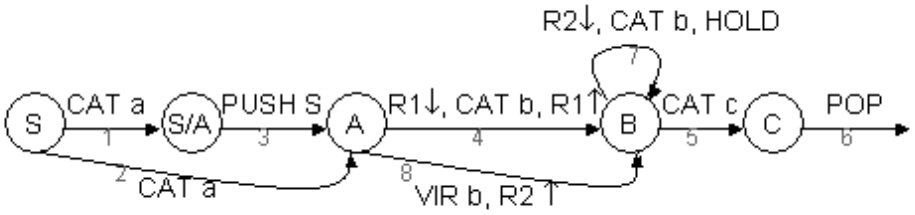
Exemple

Soit le langage L : $\{a^n b^n c^n, n \geq 1\}$ et sa grammaire G (S, V_T, V_N, P) :
$V_T = \{a, b, c\}$ $V_N = \{S, B, C\}$

P :	$S \rightarrow aBC$	$aB \rightarrow ab$	$bC \rightarrow bc$
	$S \rightarrow aSBC$	$bB \rightarrow bb$	$CB \rightarrow BC$

et un mot x de à L à analyser : x = "aaabbbccc"

L'analyse est faite avec un ATN représenté graphiquement ci-dessous. Deux registres R1 et R2 permettent d'ajouter des conditions aux transitions :
la transition 4 est possible si R1 = ↓ , R1 est alors mis à ↑,
la transition 7 est possible si R2 = ↓
la transition 8 met R2 à ↑.



Transitions	Symboles	Registres et piles
1	a	
3		A
1	a	
3		A A
2	a	
4	b	R1↑
7	b	(b)
7	b	(bb)

Transitions	Symboles	Registres et piles
5	c	
6	→ A	A
8	→ B	R2↑ (b)
5	c	
6	→ A	⊓
8	→ B	0
5	c	
6	FIN	

❑

14. Grammaire programmable

Il existe d'autres grammaires qui n'entrent pas dans les quatre types définis par la théorie des langages formels. Les grammaires programmables en font parti.

Définition

$Gp = (V_N, V_T, J, P, S)$

Les règles de production sont de la forme :

P : (r) $\alpha \rightarrow \beta$ $U \subset J$ $W \subset J$

Etiquette Cœur Succès Echecs

$r \in J$, $\alpha \in V^*V_NV^*$, $\beta \in V^*$

Fonctionnement de l'analyseur

Départ : la règle 1 est appliquée

En général, la règle ($r : \alpha \rightarrow \beta$) est applicable si la séquence courante contient α.

- (r) est appliquée
- la règle suivante à appliquer est dans les succès U de (r),
- s'il n'y a pas de règle applicable dans U alors la règle suivante est dans les échecs W,
- si W est vide, et qu'il n'y a pas de règle applicable dans U, alors la dérivation s'arrête.

15. Syntaxe et sémantique

On a déjà vu l'effet de l'introduction d'information sémantique à propos des symboles attribués. Sur les grammaires, la sémantique peut être utile pour diminuer la complexité syntaxique en réduisant le nombre de règles de production. Les règles ont alors une partie syntaxique et une partie sémantique, ce qui nécessite bien évidemment des analyseurs adaptés. L'exemple suivant montre la diminution du nombre de règles de production lorsque la grammaire est augmentée de sémantique.

Exemple

Soit G_1 une grammaire de type 2 qui engendre le langage :

$$L = \{a^{2^n} \mid n \geq 1\}$$

$G_1 = (V_N, V_T, P, S)$ $\quad V_N = \{S, A, B, C, D, E\}$ $\quad V_T = \{a\}$

P $\quad 1 : S \rightarrow ACaB \quad 2 : Ca \rightarrow aaC \quad 3 : CB \rightarrow DB \quad 4 : CB \rightarrow E$

$\quad 5 : aD \rightarrow Da \quad 6 : AD \rightarrow AC \quad 7 : aE \rightarrow Ea \quad 8 : AE \rightarrow \lambda$

La dérivation de a^{2^2} à partir de S :

$$S \overset{1}{\Rightarrow} ACaB \overset{2}{\Rightarrow} AaaCB \overset{3}{\Rightarrow} AaaDB \overset{5}{\Rightarrow} AaDaB \overset{5}{\Rightarrow} ADaaB \overset{6}{\Rightarrow} ACaaB \overset{2}{\Rightarrow} AaaCaB \overset{2}{\Rightarrow}$$

$$AaaaaCB \overset{4}{\Rightarrow} AaaaaE \overset{7}{\Rightarrow} AaaaEa \overset{7}{\Rightarrow} AaaEaa \overset{7}{\Rightarrow} AaEaaa \overset{7}{\Rightarrow} AEaaaa \overset{8}{\Rightarrow} aaaa = a^{2^2}$$

Soit G_2 la grammaire de type 2 dont les règles ont été augmentées d'une partie sémantique.

On pose : $A(a) = 1$ (la longueur de la primitive 'a' est égale à 1).

P :	Partie syntaxique	Partie sémantique
	1 : S → AA	$A(S) = A(A) + A(A)$
	2 : A → a	$A(A) = A(a) = 1$
	3 : A → BB	$A(A) = A(B) + A(B)$
	4 : B → A	$A(B) = A(A)$

G_2 engendre $L = \{a^{2^n} \mid n \geq 1\}$ si $A(S) = 2^n$

On peut imposer la condition $A(S) = 2^n$ en implémentant G_2 sous forme d'une grammaire programmable G_{2p} :

Etiquette	Cœur	Succès	Echec
1	S → AA	3	∅
2	A → a	2	∅
3	A → BB	3	4
4	B → A	4	2, 3

G_2 peut engendrer des mots qui ne sont pas dans L, mais pas G_{2p} :

$$G_2 : S \overset{1}{\Rightarrow} AA \overset{3}{\Rightarrow} BBA \overset{4}{\Rightarrow} ABA \overset{4}{\Rightarrow} AAA \overset{2*}{\Rightarrow} aaa$$

$$G_{2p} : S \overset{1}{\Rightarrow} AA \overset{3}{\Rightarrow} BBA \overset{3}{\Rightarrow} BBBB \overset{4*}{\Rightarrow} AAAA \overset{2*}{\Rightarrow} aaaa$$

❑

16. Grammaire d'arbres

Les grammaires d'arbres permettent de représenter l'information spatiale 2D, il existe aussi des grammaires 3D.

Définition

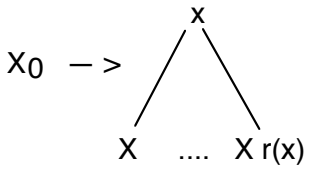
$G_t = (V, r, P, S)$

V : vocabulaire terminal et auxiliaire ($V_T \cup V_N$)

r : rang de a ($a \in V_T$) = nombre maximum de fils de a

S : départ

P :

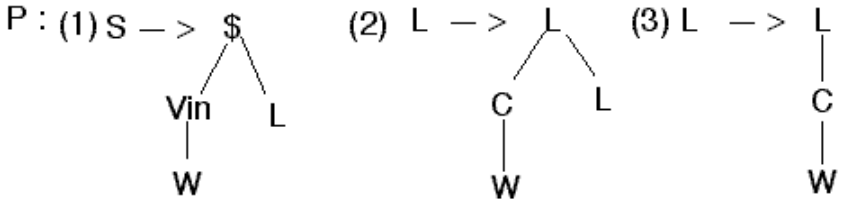


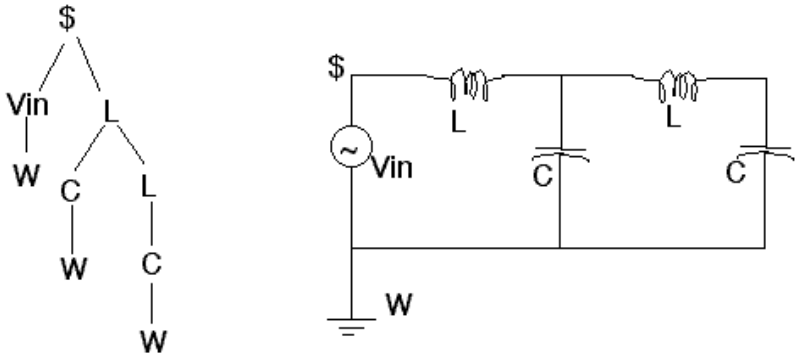
$x \in V_T$ $\quad X0, X1, \ldots Xr(x) \in V_N$

Exemple

V = { S, Vin, L, C, W, $ }

r (Vin) = 1, r(L) = {2, 1}, r(C) = 1, r(W) = 0, r($) = 2



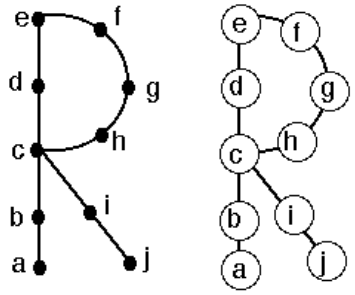


❑

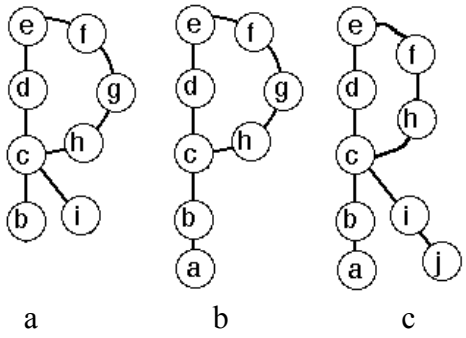
17. Grammaires de graphes

Ces grammaires sont présentées pour illustrer des représentations hiérarchiques de formes, chaque niveau de la hiérarchie correspond à des niveaux de détails différents (vue générale, vues plus fines) et chaque niveau peut être représenté par des formalismes différents.

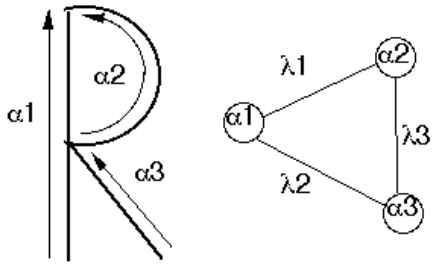
On considère d'abord une représentation à un seul niveau par un graphe. La figure suivante illustre une segmentation en primitives qui sont de petits segments de droites et de courbes. Dans le graphe de décomposition associé, les nœuds correspondent à des primitives, les arcs, à des relations de connexité.



La reconnaissance de formes inconnues peut consister à apparier un graphe de référence (celui du R ci-dessus par exemple) au graphe de la forme à reconnaître. Pour une reconnaissance performante, il faut faire la différence entre les éliminations de nœuds tolérées et celles qui ne le sont pas. Ainsi, l'élimination des deux nœuds (a) et (j) n'a pas la même signification que l'élimination des deux nœuds (i) et (j). Il faut pouvoir adapter un algorithme de comparaison de graphes à chaque classe de formes.

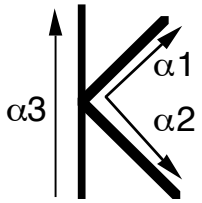


Les grammaires de graphes utilisées par Sanfeliu et Fu (1983) sous le nom de *Descriptive Graph Grammar,* adoptent deux niveaux de décomposition qui permettent de séparer la représentation de la structure globale et des structures locales. Le niveau global est un graphe, il décrit la structure forte de la classe de formes, à partir de sous-formes (courbes fermés, groupements de points alignés, ...).



Les nœuds du graphe de décomposition représentent des sous-formes et les arcs, des relations de connexité. Un ensemble d'attributs γ_{ij} défini les contraintes spatiales entre les nœuds qui doivent être respectées pour une classe de formes donnée. Chaque nœud est associé à une représentation de la sous-forme, la variabilité intra-classe est sensible à ce niveau. La représentation locale des sous-formes peut être un graphe, une grammaire, des symboles ... elle n'est pas fixée a priori. Sanféliu et Fu ont utilisé des grammaires pour représenter les sous-formes. Il est plus facile d'apprendre des grammaires locales que la grammaire globale de la forme dans la mesure où les sous-formes ont des structures plus simples que celle de la forme globale.

Exemple



Nœuds du graphe de décomposition :

α_1 - décrit par G_1

α_2 - décrit par G_2

α_3 - décrit par G_3

Les contraintes entre les nœuds sont données par les valeurs des variables de γ_{ij} :

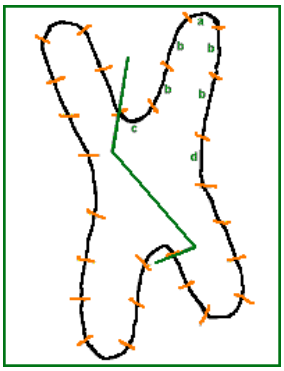
γ_{ij} = (Xmax, Ymax, Xmin, Ymin, nbre connections)

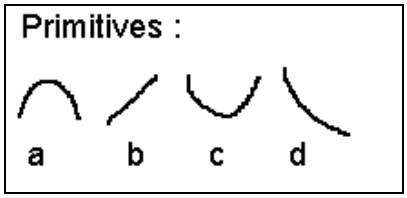
- Xmax = 6 : quelconque
  1 : Xmax αi >Xmax αj
- Ymax = 6 : quelconque
  2 : Ymax αi < Ymax αj
- Xmin = 0 : Xmin αi = Xmin αj
  1 : Ymax αi > Ymax αj
- Ymin = 6 : quelconque
  2 : Ymin αi > Ymin αj
- nbre connections = 1 si une connection

❑

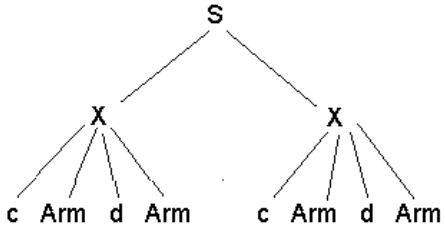
Il serait difficile de terminer un exposé de reconnaissance des formes syntaxiques sans parler des chromosomes qui ont été les formes emblèmatiques du domaine dans les années 70. On illustre ici un autre type de représentation à deux niveaux.

Exemple





Représentation globale du chromosome :



Chaque sous forme *Arm* est représentée par une grammaire apprise à partir d'exemple :

VT = {a, b}	VN ={*Arm*, A, B, C, F}	S = *Arm*
Arm →D	D → bFb	
D → bDb	D → bBb	
B → Ab	B → Bb	
F → bA	A → a	

❑

18. Actuellement ...

Les langages visuels (de programmation, d' interrogation) font largement appel à des notions de grammaires (2D pour la plupart). De même, l'interaction par des modalités naturelles (geste, parole) qui demande souvent de définir des syntaxes de commandes élaborées. Dernièrement, un langage pour éditer, reconnaître et visualiser des dessins structurés dessinés à main levée a été présenté, *LADDER* [Hammond et Davis 2003]. Ce langage consiste en un ensemble de formes prédéfinies et de contraintes, c'est un moule pour recevoir des grammaires spécifiques à des domaines de formes. Les grammaires et les analyseurs sont vus plutôt du point de vue de la programmation et du système interactif dans lequel ils sont intégrés que du point de vue des langages formels. C'est ce type de point de vue qui caractérise les relations entre langages formels et RdF.

19. Bibliographie

[FU 74] K. S. FU. *Syntactic Methods in Pattern Recognition*. Academic Press, New York, 1974.

[FU 82]] K. S. FU. *Syntactic Pattern Recognition and Application*s. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[Hammond et Davis 2003] HAMMOND Tracy, DAVIS Rendall. "LADDER: A Language to Describe Drawing, Display, and Editing in Sketch Recognition". In Proceedings of IJCAI-03, 2003.

[Miclet 84] Laurent MICLET. *Méthodes structurelles pour la reconnaissance des formes*. Eyrolles et CNET-ENST, 1984.

[Cornuéjols, Miclet 02] Antoine CORNUEJOLS, Laurent MICLET. *Apprentissage artificiel*. Eyrolles, 2002.

CHAPITRE 4 : CHAÎNES DE MARKOV ET MODÈLES DE MARKOV CACHÉS

Marc Sigelle

Motivation et Plan

Les chaînes de Markov forment depuis plusieurs dizaines d'années un sujet de choix aussi bien au niveau des investigations mathématiques [Meyn and Tweedie(1993)] que des applications à base de chaînes de Markov cachées en Reconnaissance des Formes [Cornuéjols and Miclet(2002)] : Traitement de la Parole, Traitement de l'Écriture, modèles de langages etc.. Au niveau théorique on considère souvent des modèles à temps discret ou continu (dans ce cas on parle de processus stochastiques de Markov) avec des espaces d'états finis ou infinis aussi bien discrets que continus. Nous nous limiterons ici au cas du **temps discret** et à un espace d'états **fini.** Nous avons vu au Chapitre 2 que la reconnaissance consiste à comparer un échantillon de test (par exemple en effectuant une programmation dynamique) :

- avec un ensemble de référence
- par une "distance" appropriée.

Ce processus devient coûteux lorsqu'il y a beaucoup de références, par exemple dans les grandes bases de données actuelles. De plus il s'agit d'estimer ou de modéliser les distances élémentaires, ce qui n'est pas trivial. On trouve donc ici l'intérêt d'une l'approche stochastique, où :

- un "modèle " remplace l'ensemble de références.
- les probabilités sont calculées par apprentissage (ce qui remplace les distances).

Le plan de ce chapitre sera le suivant :

1) introduction : processus stochastique(s)
2) chaînes de Markov - exemples
3) chaînes de Markov cachées (HMMs) - exemples
4) apprentissage avec les HMMs - exemples

1 Modèle stochastique

Un processus aléatoire est lié à une variable d'état pouvant changer de valeur au hasard aux instants $t = 1, 2 \dots T$. On définit alors :

- la variable aléatoire (v.a.) associée : $X(t) =$ état observé au temps t, notée aussi Q_t .
- avec les notations, qui sont équivalentes : $X(t) = i$ ou $Q_t = q_t$.
- l'évolution du système est donc décrite par une suite de transitions depuis l'état initial q_1.
- $\Rightarrow$ problème : connaître les chaînes de transition $q_1 \rightarrow q_2 \rightarrow \dots q_t \ \ \forall t \leq T$.

On peut pour cela calculer la loi d'évolution du système *ie.* la probabilité jointe d'une séquence d'états en utilisant la formule générale et fondamentale [1] : $P(A,\ B) = P(B\ /\ A)\ P(A)$

$$P(q_1,\ q_2,\ \dots q_T) = P(q_T\ /\ q_1 \dots q_{T-1})\ \underbrace{P(q_1 \dots q_{T-1})}$$

$$\text{''} \qquad\qquad P(q_{T-1}\ /\ q_1 \dots q_{T-2})\ P(q_1 \dots q_{T-2})$$

$$= \ P(q_1)\ P(q_2\ /\ q_1)\ P(q_3\ /\ q_1\ q_2) \dots P(q_T\ /\ q_1 \dots q_{T-1}) \qquad (1)$$

En résumé pour calculer cette loi jointe $P(q_1,\ q_2,\ \dots q_T)$ il faut donc connaître :

- la probabilité initiale $P(q_1)$.
- les probabilités conditionnelles $P(q_t\ /\ q_1 \dots q_{t-1}) \quad \forall t = 2 \dots T$.

2 Chaîne de Markov à temps discret et espace d'états fini

On considère un espace d'états $\{1, 2 \dots M\}$ ensemble fini.

- la propriété de Markov d'ordre k dit une dépendance limitée aux k instants précédents :

$$P(q_t\ /\ q_1 \dots q_{t-1}) = P(q_t\ /\ \underline{q_{t-k} \dots q_{t-1}})$$

- en général l'ordre d'une chaîne de Markov est 1 ou 2. Dans le cas usuel $k = 1$ on a :

$$P(q_t\ /\ q_1 \dots q_{t-1}) = P(q_t\ /\ q_{t-1}) \ \ \forall t$$

$$\Rightarrow P(q_1,\ q_2,\ \dots q_T) \quad = \quad P(q_1)\ P(q_2\ /\ q_1)\ P(q_3\ /\ q_2) \dots P(q_T\ /\ q_{T-1}) \ \ \forall T \qquad (2)$$

[1] nous omettrons souvent dans la suite la variable aléatoire dans la probabilité que celle-ci ait une valeur donnée, par exemple : $P(Q_t = q_t) \rightarrow P(q_t)$.

2.1 chaîne de Markov (d'ordre 1) stationnaire

Le système est décrit par des probabilités de transition $i \rightarrow j$ qui ne dépendent pas du temps :

$$P(Q_t = j \ / \ Q_{t-1} = i) = P(Q_{t+k} = j \ / \ Q_{t+k-1} = i) = a_{ij}$$

On définit alors :

$A = [\ a_{ij}\ ]$	matrice des probabilités de transition	$M \times M$
$\Pi = [\ \pi_i\ ]$	probabilités initiales (d'avoir une valeur d'état donnée)	M
$\pi_i = P(Q_1 = i)$		

Les formules suivantes ont bien sur lieu (normalisation) :

$$\forall i \in \{1 \dots M\} \quad 0 \leq \pi_i \leq 1 \quad \text{et} \ \sum_{i=1}^{M} \pi_i = 1$$

$$\forall i,\ j \in \{1 \dots M\} \quad 0 \leq a_{ij} \leq 1 \quad \text{et} \ \sum_{j=1}^{M} a_{ij} = 1$$

2.2 cas d'une station météo, étude de l'évolution du temps [Rabiner(1989)]

3 états : 1 = pluie, 2 = nuages , 3 = soleil . Le modèle est représenté par les matrices A et π :

$$A = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} \quad \text{on observe 3 = soleil à } t = 1 = \text{lundi}$$

$\rightarrow$ quelle est la probabilité P que le temps au cours du reste de la semaine soit

soleil	soleil	soleil	pluie	pluie	soleil	nuages ?
3	3	3	1	1	3	2

$$P(Q_1 = 3,\ Q_2 = 3,\ Q_3 = 3,\ Q_4 = 1,\ Q_5 = 1,\ Q_6 = 3,\ Q_7 = 2 \ / \ \text{modèle})$$

$$= \Pi_3 \ P(Q_2 = 3 \ / \ Q_1 = 3) \ P(Q_3 = 3 \ / \ Q_2 = 3) \ P(Q_4 = 1 \ / \ Q_3 = 3) \ \times$$

$$P(Q_5 = 1 \ / \ Q_4 = 1) \ P(Q_6 = 3 \ / \ Q_5 = 1) \ P(Q_7 = 2 \ / \ Q_6 = 3)$$

$$= 1 \ (a_{33})^2 \ a_{31} \ a_{11} \ a_{13} \ a_{32} = 7.68.10^{-4}$$

2.3 "durée de vie d'un état" [Rabiner(1989)]

Sachant que le système est dans l'état $i \rightarrow$ quelle est la probabilité qu'il y reste la durée d ?

$$P_i(d) = P(Q_1 = i,\ Q_2 = i \ \dots Q_{d-1} = i,\ Q_d = i,\ Q_{d+1} \neq i)$$

$$= P(Q_1 = i) \ P(Q_2 = i \ / \ Q_1 = i) \dots P(Q_d = i \ / \ Q_{d-1} = i) \ P(Q_{d+1} \neq i \ / \ Q_d = i)$$

$$= (a_{ii})^{d-1} \ (1 - a_{ii})$$

On obtient là un modèle exponentiel caractéristique. La durée moyenne de vie de l'état i est :

$$\mathbb{E}[D] = \sum_{d=1}^{+\infty} d \ P_i(d) = \sum_{d=1}^{+\infty} d \ (a_{ii})^{d-1} \ (1 - a_{ii}) = \frac{1}{1 - a_{ii}}$$

3 HMM hidden Markov models - chaînes de Markov cachées

Il s'agit de deux processus stochastiques dont l'un est caché *ie.* :

- une suite de v.a. cachée = suite d'**états** $q_1,\ q_2 \dots q_t, \dots q_T$
- l'autre, observable = suite de **symboles** (observations) produits en chacun de ces états $o_1,\ o_2 \dots o_t, \dots o_T$

La différence fondamentale avec les modèles classiques est que l'on on n'observe pas directement les états mais seulement les symboles produits par ces états. De plus on "code" l'évolution temporelle dans la séquence d'états (cachés), dont l'espace des valeurs possibles est en général de cardinal beaucoup plus faible que celui des observations.

3.1 caractérisation d'un HMM (λ) : deux hypothèses fondamentales

- Indépendance conditionnelle des observations :

$$P(o \ / \ q, \lambda) = P(o_1 \dots o_T \ / \ q_1 \dots q_T, \lambda) = \prod_{t=1}^{T} \underbrace{P(o_t \ / \ q_t, \lambda)}_{b_j(o_t)} \qquad \swarrow (q_t = j)$$

- La loi a priori sur la séquence d'états q est une chaîne de Markov stationnaire d'ordre 1 :

$$P(q \ / \ \lambda) = \underbrace{P(q_1 \ / \ \lambda)}_{\pi_i} \prod_{t=1}^{T-1} \underbrace{P(q_{t+1} \ / \ q_t)}_{a_{ij}} \qquad \swarrow (q_t = i \ , \ q_{t+1} = j)$$

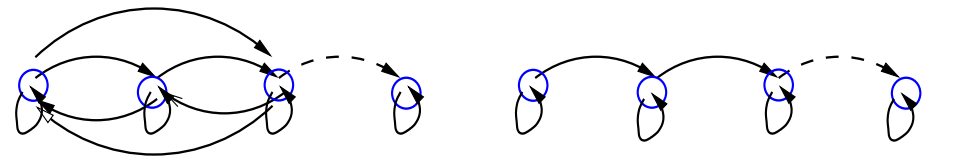
3.2 définition et notations pour un HMM à temps discret

un modèle HMM $\lambda = (A, B, \pi)$ est alors défini par :

S	=	$\{1 \dots M\}$ ensemble des états du modèle M états	
V	=	$\{V_1 \dots V_N\}$ ensemble des N symboles observables dans chaque état	$(N >> M)$
A	=	$[\ a_{ij}\ ]$ matrice des probas de transition (stationnaire)	$M \times M$
Π	=	$[\ \pi_i\ ]$ probas initiales	M
B	=	$[\ b_j(o_t)\ ]$ matrice des probas des symboles émis dans chaque état j	
avec		$b_j(o_t) = P(O_t = o_t \ / \ q_t = j)$ (stationnaire)	$M \times N$

3.3 caractéristiques des modèles HMM

La topologie désigne le graphe des transitions possibles entre états :



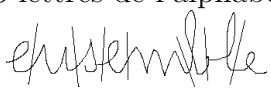
ergodique **gauche−droite**

$a_{ij} > 0 \quad \forall i, j \in \{1..M\}$ $a_{ij} > 0$ si $j = i$ ou $j = i + 1$

3.4 1er exemple : reconnaissance de l'écriture cursive (minuscules)

Les hypothèses sont les suivantes :

- il y a $M = 26$ états cachés correspondant aux 26 lettres de l'alphabet.

  Un mot est supposé déja segmenté en lettres : 

- il y a N "formes" observables, après quantification des observations. Ainsi la forme observée : '***e***' peut provenir de la lettre 'e' ou 'l' : il y a donc des confusions possibles. C'est le rôle du modèle de Markov sous-jacent de lever la confusion par la connaissance des probabilités de transitions permises entre lettres consécutives ou à l'aide d'un dictionnaire. Ainsi ***ensemble*** aura une probabilité faible d'être reconnu comme : 'enslble' .

- les formes dépendent du style et de l'algorithme de reconnaissance.

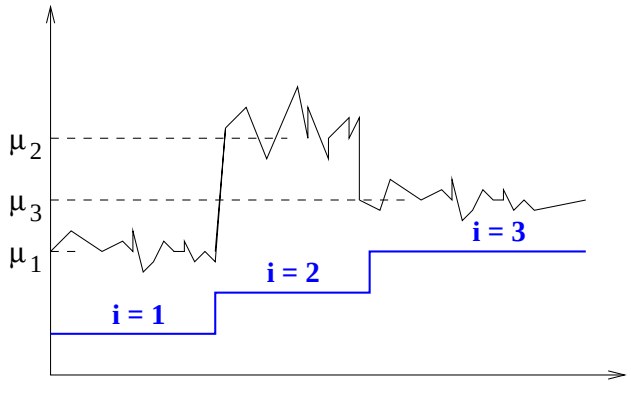
$\Rightarrow$ c'est donc un HMM avec :

- $A = \{a_{ij}\}$ probabilités de transition entre lettres : modèle de langage.
- $\Pi = \{\pi_i\}$ probabilités initiales des lettres.
- $B = \{b_j(k)\}$ probabilités des observations dans chaque état :
  cela dépend du système de reconnaissance OCR $\rightarrow$ apprentissage par comptage.

$\Rightarrow$ **but** : trouver la séquence d'états "optimale" (mot optimal).

3.5 2ème exemple : reconnaissance de la parole

Les observations sont supposées au départ continues et scalaires, pouvant être décrites en première approximation par des modèles gaussiens :



$$P(o_t \ / \ q_t = i, \ \lambda) = \frac{1}{\sqrt{2\pi}\sigma_i} \ \exp - \frac{(o_t - \mu_i)^2}{2{\sigma_i}^2}$$

En fait les descripteurs extraits à partir des observations (depuis des bancs de filtre par exemple) sont souvent vectoriels de dimension d et suivent des lois gaussiennes multi-variées :

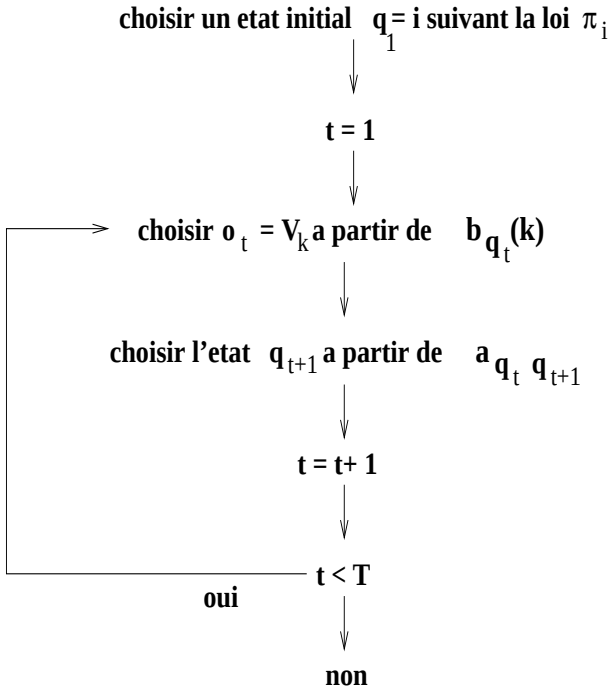
$$P(\bar{o}_t \ / \ q_t = i, \ \lambda) = \frac{1}{(2\pi)^{d/2}} \ \frac{1}{\sqrt{|\mathbf{\Sigma_i}|}} \ \exp - \frac{1}{2} \ (\bar{o}_t - \bar{\mu}_i)^t \ {\mathbf{\Sigma_i}}^{-1} \ (\bar{o}_t - \bar{\mu}_i)$$

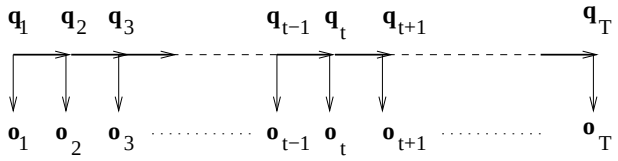
$\bar{\mu}_i$ est la moyenne de l'observation pour l'état i et $\mathbf{\Sigma_i}$ la matrice de variance-covariance associée. Cependant on affine souvent la modélisation en employant des mélanges de gaussiennes :

$$P(o_t \ / \ q_t = i, \ \lambda) = \sum_p w_p \ \mathcal{N}_p(o_t) \rightarrow \text{ idem pour des lois multi-variées.}$$

3.6 simulation d'une HMM dont le modèle est connu

On veut synthétiser une suite d'observations suivant un modèle connu. Une séquence d'observations $o = o_1 \dots o_t \dots o_T$ est alors produite par :





4 Etapes fondamentales : apprentissage - reconnaissance

a) apprentissage

• on dispose de plusieurs modèles HMM : λ_u, en vue de leur confrontation ultérieure. Exemple : mots isolés en parole (dictionnaire), caractères isolés en écriture (A-Z,0-9).

• le nombre d'états peut dépendre du modèle, et est supposé connu par avance. Il sera toujours noté M, lorsqu'aucune confusion n'est à craindre. Exemple : chaque phonème est de durée plus ou moins longue (et possède donc plus ou moins d'états constitutifs), chaque caractère manuscrit est plus ou moins large. [2]

• on dispose pour chaque modèle d'une base d'apprentissage $\{o^{(l)}\}_{l=1..L}$, de taille L (nombre d'exemples d'apprentissage) pouvant dépendre là aussi du modèle considéré. De plus la durée de chacune des séquences peut également être variable à l'intérieur d'un même modèle.

• apprentissage = estimation des paramètres de chaque modèle λ_u : $\pi_i,\ a_{ij}\ , b_j(o_t)$

b) reconnaissance

• on dispose d'une base de test $\{o^{(r)}\}$: signaux éventuellement "dégradés".

• chaque signal de test est confronté aux divers modèles λ_u appris.

• pour un signal de test $o^{(r)}$ donné → on sélectionne le modèle le plus "vraisemblable".

• on évalue les performances de la reconnaissance globale sur l'ensemble de la base de test.

[2] L'estimation du nombre d'états (optimal) associé un modèle donné est toujours un problème ouvert.

4.1 estimation des paramètres en "données complètes"

La base de tous les raisonnements futurs est l'étude du cas (simple) où une séquence d'observation et la séquence d'états associés sont connues [3] : $(o,\ q)$. La loi **jointe** observations-états s'écrit :

$$\begin{aligned} P(o,\ q\ /\ \lambda) &= P(o\ /\ q,\ \lambda)\ P(q\ /\ \lambda) \\ &= \underbrace{P(q_1\ /\ \lambda)}_{\pi_i}\ \prod_{t=1}^{T-1}\ \underbrace{P(q_{t+1}\ /\ q_t,\ \lambda)}_{a_{ij}}\ \prod_{t=1}^{T}\ \underbrace{P(o_t\ /\ q_t,\ \lambda)}_{b_j(o_t)} \end{aligned}$$

Si on a affaire à plusieurs séquences d'observations indépendantes (*ie.* dans la phase d'apprentissage) avec leurs séquences d'états associés connues $o^{(l)}, q^{(l)}\ , l = 1..L$, alors :

$$P(o,\ q\ /\ \lambda) = \prod_{l=1}^{L} P(o^{(l)}, q^{(l)}\ /\ \lambda) = \prod_{i=1}^{M} {\pi_i}^{N_i^{(1)}}\ \ \prod_{i,j=1}^{M} {a_{ij}}^{N_{ij}}\ \ \prod_{j,k} b_j(o_t = k)^{N_{jk}} \qquad (3)$$

où :

- $N_i^{(1)} = \sum_{l=1}^{L} \mathbb{1}_{q_1^{(l)}=i}$ est le nombre de séquences d'apprentissage dont l'état initial est i .
- $N_{ij} = \sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} \mathbb{1}_{q_t^{(l)}=i,\ q_{t+1}^{(l)}=j}$ est le nombre de fois dans l'ensemble des séquences d'apprentissage où l'on rencontre le couple d'états (consécutifs) : $q_t^{(l)} = i,\ q_{t+1}^{(l)} = j$.
- $N_{jk} = \sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} \mathbb{1}_{q_t^{(l)}=j,\ o_t^{(l)}=k}$ est le nombre de fois dans l'ensemble des séquences d'apprentissage où l'on rencontre le couple simultané état-observation : $q_t^{(l)} = j,\ o_t^{(l)} = k$.

On peut alors estimer les paramètres de la chaîne de Markov sous-jacente au maximum de vraisemblance, *ie.* en maximisant la probabilité jointe $P(o,\ q\ /\ \lambda)$ par rapport aux paramètres du modèle sous contraintes. En effet, ces paramètres étant des probabilités, donc normalisés à 1, l'emploi d'autant de multiplicateurs de Lagrange associés $\{\nu\}$ permet d'obtenir les estimateurs suivants.

- Ainsi, pour les probabilités initiales, définissons le lagrangien suivant :

$$\mathcal{L} = \log P(o,\ q\ /\ \lambda) - \nu \left(\sum_{i=1}^{M} \pi_i - 1 \right)$$

De (3) nous déduisons :

$$\frac{\partial \mathcal{L}}{\partial \pi_i} = \frac{\partial \log P(o,\ q\ /\ \lambda)}{\partial \pi_i} - \nu \quad = \quad \frac{N_i^{(1)}}{\pi_i} - \nu = 0$$

$$\Rightarrow \frac{N_i^{(1)}}{\pi_i} = \nu = \frac{\sum_{i=1}^{M} N_i^{(1)}}{\sum_{i=1}^{M} \pi_i}\ \text{(proportions !)} = L \quad \Rightarrow \quad \widehat{\pi_i} = \frac{N_i^{(1)}}{L} = \frac{\sum_{l=1}^{L} \mathbb{1}_{q_1^{(l)}=i}}{L} \qquad (4)$$

[3] on peut considérer q comme une segmentation, supposée connue, du signal observé o.

On retrouve bien pour l'état initial i la probabilité empirique en tant que quotient du nombre de séquences qui commencent par cet état sur le nombre de séquences d'apprentissage total.

• De même, pour les probabilités de transitions, définissons le lagrangien suivant :

$$\mathcal{L} = \log P(o,\ q\ /\ \lambda) - \sum_{i=1}^{M} \nu_i \left(\sum_{j=1}^{M} a_{ij} - 1 \right)$$

De (3) nous déduisons :

$$\frac{\partial \mathcal{L}}{\partial a_{ij}} = \frac{\partial \log P(o,\ q\ /\ \lambda)}{\partial a_{ij}} - \nu_i \quad = \quad \frac{N_{ij}}{a_{ij}} - \nu_i = 0$$

$$\Rightarrow \frac{N_{ij}}{a_{ij}} = \nu_i = \frac{\sum_{j=1}^{M} N_{ij}}{\sum_{j=1}^{M} a_{ij}} \text{ (proportions !)} = N_i^* \ \Rightarrow \ \widehat{a_{ij}} = \frac{N_{ij}}{N_i^*} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} \mathbb{1}_{q_t^{(l)}=i,\ q_{t+1}^{(l)}=j}}{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} \mathbb{1}_{q_t^{(l)}=i}} \quad (5)$$

On trouve le quotient du nombre total de couples d'instants consécutifs où l'on rencontre les états successifs i et j sur le nombre d'instants (excepté l'instant final) où l'on est sur l'état i.

• En ce qui concerne les lois d'observation on pourrait obtenir des formules similaires pour un ensemble d'observations discret fini [4], que nous laissons au lecteur à titre d'exercice. Comme on l'a vu précédemment, les observations sont souvent continues et modélisées, dans le cas le plus simple, par des gaussiennes pures. La composante de la log-vraisemblance dépendant de ces paramètres peut s'écrire assez facilement à l'aide des fonctions indicatrices d'état :

$$LL = \sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} \sum_{i=1}^{M} \mathbb{1}_{q_t^{(l)}=i} \ \left(-\frac{(o_t - \mu_i)^2}{2\sigma_i^2} - \log \sigma_i \right)$$

Il est alors assez facile de montrer en dérivant la log-vraisemblance par rapport aux divers paramètres des lois gaussiennes (moyenne et variance dans chaque état) que l'on on obtient les moyennes et variances empiriques suivantes :

$$\widehat{\mu_i} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} o_t^{(l)}\ \mathbb{1}_{q_t^{(l)}=i}}{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} \mathbb{1}_{q_t^{(l)}=i}} \qquad \widehat{(\sigma_i)^2} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} \left(o_t^{(l)} - \widehat{\mu_i}\right)^2\ \mathbb{1}_{q_t^{(l)}=i}}{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} \mathbb{1}_{q_t^{(l)}=i}} \quad (6)$$

On laisse le soin au lecteur de trouver les formules équivalentes dans le cas des lois gaussiennes multi-variées.

[4] par exemple quantifié à l'aide d'un codebook.

5 Trois problèmes fondamentaux liés aux HMM [Rabiner(1989)]

Etant donnés : un modèle $\lambda = (A, B, \pi)$ (exemple : mot ou caractère isolé) et une séquence d'observations $o = o_1 \dots o_t \dots o_T$:

1) $\rightarrow$ calculer $P(o \ / \ \lambda)$.

2) $\rightarrow$ trouver une séquence d'états "optimale" a posteriori $\widehat{q} = q_1 \dots q_t \dots q_T = \arg\max_q P(q \ / \ o, \ \lambda)$

3) $\rightarrow$ étant données plusieurs séquences d'observations $o^{(l)}$ (de longueur fixe ou variable) et un modèle λ courant , réestimer les paramètres du modèle $\lambda = (A, B, \pi)$ pour maximiser (en fait augmenter) la "vraisemblance" des observations $P(\dots o^{(l)} \dots \ / \ \lambda)$.

Ce programme en trois points peut être réalisé à partir des variables backward-forward que nous allons maintenant introduire.

5.1 introduction aux variables backward-forward

On va dans ce qui suit utiliser l'aspect "**graphique**" des HMM. On rappelle ce qui connu :

- une séquence d'observations o de durée T.
- un modèle λ

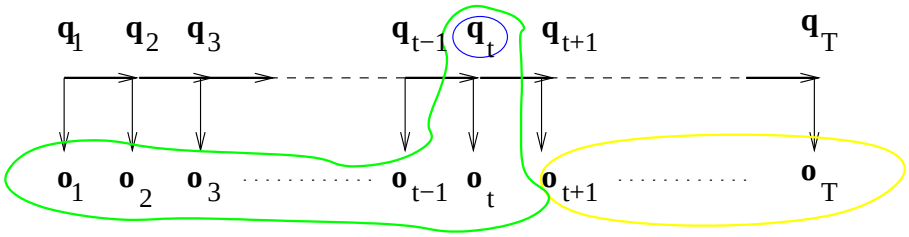


FIG. 1: aspect graphique des HMM

Essayons d'analyser l'expression suivante (Fig. 1) :

$$P(o_{t+1} \dots o_T \ / \ o_1 \dots o_t, \ q_t)$$

Si nous pensons en termes de simulation (cf. paragraphe 3.6), on voit que pour simuler la séquence $(o_{t+1} \dots o_T)$ connaissant $(o_1 \dots o_t, \ q_t)$, il suffit de connaître la valeur de q_t [5] puisque celle-ci permettra de générer les valeurs de q_{t+1}, o_{t+1} etc.. Cette remarque permet alors de définir les expressions suivantes pour chaque instant $1 \leq t \leq T$:

$$P(o_{t+1} \dots o_T \ / \ o_1 \dots o_t, \ q_t) = \boxed{P(o_{t+1} \dots o_T \ / \ \mathbf{q_t}) = \beta_t(i)} \quad \leftarrow \text{variable backward}$$

$$\boxed{P(o_1 \dots o_t, \ \mathbf{q_t}) = \alpha_t(i)} \quad \leftarrow \text{variable forward}$$

[5] qui est le noeud "entrant" vers $(o_{t+1} \dots o_T)$.

5.2 application des variables backward-forward

On a pour un instant donné t :

$$\begin{aligned} P(o,\ q_t = i\ /\ \lambda) &= P(o_1 \dots o_t,\ q_t,\ o_{t+1} \dots o_T) = P(o_{t+1} \dots o_T\ /\ o_1 \dots o_t,\ q_t)\ P(o_1 \dots o_t,\ q_t) \\ &= \underbrace{P(o_{t+1} \dots o_T\ /\ q_t)}_{\beta_t(i)}\ \underbrace{P(o_1 \dots o_t,\ q_t)}_{\alpha_t(i)} \end{aligned}$$

Donc

$$P(o,\ q_t = i\ /\ \lambda) = \beta_t(i)\ \alpha_t(i) \tag{7}$$

$$\Rightarrow P(o\ /\ \lambda) = \sum_{i=1}^{M} P(o,\ q_t = i\ /\ \lambda) = \sum_{i=1}^{M} \beta_t(i)\ \alpha_t(i) \tag{8}$$

$$\Rightarrow P(q_t = i\ /\ o,\ \lambda) = \frac{P(o,\ q_t = i\ /\ \lambda)}{P(o\ /\ \lambda)} = \frac{\alpha_t(i)\ \beta_t(i)}{\sum_{i=1}^{M} \alpha_t(i)\ \beta_t(i)} \tag{9}$$

5.3 formules à deux états

On étudie de la même façon que précédemment les probabilités suivantes faisant intervenir les états associés à deux instants consécutifs :

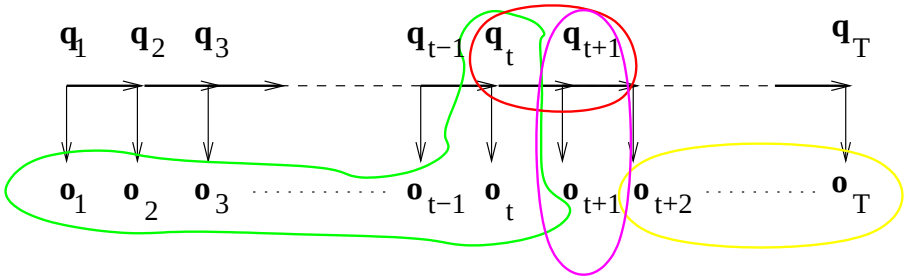


FIG. 2: aspect graphique des HMM : formules à 2 états

Décomposons la probabilité suivante :

$$P(o,\ q_t = i,\ q_{t+1} = j)$$

On a :

$$\begin{aligned} P(o,\ q_t = i,\ q_{t+1} = j) &= P(o_{t+2} \dots o_T\ /\ o_1 \dots o_{t+1},\ q_t,\ q_{t+1})\ P(o_1 \dots o_{t+1},\ q_t,\ q_{t+1}) \\ &= P(o_{t+2} \dots o_T\ /\ q_{t+1})\ P(o_{t+1}\ /\ q_{t+1})\ P(q_{t+1}\ /\ q_t)\ P(o_1 \dots o_t,\ q_t) \\ &= \beta_{t+1}(j)\ b_j(o_{t+1})\ a_{ij}\ \alpha_t(i) \end{aligned}$$

Donc

$$P(o,\ q_t = i,\ q_{t+1} = j\ /\ \lambda) \ = \ \beta_{t+1}(j)\ b_j(o_{t+1})\ a_{ij}\ \alpha_t(i) \tag{10}$$

$$P(q_t = i,\ q_{t+1} = j\ /\ o, \lambda) \ = \ \frac{P(o,\ q_t = i,\ q_{t+1} = j\ /\ \lambda)}{P(o\ /\ \lambda)} = \frac{\beta_{t+1}(j)\ b_j(o_{t+1})\ a_{ij}\ \alpha_t(i)}{\sum_{i=1}^{M} \alpha_t(i)\ \beta_t(i)} \tag{11}$$

• dans (10) "sommons" sur toutes les valeurs possibles de j :

$$P(o,\ q_t = i\ /\ \lambda) \ = \ \alpha_t(i)\ \left[\sum_{j=1}^{M} \beta_{t+1}(j)\ b_j(o_{t+1})\ a_{ij}\right] = \alpha_t(i)\ \beta_t(i) \quad \text{(d'après (7)) !}$$

$$\Rightarrow \beta_t(i) \ = \ \sum_{j=1}^{M} \beta_{t+1}(j)\ b_j(o_{t+1})\ a_{ij}$$

• dans (10) on "somme" maintenant sur toutes les valeurs possibles de i :

$$P(o,\ q_{t+1} = j\ /\ \lambda) \ = \ \beta_{t+1}(j)\ b_j(o_{t+1})\ \left[\sum_{i=1}^{M} \alpha_t(i)\ a_{ij}\right] = \alpha_{t+1}(j)\ \beta_{t+1}(j) \quad \text{(d'après (7)) !}$$

$$\Rightarrow \alpha_{t+1}(j) \ = \ b_j(o_{t+1})\ \left[\sum_{i=1}^{M} \alpha_t(i)\ a_{ij}\right]$$

5.4 calcul des variables backward-forward

En récapitulant les résultats précédents on obtient donc les formules de récurrence :

$$\alpha_{t+1}(j) \ = \ b_j(o_{t+1})\ \left[\sum_{i=1}^{M} \alpha_t(i)\ a_{ij}\right] \tag{12}$$

$$\beta_t(i) \ = \ \sum_{j=1}^{M} \beta_{t+1}(j)\ b_j(o_{t+1})\ a_{ij} \tag{13}$$

L'initialisation de ces variables se fait avec les remarques simples suivantes :

$$\alpha_T(i) = P(o, q_T = i) = \alpha_T(i)\ \beta_T(i) \Rightarrow$$

$$\beta_T(i) \ = \ 1$$

$$\alpha_1(i) \ = \ P(o_1,\ q_1 = i) = P(o_1\ /\ q_1 = i)\ P(q_1 = i) = \pi_i\ b_i(o_1)$$

6 Estimation des paramètres en "données incomplètes"

Il s'agit du problème le plus difficile de ce chapitre. On ne connaît maintenant que les séquences d'observations (pas les états). Or on veut aussi estimer les paramètres du modèle HMM, et de façon optimale ! Pour utiliser les résultats vus en données complètes, il va falloir effectuer une "statistique" combinatoire sur l'ensemble des séquences d'états cachés possibles (chemins) associés à chaque observation (séquence temporelle). On verra que c'est la loi de l'ensemble des séquences d'états $P(q \ / \ o, \lambda)$ **a posteriori** *ie.* connaissant les séquences d'observations, qui va s'imposer d'un point de vue statistique dans ce qui suit.
Pour toute v.a. U, nous noterons alors son espérance a posteriori par : $\tilde{\mathbb{E}}[U] = \mathbb{E}[U \ / \ o, \lambda]$.
Annonçons maintenant de suite les résultats :

• les paramètres de la chaine de Markov (probabilités) vérifient les équations suivantes :

$$\widehat{\pi_i} = \frac{\tilde{\mathbb{E}}[N_i^{(1)}]}{L} = \frac{\sum_{l=1}^{L} \tilde{\mathbb{E}}[\mathbb{1}_{q_1^{(l)}=i}]}{L} = \frac{\sum_{l=1}^{L} P(q_1^{(l)} = i \ / \ o^{(l)}, \widehat{\lambda})}{L} \tag{14}$$

$$\widehat{a_{ij}} = \frac{\tilde{\mathbb{E}}[N_{ij}]}{\tilde{\mathbb{E}}[N_i{}^*]} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} \tilde{\mathbb{E}}[\mathbb{1}_{q_t^{(l)}=i, \ q_{t+1}^{(l)}=j}]}{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} \tilde{\mathbb{E}}[\ \mathbb{1}_{q_t^{(l)}=i}]} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} P(q_t^{(l)} = i, \ q_{t+1}^{(l)} = j \ / \ o^{(l)}, \widehat{\lambda})}{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} P(q_t^{(l)} = i \ / \ o^{(l)}, \widehat{\lambda})} \tag{15}$$

D'où cela vient-il ? Prenons le cas des probabilités de transition. On a vu que

$$\frac{\partial \log P(o, \ q \ / \ \lambda)}{\partial a_{ij}} = \frac{1}{a_{ij}} \ N_{ij}(q)$$

Ici on indique bien que la quantité N_{ij} dépend de la séquence d'états courante q. Or

$$\begin{aligned} P(o \ / \ \lambda) &= \sum_q P(o, \ q \ / \ \lambda) \\ \Rightarrow \frac{\partial P(o \ / \ \lambda)}{\partial a_{ij}} &= \frac{1}{a_{ij}} \ \sum_q N_{ij}(q) \ P(o, \ q \ / \ \lambda) \\ \Rightarrow \frac{\partial \log P(o \ / \ \lambda)}{\partial a_{ij}} &= \frac{1}{a_{ij}} \ \sum_q N_{ij}(q) \ \frac{P(o, \ q \ / \ \lambda)}{P(o \ / \ \lambda)} = \frac{1}{a_{ij}} \ \tilde{\mathbb{E}}[N_{ij}] \end{aligned}$$

En reprenant les conditions de normalisation des probabilités $\sum_{j=1}^{M} a_{ij} = 1 \ \ \forall i$ sous la forme des multiplicateurs de Lagrange précédents (paragraphe 4.1) on arrive facilement à :

$$\widehat{a_{ij}} = \frac{\tilde{\mathbb{E}}[N_{ij}]}{\sum_{j=1}^{M} \tilde{\mathbb{E}}[N_{ij}]} = \frac{\tilde{\mathbb{E}}[N_{ij}]}{\tilde{\mathbb{E}}[N_i{}^*]}$$

On voit (et ceci est très général) que d'un point de vue "mnémotechnique", il suffit de remplacer au numérateur et au dénominateur les quantités déterministes par leurs espérances a posteriori.

• les paramètres des lois d'observations gaussiennes vérifient les équations suivantes :

$$\widehat{\mu_i} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} o_t^{(l)}\ \tilde{\mathbb{E}}[\mathbb{1}_{q_t^{(l)}=i}]}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} \tilde{\mathbb{E}}[\mathbb{1}_{q_t^{(l)}=i}]} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} o_t^{(l)}\ P(q_t^{(l)}=i\ /\ o^{(l)},\widehat{\lambda})}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} P(q_t^{(l)}=i\ /\ o^{(l)},\widehat{\lambda})} \tag{16}$$

$$\widehat{(\sigma_i)^2} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} (o_t^{(l)}-\widehat{\mu_i})^2\ \tilde{\mathbb{E}}[\mathbb{1}_{q_t^{(l)}=i}]}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} \tilde{\mathbb{E}}[\mathbb{1}_{q_t^{(l)}=i}]} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} (o_t^{(l)}-\widehat{\mu_i})^2\ P(q_t^{(l)}=i\ /\ o^{(l)},\widehat{\lambda})}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} P(q_t^{(l)}=i\ /\ o^{(l)},\widehat{\lambda})} \tag{17}$$

On obtient donc un ensemble d'équations (14) (15) et (16) (17) auto-cohérentes puisque les paramètres figurent des deux cotés des équations, et non solubles analytiquement. Une méthode itérative extrêmement puissante pour estimer ces paramètres va maintenant être abordée.

7 Algorithme EM

Cet algorithme a pour base les résultats fondamentaux obtenus en données complètes. Pour les lecteurs intéressés, on trouvera un exposé complet de l'algorithme EM dans [Redner and Walker(1984)], avec en particulier l'application importante à l'estimation de mélanges (de gaussiennes). Son principe général en est le suivant. Chaque étape de l'algorithme comprend deux phases :

- phase E (Expectation) : estimation des statistiques a posteriori courantes $\tilde{\mathbb{E}}^{(\mathbf{n})}[\ U\ ]$
- phase M (Maximisation) : remise à jour des paramètres du modèle suivant le schéma :

$$\lambda^{(\mathbf{n+1})} = \arg\max_{\lambda} \tilde{\mathbb{E}}^{(\mathbf{n})}[\ \log P(q,\ o\ /\ \lambda)\ ] \tag{18}$$

Le résultat fondamental est que la vraisemblance des paramètres croît au cours des itérations. Dans notre cas l'EM va transformer le système d'équations précédentes exactes en un système itératif. Annonçons là aussi de suite les résultats de la phase M (Maximization) :

• en ce qui concerne les paramètres de la chaîne de Markov cachés :

$$\pi_i{}^{(\mathbf{n+1})} = \frac{\tilde{\mathbb{E}}^{(\mathbf{n})}[N_i^{(1)}]}{L} = \frac{\sum_{l=1}^{L} P(q_1^{(l)}=i\ /\ o^{(l)},\lambda^{(\mathbf{n})})}{L} \tag{19}$$

$$a_{ij}{}^{(\mathbf{n+1})} = \frac{\tilde{\mathbb{E}}^{(\mathbf{n})}[N_{ij}]}{\tilde{\mathbb{E}}^{(\mathbf{n})}[N_i{}^*]} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}-1} P(q_t^{(l)}=i,\ q_{t+1}^{(l)}=j\ /\ o^{(l)},\lambda^{(\mathbf{n})})}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}-1} P(q_t^{(l)}=i\ /\ o^{(l)},\lambda^{(\mathbf{n})})} \tag{20}$$

D'où cela vient-il ? Prenons ici encore à titre d'exemple le cas des probabilités de transition.

On forme le lagrangien : $\mathcal{L} = \tilde{\mathbb{E}}^{(\mathbf{n})}[\ \log P(q,\ o\ /\ \lambda)\ ] - \sum_{i=1}^{M} \nu_i \left(\sum_{j=1}^{M} a_{ij} - 1 \right)$.

Il faut bien noter ici que l'espérance a posteriori est prise pour les valeurs de paramètres courants, donc fixés. Comme on sait maintenant presque par coeur (!) que

$$\frac{\partial \log P(o,\ q\ /\ \lambda)}{\partial a_{ij}} = \frac{1}{a_{ij}}\ N_{ij}(q)\ , \quad \text{on en déduit}$$

$$\frac{\partial \mathcal{L}}{\partial a_{ij}} = \tilde{\mathbb{E}}^{(\mathbf{n})}[\ \frac{\partial \log P(o,\ q\ /\ \lambda)}{\partial a_{ij}}\ ] - \nu_i = \frac{1}{a_{ij}}\ \tilde{\mathbb{E}}^{(\mathbf{n})}[N_{ij}] - \nu_i \quad = \quad 0$$

$$\Rightarrow \frac{\tilde{\mathbb{E}}^{(\mathbf{n})}[N_{ij}]}{a_{ij}} = \nu_i = \frac{\sum_{j=1}^{M} \tilde{\mathbb{E}}^{(\mathbf{n})}[N_{ij}]}{\sum_{j=1}^{M} a_{ij}}\ \text{(proportions !)} = \tilde{\mathbb{E}}^{(\mathbf{n})}[N_i^*] \ \Rightarrow\ \widehat{a_{ij}} = \frac{\tilde{\mathbb{E}}^{(\mathbf{n})}[N_{ij}]}{\tilde{\mathbb{E}}^{(\mathbf{n})}[N_i^*]}$$

• pour les paramètres des lois gaussiennes :

$$\mu_i{}^{(\mathbf{n+1})} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} o_t^{(l)}\ \tilde{\mathbb{E}}^{(\mathbf{n})}[\mathbb{1}_{q_t^{(l)}=i}]}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} \tilde{\mathbb{E}}^{(\mathbf{n})}[\mathbb{1}_{q_t^{(l)}=i}]} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} o_t^{(l)}\ P(q_t^{(l)} = i\ /\ o^{(l)}, \lambda^{(\mathbf{n})})}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} P(q_t^{(l)} = i\ /\ o^{(l)}, \lambda^{(\mathbf{n})})} \tag{21}$$

$$(\sigma_i)^{2(\mathbf{n+1})} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} \left(o_t^{(l)} - \mu_i{}^{(\mathbf{n})}\right)^2\ \tilde{\mathbb{E}}^{(\mathbf{n})}[\mathbb{1}_{q_t^{(l)}=i}]}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} \tilde{\mathbb{E}}^{(\mathbf{n})}[\mathbb{1}_{q_t^{(l)}=i}]} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} \left(o_t^{(l)} - \mu_i{}^{(\mathbf{n})}\right)^2\ P(q_t^{(l)} = i\ /\ o^{(l)}, \lambda^{(\mathbf{n})})}{\sum_{l=1}^{L}\sum_{t=1}^{T^{(l)}} P(q_t^{(l)} = i\ /\ o^{(l)}, \lambda^{(\mathbf{n})})} \tag{22}$$

La méthode EM résout donc bien le système (14 - 17) sous une forme itérative de type "point-fixe" : $x^{(\mathbf{n+1})} = f(x^{(\mathbf{n})})$ en assurant que la vraisemblance des paramètres augmente à chaque étape. Le calcul des membres de dorite des équations associées nécessite bien de connaître

$$P(q_t^{(l)} = i\ /\ o^{(l)}, \lambda^{(\mathbf{n})}) \quad \text{et} \quad P(q_t^{(l)} = i,\ q_{t+1}^{(l)} = j\ /\ o^{(l)}, \lambda^{(\mathbf{n})}) \quad \text{(inférence)}$$

à chaque étape, ce qui se fait grâce au calcul des variables backward-forward associées à chaque séquence d'observations $o^{(l)}$ pour les valeurs courantes des paramètres du modèle. Ce calcul étant lui-même mené grâce aux formules de récursion (12) et (13) . On perçoit bien ici la complexité (au moins computationnelle) de l'ensemble des calculs mis en jeu, bien que les formules associées soient analytiquement exactes [6] dans le cadre de l'algorithme EM et pour le cas des HMMs. Il faut également noter l'importance de l'initialisation des paramètres [7] dans ce type d'algorithme puisque l'optimum atteint est local.

[6] Ceci est complètement différent dans le cadre des Champs de Markov en Traitement d'Images, où la phase d'Estimation de l'EM ne peut en général être menée de façon exacte, et doit se faire par simulation pour les valeurs courantes des paramètres.

[7] Voir remarques dans la conclusion de ce chapitre Section 9.

8 Recherche de la segmentation optimale

Rappelons ce qui est connu :

- une séquence d'observations o, de durée T.
- un modèle λ.

On cherche à trouver la segmentation optimale, c'est-à-dire une séquence d'états optimale, dans un sens à préciser, associée à la suite d'observations. Nous verrons dans la section suivante l'application pratique de la résolution de ce problème. Deux possibilités s'offrent alors :

a) on peut choisir choisir l'état q_t le plus probable de façon **locale** :
il s'agit de l'estimateur du Maximum de la Marginale a Posteriori (MPM). En effet d'après tout ce que l'on a vu précédemment on peut calculer exactement :

$$P(q_t = i \ / \ o,\ \lambda) = \frac{\alpha_t(i)\ \beta_t(i)}{P(o \ / \ \lambda)} \Rightarrow \boxed{\widehat{q_t} = \arg\max_{i \in E} \alpha_t(i)\ \beta_t(i)}$$

b) on peut choisir un chemin optimal **global** $\widehat{q}$ le plus probable :
il s'agit de l'estimateur du Maximum a Posteriori (MAP). En effet le théorème de Bayes nous permet d'affirmer que :

$$P(q \ / \ o,\ \lambda) = \frac{P(o \ / \ q,\ \lambda)\ P(q \ / \ \lambda)}{P(o \ / \ \lambda)}$$

La segmentation optimale au sens du MAP est donc telle que

$$\boxed{\widehat{q} = \arg\max_q P(q \ / \ o,\ \lambda) = \arg\max_q P(o \ / \ q,\ \lambda)\ P(q \ / \ \lambda) = \arg\max_q P(o,\ q \ / \ \lambda)}$$

$$= \arg\max_q \pi_{q_1}\ b_{q_1}(o_1)\ a_{q_1 q_2}\ b_{q_2}(o_2) \ldots a_{q_{T-1} q_T}\ b_{q_T}(o_T)$$

Le logarithme de la quantité à maximiser

$$\log P(o \ / \ q,\ \lambda)\ P(q \ / \ \lambda)$$

s'écrit donc comme somme de termes :

- à une seule variable $\Phi(q_t) = \log b_{q_t}(o_t)$ associés à l'attache aux données (loi d'observation).
- à deux variables consécutives $\Psi(q_t, q_{t+1}) = \log a_{q_t q_{t+1}}$ associés à la chaîne sous-jacente (probabilités de transition).

$\Rightarrow$ c'est donc ici que la programmation dynamique s'impose comme méthode particulièrement adaptée à la tache de segmentation proposée.

9 Récapitulation pour les HMM

Procédons pour conclure à une récapitulation de l'approche statistique dans la modélisation et l'utilisation des HMMs, en rapport avec les trois problèmes envisagés plus haut Section 5.

9.1 apprentissage d'un modèle (λ_u)

Rappelons que nous avons décrit la méthode EM qui consiste dans les étapes suivantes :

1) initialisation des paramètres du modèle $[\ \pi_i, a_{ij}, \mu_i, \sigma_i\ ]^{(\mathbf{0})}$:
- en ce qui concerne la chaîne cachée on introduit alors souvent un instant initial "virtuel" $t = 0$ et un état initial "déterministe" $i = 0$ ($\pi_0 = 1$), en imposant :

- soit un modèle ergodique : $a_{01} = \dfrac{1}{M}$ uniforme [8] , π_i et a_{ij} également uniformes.
- ou un modèle gauche-droite : $a_{01} = 1$, de sorte que l'état à $t = 1$ est $i = 1$. Pour les valeurs des états courants on ne retient que les coefficients : a_{ii} et $a_{i\ i+1} = 1 - a_{ii}$.

- en ce qui concerne la loi d'observation supposée gaussienne, on choisit souvent des moyennes μ_i équi-réparties dans un domaine admissible d'observations, et des variances $(\sigma_i)^2$ en conséquence.

2) à l'itération (n) : on utilise les variables backward-forward $\alpha_t(i)$ et $\beta_t(i)$ qui doivent être calculées pour chaque séquence d'apprentissage $o^{(l)}$.

$\Rightarrow$ On a donc résolu ici le problème (3) .

9.2 reconnaissance : observation

On dispose d'une séquence d'obervations $o^{(r)}$ dite "de test". On veut calculer le "score" du modèle λ_u pour cette observation. On procéde pour cela en calculant la vraisemblance du modèle pour la séquence d'obervations de test :
a) soit de façon exacte d'après (7) :

$$P(o^{(r)}\ /\ \lambda_u) = \sum_{i \in M_u} \alpha_t(i)\ \beta_t(i) \qquad \text{(avec } t = 1 \text{ ou } T^{(r)} \text{ très souvent)}$$

grâce au calcul encore une fois des variables backward-forward $\alpha_t(i)$ et $\beta_t(i)$ associées à l'observation $o^{(r)}$ et au modèle λ_u.

$\Rightarrow$ On a donc utilisé ici la résolution du problème (1) .
b) soit par l'approximation de Viterbi : on ne retient dans l'expression de la vraisemblance cherchée que la séquence d'états cachés optimale *a posteriori*

$$P(o^{(r)}\ /\ \lambda_u) \ = \ \sum_q P(o^{(r)},\ q\ /\ \lambda_u) \approx P(o^{(r)},\ \widehat{q}\ /\ \lambda_u)$$

$$\text{avec } \widehat{q} \ = \ \arg\max_q P(o^{(r)},\ q\ \ /\ \lambda) \ \ \text{(segmentation optimale MAP)}$$

On emploie donc pour cela la programmation dynamique vue dans la section précédente, puis l'on calcule ensuite facilement $P(o^{(r)},\ \widehat{q}\ /\ \lambda_u) = P(o^{(r)}\ /\ \widehat{q},\ \lambda_u)\ P(\widehat{q},\ \lambda_u)$
(ou les logarithmes de chacun de ces termes) pour la séquence optimale $\widehat{q}$ ainsi trouvée.

$\Rightarrow$ On a donc utilisé ici la résolution du problème (2) .

[8] ou a_{0i} si on part de l'état i au temps $t = 1$ de façon certaine.

Références

[Cornuéjols and Miclet(2002)] A. Cornuéjols and L. Miclet. *Apprentissage artificiel. Concepts et Algorithmes*. Eyrolles, 2002.

[Meyn and Tweedie(1993)] S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, London, 1993.

[Rabiner(1989)] L. Rabiner. A tutorial on Hidden Markov Models and selected applications in Speech Recognition http://www.ai.mit.edu/courses/6.867-f02/papers/rabiner.pdf. *Proceedings of the IEEE*, 77(2):257–285, 1989.

[Redner and Walker(1984)] R.A. Redner and H.F. Walker. Mixture Densities, Maximum Likelihood and the EM algorithm. *SIAM Review*, 26:195–239, 1984.

CHAPITRE 5 : MODELES CONNEXIONNISTES - PERCEPTRONS MULTICOUCHES

Alain Grumbach

1. Architecture d'un réseau connexionniste

1.1 Historique

Cette approche est une résurgence d'un courant qui a marqué les travaux en Intelligence Artificielle et en Psychologie des années 60, dont la réalisation la plus marquante a été le Perceptron [Rosenblatt 62]. Elle est fondée sur le comportement de réseaux de processeurs élémentaires interconnectés. Chaque processeur peut être considéré comme une simplification extrême d'un neurone, ce que Mc Culloch et Pitts (1943) ont appelé un "neurone formel".

La première réalisation majeure : Perceptron [Rosenblatt 62] fut sujette à des critiques, en particulier de Minsky et Papert qui, irrités essentiellement par le succès du Perceptron, s'attachèrent à mettre en évidence des limites théoriques et pratiques de celui-ci (problèmes non modélisables par cette approche, difficulté de généralisation à une structure comportant plusieurs niveaux, explosion combinatoire). Suite à cette critique, d'une scientificité étriquée dans la mesure où elle n'envisageait pas des extensions possibles de l'approche, elle fut quasiment abandonnée à la fin des années 60.

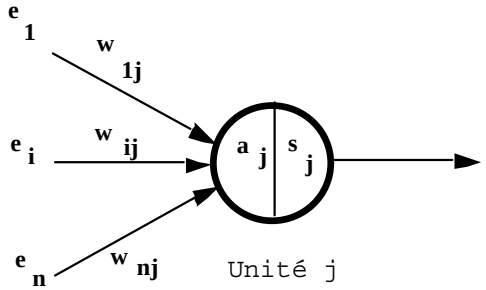
Depuis le début des années 80, on assiste à une renaissance de l'approche connexionniste, que l'on peut expliquer par l'intérêt marqué pour le parallélisme d'une part, l'apprentissage de l'autre, ainsi que par le développement technologique qui rend possible la simulation voire la réalisation de machines correspondantes, avec des capacités respectables.

1.2 Structure d'un réseau connexionniste

1.2.1 Une unité : le "neurone formel"

Une unité du réseau est un processeur élémentaire, que l'on peut représenter par une **fonction de transition** f, linéaire ou non, permettant de calculer la sortie de l'unité en fonction de ses entrées. Parmi ces unités se trouvent des unités particulières, qui réalisent l'interface avec l'environnement : **unités d'entrée et de sortie**.

Les connexions réalisent le lien entre les unités. Une (ou deux) **pondération** w_{ij} leur est associée.



Une transition concernant une unité j peut être décrite par des formules du type :

e_i : sortie de l'unité i, devenant une entrée pour l'unité j;
s_j : sortie de l'unité j

L'**activité** de l'unité j est alors donnée par : $a_j = \Sigma_i w_{ij} e_i$

et sa **sortie** par : $s_j = f_j(a_j)$

Exemples :

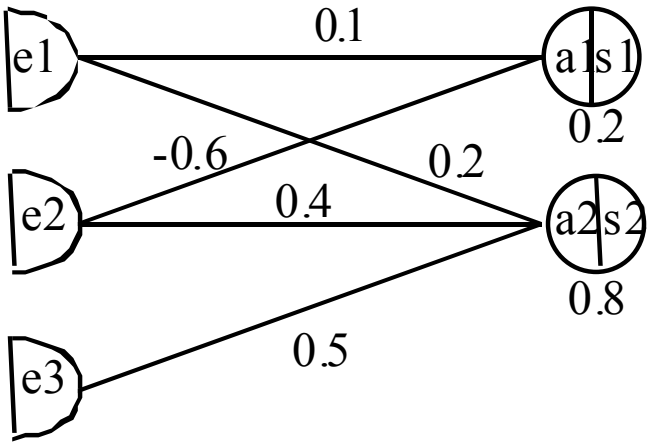
* f_j fonction identité : $s_j = a_j = \Sigma_i w_{ij} e_i$
* f_j fonction à seuil :

  $s_j = 0$ si $a_j \leq$ seuil , $s_j = 1$ si $a_j >$ seuil
* f_j fonction continue dérivable proche de la fonction à seuil : sigmoïde

  par exemple : $s_j = (e^{k(a_j-\sigma_j)}-1) / (e^{k(a_j-\sigma_j)}+1)$ σj étant un seuil local à une unité

1.2.2 Un réseau complet

Considérons l'exemple suivant :



L'information entre à gauche (e1, e2, e3), transite sur les connexions puis sort à droite (s1, s2).
Les informations figurant sur les connexions sont les poids, celles figurant sous les unités sont les seuils.

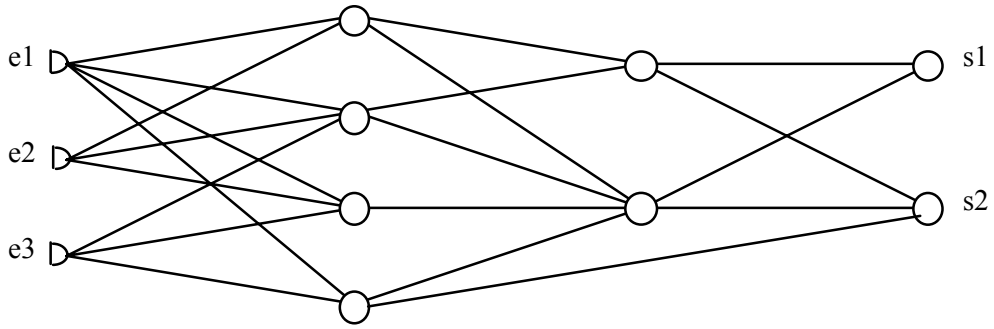
2. Perceptrons

2.1 Réseaux à deux couches d'unités : Perceptrons (simples)

Les Perceptrons (ou "Perceptrons simples") sont des réseaux caractérisés par des connexions directes entre unités d'entrée et unités de sortie. Le réseau du §1.2.2 est un Perceptron. Pour une fonction f à seuil sur [0,1], et un vecteur d'entrée : 0;1;1, on obtient un vecteur de sortie : 0; 1.

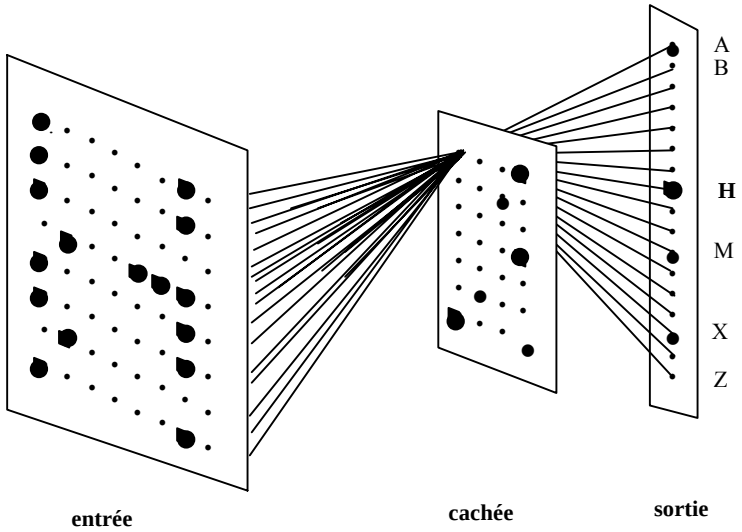
2.2 Réseaux à plusieurs couches d'unités : Perceptrons multi-couches

La puissance d'expression et d'apprentissage des réseaux du 1er type étant limitée [Minsky et Papert 69], des couches d'unités intermédiaires sont introduites :



En général, la connectivité de ce type de réseau est totale inter-couches adjacentes, nulle sinon. L'exemple de réseau de reconnaissance de caractères décrit en tête de cette présentation entre dans cette catégorie.

Un exemple est le réseau suivant dont la fonction est de reconnaître des caractères.



Chaque point représente une unité. Celles-ci sont regroupées en couches interconnectées. Les informations en entrée sont des points correspondant à une image d'un caractère (H bruité). L'information

en sortie indique le caractère reconnu : majoritairement le symbole H. Une couche intermédiaire dite "cachée" complète la structure. Chaque unité de cette couche cachée est connectée à chaque unité d'entrée et chaque unité de sortie. A chaque connexion est associé un poids, qui constitue la "mémoire" du réseau. Chaque unité effectue un traitement élémentaire de l'information fournie en entrée.

3. Techniques d'apprentissage

Dans la majorité des cas, la mise en oeuvre d'un réseau consiste, dans une première phase à apprendre à associer un ensemble d'entrées avec un ensemble de sorties (pixels du caractère H avec la lettre H), par modification des poids des connexions. En phase d'exploitation, le réseau ainsi configuré traite les données fournies en entrée, calcule une sortie associée. Il est alors capable de reconnaître des formes apprises ou des formes voisines de celles-ci, ou de leur associer d'autres informations.

3.1 Règle delta

L'apprentissage des réseaux de type Perceptron (simple), suivant la règle delta, s'effectue ainsi :

0. Elaboration de la base d'exemples : couples {entrée X - sortie désirée associée Z}

00. Initialisation des poids (valeurs aléatoires par exemple)

1. Sélection d'un exemple (ordre ou autre critère)

2. Calcul de la sortie Y élaborée par le réseau

3. Modification des poids suivant la règle delta : $\Delta w^i_{jk} = nu\ (z^i_k - y^i_k)\ x^i_j$

4. Itération des étapes 1 à 3 sur la base d'exemples (une époque)

5. Calcul de l'erreur moyenne sur la base d'exemples : $E = \Sigma_i \Sigma_k (z^i_k - y^i_k)^2$
(variante : sur une base de test)

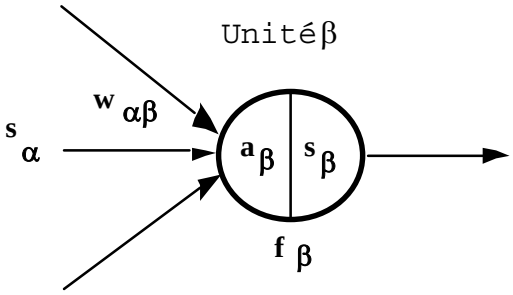
6. Si erreur E > seuil donné, itération des étapes 1 à 6; sinon fin.

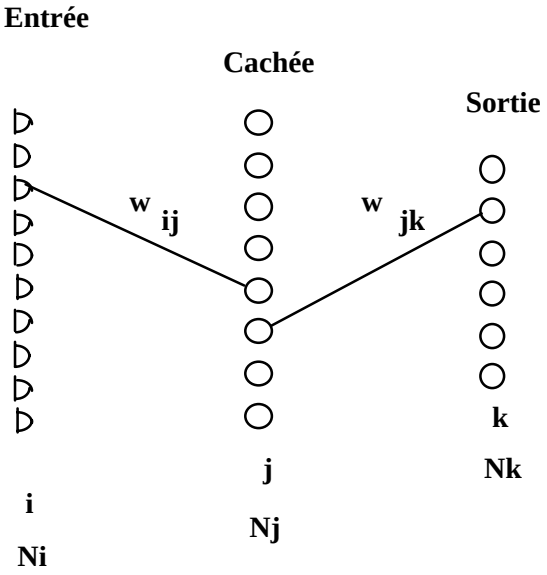
3.2 Apprentissage des Perceptrons multi-couches

L'algorithme étudié ci-dessus n'est applicable qu'à des Perceptrons simples (sans unités cachées). Il a été repris et adapté à des réseaux à plusieurs couches : méthode de **rétro-propagation de gradient** due simultanément à Le Cun, Rumelhart, et Parker (1985).

Rappelons que la fonction de transfert est du type : non linéaire mais continue, dérivable : sigmoïde.

Considérons un réseau à 3 couches d'unités, dont 1 couche d'unités "cachées".





L'algorithme d'apprentissage est le suivant :

1- présentation d'un vecteur d'entrée de la base d'exemples : e (implicite dans les formules des pas 2 à 5)

2- calcul de la sortie s :

$a_\beta = \Sigma_\alpha w_{\alpha\beta} s_\alpha$ $\quad (\alpha,\beta) = (i,j)$ puis (j,k)

$s_\beta = f_\beta(a_\beta)$ $\quad f_\beta$: fonction sigmoïde, par exemple :

$f_\beta(x) = (e^{(x-\sigma_\beta)} -1)/ (e^{(x-\sigma_\beta)} +1)$

3- présentation du vecteur de sortie désirée d :

calcul des gradients :

$\delta_k = 2 f'_k (a_k) \ (s_k - d_k)$ pour les cellules de sortie

$\delta_j = f'_j (a_j) \ \Sigma_k \ \delta_k \ w_{jk}$ pour les cellules cachées

4- modification des poids :

$\Delta w_{\alpha\beta} = - nu \ \delta_\beta \ s_\alpha \quad (\alpha,\beta) = (j,k)$ puis (i,j)

5- itération des pas 1 à 4 sur la base d'exemples

6- itération des pas 1 à 5 tant que l'erreur totale sur la base d'exemples :

$E = \Sigma_e \Sigma_k (s(e)_k - d(e)_k)^2$

est supérieure à un seuil donné (et tant que le temps limite alloué n'est pas atteint).

Pour calculer automatiquement les seuils σ_β en même temps que les poids, on remarque que : $\Sigma_\alpha w_{\alpha\beta} s_\alpha - \sigma_\beta$ peut s'écrire : $\Sigma_\alpha w_{\alpha\beta} s_\alpha + \sigma_\beta .(-1)$, càd que σ_β peut être considéré comme le poids d'une connexion reliée à une entrée supplémentaire fixe à -1. Une unité de ce type doit être ajoutée à la couche d'entrée et une autre à la couche cachée. Chacune est reliée à toutes les unités de la couche suivante, avec des "poids" qui sont les seuils des unités de cette couche suivante.

Bibliographie

Parallel Distributed Processing (3 vol.)
D. Rumelhart et J. MacClelland
MIT Press,1986

E. Davalo et P. Naïm
Des réseaux de neurones
Eyrolles, 1989