

## Sprachverarbeitung: Übung 21

### Trainieren diskreter Hidden-Markov-Modelle

Mit dem Baum-Welch-Algorithmus kann ausgehend von einem Initial-HMM  $\lambda^{(0)}$  für eine Beobachtungssequenz  $\mathbf{X}$  ein neues HMM  $\lambda^{(1)}$  geschätzt werden, wobei für die A-posteriori-Wahrscheinlichkeiten im Allgemeinen gilt:  $P(\mathbf{X}|\lambda^{(0)}) < P(\mathbf{X}|\lambda^{(1)})$ . Der Baum-Welch-Algorithmus kann somit so lange iteriert werden, bis  $P(\mathbf{X}|\lambda^{(i)})$  nur noch unwesentlich zunimmt.

Weil jedoch eine einzelne Beobachtungssequenz nicht genügend Information über die inhärente Statistik<sup>1</sup> liefert, muss eine grössere Menge von Beobachtungssequenzen  $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_S\}$  verwendet werden. Die Resultate, welche der Baum-Welch-Algorithmus für jede Beobachtungssequenz ermittelt, müssen miteinander verrechnet werden, indem über alle Beobachtungssequenzen aufsummiert wird, wie oft sich das HMM im Zustand  $S_i$  befindet, wie oft es vom Zustand  $S_i$  in den Zustand  $S_j$  übergeht und wie oft es im Zustand  $S_j$  die Beobachtung  $c_k$  ausgibt.

Es werden also nicht die  $a_{ij}$  der einzelnen Beobachtungssequenzen miteinander verrechnet, sondern die Zählwerte, aus denen diese zu berechnen sind. Das Analoge gilt auch für die Beobachtungswahrscheinlichkeiten  $b_j(k)$ . Erst wenn die Zählwerte über alle Beobachtungssequenzen aufsummiert sind, werden die neuen Schätzwerte  $\tilde{a}_{ij}$  und  $\tilde{b}_j(k)$  gemäss den Formeln (114) bis (117) im Buch ermittelt.

#### Aufgabe 1: Baum-Welch-Algorithmus für eine Beobachtungssequenz

Schreiben Sie für den diskreten Baum-Welch-Algorithmus eine Matlab-Funktion, welche nicht die  $a_{ij}$  und die  $b_j(k)$  ausgibt, sondern die oben erwähnten Zählwerte, also die folgende Form hat: `[ns,nt,nb] = discr_baum_welch_alg(a,b,X)`. Die Eingangsargumente der Funktion sind das durch die Matrizen **a** und **b** gegebene HMM und die Beobachtungssequenz **X**. Die Ausgabeargumente der Funktion haben die folgende Bedeutung: **ns(i)** ist die erwartete Anzahl Male im Zustand  $S_i$  (vergl. Formel 112 im Buch), **nt(i,j)** ist die erwartete Anzahl Übergänge vom Zustand  $S_i$  in den Zustand  $S_j$  (vergl. Formel 113) und **nb(j,k)** ist die erwartete Anzahl Male im Zustand  $S_j$  mit der Beobachtung  $c_k$  (Zähler von Formel 117).

Die Matlab-Funktionen `discr_forward_alg.m` und `discr_backward_alg.m` zum Ermitteln der Vorwärtswahrscheinlichkeiten  $\alpha_t(i)$  bzw. der Rückwärtswahrscheinlichkeiten  $\beta_t(i)$  stehen zur Verfügung. Daraus können die Hilfwahrscheinlichkeiten  $\gamma_t(i)$  und  $\xi_t(i,j)$  gemäss den Formeln (108) und (111) berechnet werden und mit diesen schliesslich die Zählwerte:

<sup>1</sup>Die hier verwendeten Beobachtungssequenzen werden mit einem DDHMM erzeugt und können, wie Übung 19 gezeigt hat, für ein gegebenes DDHMM sehr verschieden aussehen. Erst aufgrund einer genügend grossen Menge von Beobachtungssequenzen kann beispielsweise darauf geschlossen werden, wie oft im Mittel vom Zustand  $S_i$  in den Zustand  $S_j$  gewechselt wird.

$$ns(i) = \sum_{t=1}^T \gamma_t(i) , \quad nt(i, j) = \sum_{t=1}^{T-1} \xi_t(i, j) , \quad nb(j, k) = \sum_{t \text{ mit } \mathbf{x}_t = c_k} \gamma_t(j)$$

Diese Zählwerte entsprechen den Zählern bzw. den Nennern der Formeln (114) bis (117), wo die Zustände  $S_1$  und  $S_N$  teilweise als Spezialfälle betrachtet werden müssen. Überlegen Sie deshalb anhand dieser Formeln und des Trellis-Diagramms, wie die Zählwerte für diese Fälle zu bestimmen sind. Beispielweise hat die Gleichung (114) auf der rechten Seite keinen Nenner, im Gegensatz zur Gleichung (115). Dies ist deshalb so, weil ein HMM, wenn es eine Beobachtungssequenz generiert, genau einmal im Zustand  $S_1$  ist. Der besagte Nenner wäre also 1 und wird somit nicht geschrieben. Um  $S$  Beobachtungssequenzen zu erzeugen, muss das HMM jedoch  $S$  Mal im Zustand  $S_1$  sein, wodurch  $ns(1) = S$  wird. Überlegen Sie die weiteren Spezialfälle analog.

Wenn Sie die Funktion für den Baum-Welch-Algorithmus geschrieben haben, dann können Sie sie mit dem Matlab-Skript `ueb21_1` testen.

## Aufgabe 2: Baum-Welch-Training mit mehreren Beobachtungssequenzen

Schreiben Sie eine Matlab-Funktion `[a,b] = discr_baum_welch_training(Xset,N,M,K)`, welche ausgehend von einem Initial-DDHMM für einen Satz von Beobachtungssequenzen `Xset` die Zustandsübergangs-Wahrscheinlichkeiten `a` ( $N \times N$ -Matrix) und die Beobachtungswahrscheinlichkeiten `b` ( $N \times M$ -Matrix) ermittelt. Die Funktion soll  $K$  Trainings-Iterationen durchführen. Die Beobachtungssequenzen sind in einem Cell Array gespeichert; die  $i$ -te Beobachtung der  $j$ -ten Sequenz wird also mit `Xset{j}(i)` adressiert. Das für die Iteration benötigte Initial-DDHMM mit  $N$  Zuständen (inklusive Anfangs- und Endzustand) und  $M$  diskreten Beobachtungssymbolen kann mit der Matlab-Funktion `[a,b] = init_ddhmm(N,M)` erzeugt werden.

Um die realisierte Funktion zu überprüfen, können Sie das Matlab-Skript `ueb21_2` aufrufen, das Ihnen mitteilen wird, ob Sie die Aufgabe 2 als korrekt gelöst betrachten und zur nächsten Aufgabe gehen dürfen.

Hinweis: Verwenden Sie die Matlab-Funktion `discr_baum_welch_alg`, die Sie in Aufgabe 1 realisiert haben, um für einzelne Beobachtungssequenzen die Zählwerte zu ermitteln und summieren Sie diese auf (siehe auch Erläuterung am Anfang der Übung).

## Aufgabe 3: Trainieren von Wort-DDHMM

In dieser Aufgabe werden mit einem DDHMM  $\lambda$  Beobachtungssequenzen generiert und daraus wiederum ein DDHMM  $\tilde{\lambda}$  trainiert. Naheliegenderweise wird nur dann  $\lambda \approx \tilde{\lambda}$  sein, wenn die Zahl  $S$  der Beobachtungssequenzen genügend gross ist, und wenn die Anzahl  $K$  der Iterationen des Baum-Welch-Trainings genügend hoch ist.

Mit der Matlab-Funktion `ueb21_3(Ind,S,K,initRand)` können Sie durch Experimentieren die Anzahl der Beobachtungssequenzen  $S$  (Grösse des Trainingssets) und die Zahl der Iterationen  $K$  so festlegen, dass sie möglichst klein sind, aber das trainierte DDHMM einigermaßen dem generierenden entspricht (d.h. nur kleine Abweichung der betreffenden Modell-Parameter). Mit

dem Argument `Ind` können eines oder mehrere DDHMM ausgewählt werden und das Argument `initRand` steuert die Initialisierung des Zufallsgenerators.<sup>2</sup>

Bestimmen Sie die optimalen Werte für  $S$  und  $K$  für die DDHMM 1 und 2 und erklären Sie die gefundenen Unterschiede, einerseits zwischen dem generierenden und dem trainierten Modell und andererseits zwischen den DDHMM 1 und 2.

Wählen Sie nun die Parameter  $S$  und  $K$  so, dass sie für die DDHMM 1 bis 4 passen. Wenn Sie dafür die Matlab-Funktion `ueb21_3` benutzen, dann werden die trainierten DDHMM automatisch in der Datei `ddhmm_s_trained.mat` gespeichert und stehen damit für die nächste Aufgabe zur Verfügung.

#### Aufgabe 4: Testen der trainierten Wort-DDHMM

Mit dem Matlab-Skript `ueb21_4` können Sie die trainierten DDHMM testen: Es werden mit den ursprünglichen DDHMM Beobachtungssequenzen generiert und diese mit den trainierten DDHMM erkannt. Wie Sie feststellen werden, ist die Anordnung ähnlich wie in Aufgabe 3 in Übung 20, die erkennenden DDHMM sind jedoch durch die neu trainierten ersetzt.

Auch hier ist festzustellen, dass für eine mit einem bestimmten DDHMM generierte Beobachtungssequenz nicht immer das zugehörige trainierte DDHMM die höchste Forward- oder Viterbi-Wahrscheinlichkeit zeigt. Die Verwechslungen sind in der Regel umso häufiger, je schlechter die DDHMM trainiert worden sind.

Interessant ist, dass nun der Fall auftreten kann, dass die Summe der Erkennungsraten nicht immer 100 % ergibt. Dies ist z.B. der Fall, wenn das Training mit `ueb21_3([1 2 3 4 6],50,10,1)` durchgeführt wird und `ddhmm(3)` beim Testen als generierendes Modell gebraucht wird. Woher rührt dies und wie lässt sich dieser Effekt vermeiden?

In Übung 20 haben wir festgestellt, dass bei der Erkennung mit den Modellen `ddhmm(1)`, `ddhmm(3)` und `ddhmm(4)` für den Forward- und den Viterbi-Algorithmus stets dieselbe Wahrscheinlichkeit resultiert. Warum ist dies hier nicht mehr der Fall?

---

<sup>2</sup>Wenn das Argument `initRand` vorhanden und gleich 1 ist, dann wird der Matlab-interne Zufallsgenerator mit `rand('state',0)` initialisiert. So liefert die Funktion `ueb21_3(Ind,S,K,1)` beim Aufruf mit festen Argumenten stets die gleichen Resultate. Andernfalls generiert die Matlab-Funktion `ueb21_3` bei jedem Aufruf ein anderes Trainingsset und die trainierten Modelle werden jedesmal etwas anders sein, insbesondere bei kleinen Trainingssets.