550, rue Sherbrooke Ouest, bureau 100
Montréal (Québec) H3A 1B9
Tél. : (514) 840-1234 ; Téléc. : (514) 840-1244
888, rue St-Jean, bureau 555
Québec (Québec) G1R 5H6
Tél. : (418) 648-8080 ; Téléc. : (418) 648-8141
http ://www.crim.ca

CRIM - Documentation/Communications

E-Inclusion Core Speech
Forward-Backward Algorithm

Technical report

CRIM-06/05-05

Patrick Cardinal
Research agent
Speech recognition group

May 2006

Collection scientifique et technique

ISBN-13: 978-2-89522-074-9
ISBN-10: 2-89522-074-3

Table des matières

1 Introduction

This chapter describes the algorithm used to compute the probability of observing an observation o_t at time t given the lattice. This algorithm is called forward-backward.

2 Lattice

A lattice is a WFSA (Weighted Finite-State Acceptor) in which the transitions carry a distribution symbol d and the log probability that the observation has been generated by this model. The following figure shows an example of such a lattice.

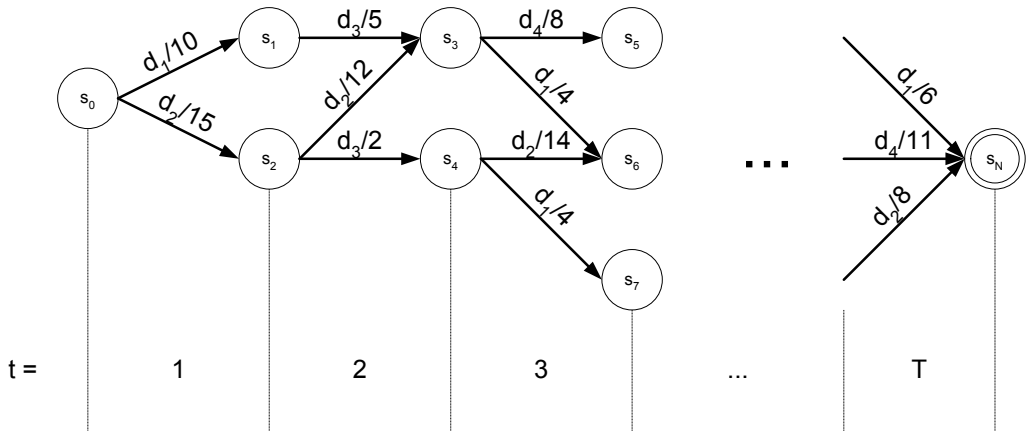

FIG. 1. ***Example of a lattice.***

3 Forward-Backward probabilities

From a given lattice, we want to compute the probability of going from state i to state j at time t. This transition is denoted by a_{ij}. This probability is denoted by $\xi_{ij}(t)$ and is formally defined as :

$$\begin{aligned}
\xi_{ij}(t) &= P(a_t = a_{ij}|y_1, y_2, \cdots, y_t) \\
&= \frac{P(y_1, y_2, \cdots, y_t|a_t = a_{ij})P(a_t = a_{ij})}{P(y_1, y_2, \cdots, y_t)} \\
&= \frac{P(y_1, y_2, \cdots, y_t, a_t = a_{ij})}{P(y_1, y_2, \cdots, y_t)}
\end{aligned}$$

Where a_t represents a transition at time t, y_t is the observation at time t and T is the total time. In the case of lattices (which are acyclic), $\xi_{ij}(t) = 0$ if the transition from state i to j is not at time t.

The algorithm used to compute $\xi_{ij}(t)$ is called the forward-backward algorithm. Since a lattice is an acyclic Markov chain, a simplified version of the algorithm can be used to compute this value. The first part of the algorithm, called the forward pass, computes iteratively the joint probability of the observation sequence $y_1, y_2, \cdots, y_t$ and being in transition a_{ij} at time t :

$$\begin{aligned} \alpha_{ij}(t) &= P(a_t = a_{ij}, y_1, y_2, \cdots, y_t) \\ &= (\sum_{k=0}^{N} \alpha_{ki}(t-1)) b_{ij}(y_t) \end{aligned}$$

where $b_{ij}(y_t)$ is the probability that the observation has been generated by the distribution carried by the transition a_{ij}, N is the number of states in the lattice and $\alpha_{ki}(t-1) = 0$ if there is no path from the initial state to state i containing the state k.

The value of α at time $t = 1$ is the probabilty of generating y_1 and thus, is simply the cost of the transition :

$$\alpha_{ij}(1) = b_{ij}(y_1)$$

The second part, called the backward pass, computes the probability of generating the remaining observation sequence $y_{t+1}, y_{t+2}, \cdots, y_T$ and is defined as :

$$\begin{aligned} \beta_{ij}(t) &= P(y_{t+1}, y_{t+2}, \cdots, y_T | a_t = a_{ij}) \\ &= \sum_{k=0}^{N} \beta_{jk}(t+1) b_{ij}(y_{t+1}) \end{aligned}$$

with the initial value set to :

$$\beta_{ij}(T) = 1$$

From these values, we can compute the probability $\xi_{ij}(t)$ for all t using the following formula :

$$\xi_{ij}(t) = \frac{\alpha_{ij}(t) \cdot \beta_{ij}(t)}{\alpha_F(T)}$$

where $\alpha_F(T)$ is the sum of $\alpha_{ij}(T)$ for which j is a final state of the lattice.

From the $\xi_{ij}(t)$, we can obtain $\gamma_d(t)$, the probability that the observation y_t was generated by the distribution d at time t. This value is obtained by summing all transitions at time t carrying the same distribution symbol :

$$\gamma_d(t) = \sum_{i \in s[t] \wedge j \in s[t+1] \wedge sym[a_{ij}]=d} \xi_{ij}(t)$$

Where $s[t]$ is the set of states at time t and $sym[a]$ is the distribution symbol carried by the transition a.

3.1 Implementation

The computation of α and β is obtained by performing a Breadth First Search (BFS) algorithm on the lattice. In addition to the distribution symbol and the cost, each transition is associated with its α and β values that are updated during the BFS.

The algorithm for computing α is described here.

Algorithm 1 ForwardPass

ComputeForward(Lattice)

```
1:  Queue ← ∅
2:  for all q ∈ Q do
3:      visited[q] ← false
4:      accum[q] ← 0
5:  Queue ← intialState[FST]
6:  visited[intialState[FST]] ← true
7:  while Queue ≠ ∅ do
8:      q ← Head[Queue]
9:      for each transition t going out state q do
10:         α ← cost[t] · accum[q]
11:         accum[to[t]] ← accum[to[t]] + α
12:         forward[t] ← α
13:         if visited[to[t]] = false then
14:             Queue ← to[t]
15:             visited[to[t]] ← true
```

The algorithm works as follows. The loop at lines 2-4 initialize the search. The loop starting at line 7 will be executed while there are remaining states to explore . In this loop, each transition of the explored state is visited (lines 9-15). Line 10 computes $\alpha(t)$ for the current transition. Recall that this value is defined as :

$$\alpha_{ij}(t) = (\sum_{k=0}^{N} \alpha_{ki}(t-1))b_{ij}(y_t)$$

In this implementation, an accumulator representing the summation is associated to every state. Given a transition a_{ij}, the state i contains the sum of forward probabilities of all paths passing by this state. Thus, the value $\sum_{k=0}^{N} \alpha_{ki}(t-1)$ needed to compute $\alpha_{ij}(t)$ is stored in state i.

Line 11 updates the summation value for state j which will be needed to compute the α value for every transition leaving it.

Since the states are visited in topological order, all the information needed for computing the forward probabilities are available when the transition is reached.

The complexity of the algorithm depends on loops 7-15 and 9-15. The first loop will pass through every state only one time, thus is $\mathcal{O}(|Q|)$. The second loop passes through all transitions of the current state. Thus, the total complexity is :

$$\mathcal{O}(|Q| + \sum_{q \in Q} |E[q]|) = \mathcal{O}(|E| + |Q|)$$

The backward pass implementation is based on the same idea and will not be described here.

4 Results

The implementation is quite efficient. For example, the computation time of a lattice of 117000 nodes and 288000 states, generated using an audio file of 5 seconds, is 0.88 second.