HMM Algorithms

Peter Bell

Automatic Speech Recognition— ASR Lecture 3
20 January 2020

Overview

HMM algorithms

- HMM recap
- HMM algorithms
  - Likelihood computation (forward algorithm)
  - Finding the most probable state sequence (Viterbi algorithm)
  - Estimating the parameters (forward-backward and EM algorithms)

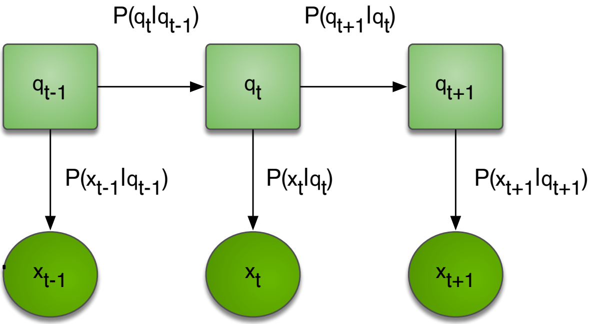
Overview

HMM algorithms

- HMM recap
- HMM algorithms
  - Likelihood computation (forward algorithm)
  - Finding the most probable state sequence (Viterbi algorithm)
  - Estimating the parameters (forward-backward and EM algorithms)

Warning: the maths continues!

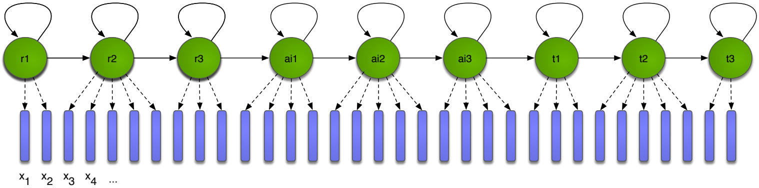
Recap: the HMM



- A *generative model* for the sequence $X = (x_1, \ldots, x_T)$
- Discrete states q_t are unobserved
- q_{t+1} is conditionally independent of $q_1, \ldots, q_{t-1}$, given q_t
- Observations x_t are conditionally independent of each other, given q_t.

HMMs for ASR

The three-state left-to-right topology for phones:



Computing likelihoods with the HMM

Joint likelihood of X and $Q = (q_1, \ldots, q_T)$:

$$P(X, Q|\lambda) = P(q_1)P(\mathbf{x}_1|q_1)P(q_2|q_1)P(\mathbf{x}_2|q_2)\ldots \quad (1)$$

$$= P(q_1)P(\mathbf{x}_1|q_1)\prod_{t=2}^{T} P(q_t|q_{t-1})P(\mathbf{x}_t|q_t) \quad (2)$$

$P(q_t)$ denotes the initial occupancy probability of each state

HMM parameters

The parameters of the model, λ, are given by:

- Transition probabilities $a_{kj} = P(q_{t+1} = j | q_t = k)$
- Observation probabilities $b_j(\mathbf{x}) = P(\mathbf{x} | q = j)$

The three problems of HMMs

Working with HMMs requires the solution of three problems:

1. **Likelihood** Determine the overall likelihood of an observation sequence $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$ being generated by a known HMM topology, $\mathcal{M}$.
   $\rightarrow$ the *forward algorithm*

The three problems of HMMs

Working with HMMs requires the solution of three problems:

1. **Likelihood** Determine the overall likelihood of an observation sequence $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$ being generated by a known HMM topology, $\mathcal{M}$.
   $\rightarrow$ the *forward algorithm*
2. **Decoding and alignment** Given an observation sequence and an HMM, determine the most probable hidden state sequence
   $\rightarrow$ the *Viterbi algorithm*

The three problems of HMMs

Working with HMMs requires the solution of three problems:

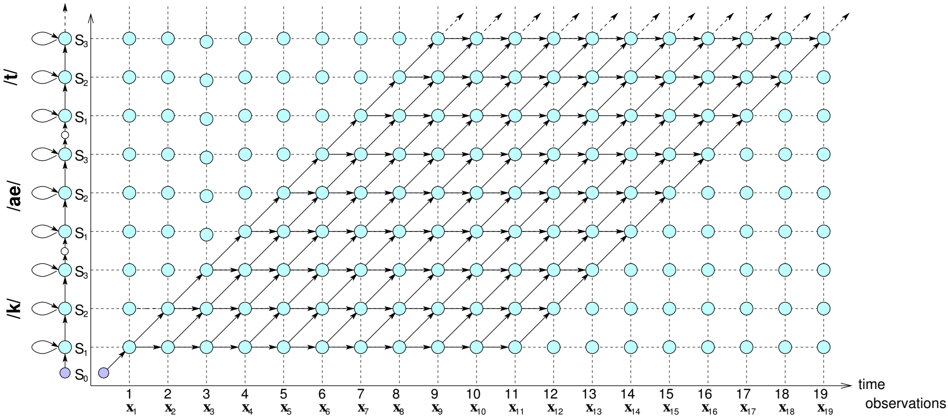
1. **Likelihood** Determine the overall likelihood of an observation sequence $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$ being generated by a known HMM topology, $\mathcal{M}$.
   $\rightarrow$ the *forward algorithm*
2. **Decoding and alignment** Given an observation sequence and an HMM, determine the most probable hidden state sequence
   $\rightarrow$ the *Viterbi algorithm*
3. **Training** Given an observation sequence and an HMM, learn the state occupation probabilities, in order to find the best HMM parameters $\boldsymbol{\lambda} = \{\{a_{jk}\}, \{b_j()\}\}$
   $\rightarrow$ the *forward-backward* and *EM* algorithms

Notes on the HMM topology

By the HMM topology, $\mathcal{M}$, we can mean:

- A restricted left-to-right topology based on a known word/sentence, leading to a "trellis-like" structure over time
- A much less restricted topology based on a grammar or language model – or something in between
- The forward/backward algorithms are not (generally) suitable for unrestricted topologies

Example: trellis for /k ae t/



1. Likelihood

- Goal: determine $p(\mathbf{X} \,|\, \mathcal{M})$

1. Likelihood

- Goal: determine $p(\mathbf{X} \mid \mathcal{M})$
- Sum over all possible state sequences $Q = (q_1, \ldots, q_T)$ that could result in the observation sequence $\boldsymbol{X}$

$$
\begin{aligned}
p(\mathbf{X}|\mathcal{M}) &= \sum_{Q \in \mathcal{Q}} P(\mathbf{X}, Q|\mathcal{M}) \\
&= P(q_1)P(\mathbf{x}_1|q_1) \prod_{t=2}^{T} P(q_t|q_{t-1})P(\mathbf{x}_t|q_t)
\end{aligned}
$$

1. Likelihood

- Goal: determine $p(\mathbf{X} \mid \mathcal{M})$
- Sum over all possible state sequences $Q = (q_1, \ldots, q_T)$ that could result in the observation sequence $\boldsymbol{X}$

$$p(\mathbf{X}|\mathcal{M}) = \sum_{Q \in \mathcal{Q}} P(\mathbf{X}, Q|\mathcal{M})$$

$$= P(q_1)P(\mathbf{x}_1|q_1) \prod_{t=2}^{T} P(q_t|q_{t-1})P(\mathbf{x}_t|q_t)$$

- How many paths Q do we have to calculate?

$$\sim \quad \underbrace{N \times N \times \cdots N}_{T \text{ times}} = N^T \qquad \begin{array}{ll} N: & \text{number of HMM states} \\ T: & \text{length of observation} \end{array}$$

e.g. $N^T \approx 10^{10}$ for $N=3$, $T=20$

1. Likelihood

- Goal: determine $p(\mathbf{X} \mid \mathcal{M})$
- Sum over all possible state sequences $Q = (q_1, \ldots, q_T)$ that could result in the observation sequence $\boldsymbol{X}$

$$p(\mathbf{X}|\mathcal{M}) = \sum_{Q \in \mathcal{Q}} P(\mathbf{X}, Q|\mathcal{M})$$

$$= P(q_1)P(\mathbf{x}_1|q_1) \prod_{t=2}^{T} P(q_t|q_{t-1})P(\mathbf{x}_t|q_t)$$

- How many paths Q do we have to calculate?

$$\sim \underbrace{N \times N \times \cdots N}_{T \text{ times}} = N^T$$

N : number of HMM states
T : length of observation

e.g. $N^T \approx 10^{10}$ for $N\!=\!3,\ T\!=\!20$

- Computation complexity of multiplication: $O(2TN^T)$

Likelihood: The Forward algorithm

The **Forward algorithm**:

- Rather than enumerating each sequence, compute the probabilities recursively (exploiting the Markov assumption)

Likelihood: The Forward algorithm

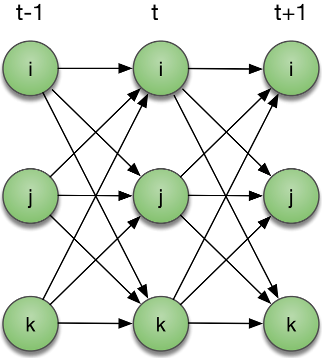
The **Forward algorithm**:

- Rather than enumerating each sequence, compute the probabilities recursively (exploiting the Markov assumption)
- Reduces the computational complexity to $O(TN^2)$

Likelihood: The Forward algorithm

The **Forward algorithm**:

- Rather than enumerating each sequence, compute the probabilities recursively (exploiting the Markov assumption)
- Reduces the computational complexity to $O(TN^2)$
- Visualise the problem as a *state-time trellis*



The forward probability

Define the *Forward probability*, $\alpha_t(j)$: the probability of observing the observation sequence $\mathbf{x}_1 \dots \mathbf{x}_t$ and being in state j at time t:

$$\alpha_j(t) = p(\mathbf{x}_1, \dots, \mathbf{x}_t, q_t = j \mid \mathcal{M})$$

We can *recursively* compute this probability

Initial and final state probabilities

It what follows it is convenient to define:

- an additional single initial state $S_I = 0$, with transition probabilities

  $$a_{0j} = P(q_1 = j)$$

  denoting the probability of starting in state j
- a single final state, S_E, with transition probabilities a_{jE} denoting the probability of the model terminating in state j.
- S_I and S_E are both *non-emitting*

1. Likelihood: The Forward recursion

- Initialisation

$$\alpha_j(0) = 1 \qquad j = 0$$
$$\alpha_j(0) = 0 \qquad j \neq 0$$

- Recursion

$$\alpha_j(t) = \sum_{i=1}^{J} \alpha_i(t-1) a_{ij} b_j(\mathbf{x}_t) \qquad 1 \leq j \leq J,\, 1 \leq t \leq T$$

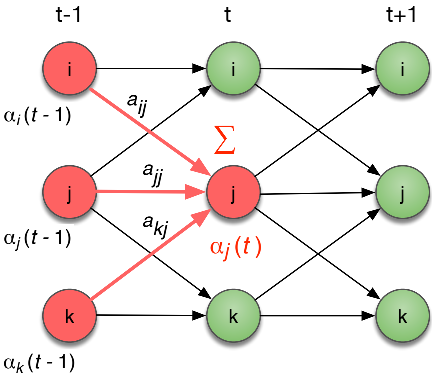
- Termination

$$p(\mathbf{X} \,|\, \mathcal{M}) = \alpha_E = \sum_{i=1}^{J} \alpha_i(T) a_{iE}$$

s_I: initial state, s_E: final state

1. Likelihood: Forward Recursion

$$\alpha_j(t) = p(\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = j | \mathcal{M}) = \sum_{i=1}^{J} \alpha_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$



Viterbi algorithm

- Instead of summing over all possible state sequences, just consider just the most probable path:

$$P^*(X|\mathcal{M}) = \max_{Q \in \mathcal{Q}} P(X, Q|\mathcal{M})$$

Viterbi algorithm

- Instead of summing over all possible state sequences, just consider just the most probable path:

$$P^*(X|\mathcal{M}) = \max_{Q \in \mathcal{Q}} P(X, Q|\mathcal{M})$$

- Achieve this by changing the summation to a maximisation in the forward algorithm recursion:

$$V_j(t) = \max_i V_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$

Viterbi algorithm

- Instead of summing over all possible state sequences, just consider just the most probable path:

$$P^*(X|\mathcal{M}) = \max_{Q \in \mathcal{Q}} P(X, Q|\mathcal{M})$$

- Achieve this by changing the summation to a maximisation in the forward algorithm recursion:

$$V_j(t) = \max_i V_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$

- If we are performing decoding or forced alignment, then only the most likely path is needed

Viterbi algorithm

- Instead of summing over all possible state sequences, just consider just the most probable path:

$$P^*(X|\mathcal{M}) = \max_{Q \in \mathcal{Q}} P(X, Q|\mathcal{M})$$

- Achieve this by changing the summation to a maximisation in the forward algorithm recursion:

$$V_j(t) = \max_i V_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$

- If we are performing decoding or forced alignment, then only the most likely path is needed
- In training, it can be used as an approximation

Viterbi algorithm

- Instead of summing over all possible state sequences, just consider just the most probable path:

$$P^*(X|\mathcal{M}) = \max_{Q \in \mathcal{Q}} P(X, Q|\mathcal{M})$$

- Achieve this by changing the summation to a maximisation in the forward algorithm recursion:

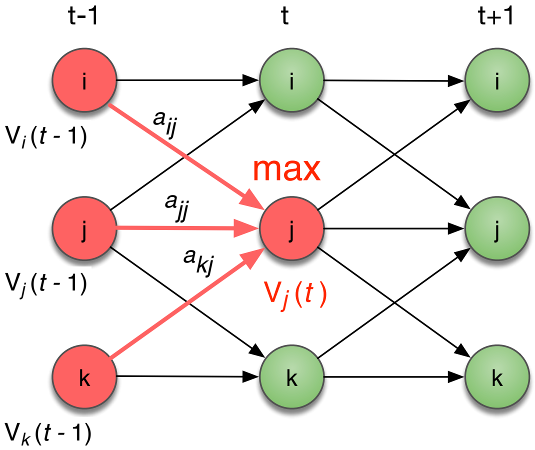
$$V_j(t) = \max_i V_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$

- If we are performing decoding or forced alignment, then only the most likely path is needed
- In training, it can be used as an approximation
- We need to keep track of the states that make up this path by keeping a sequence of *backpointers* to enable a Viterbi *backtrace*: the backpointer for each state at each time indicates the previous state on the most probable path

Viterbi Recursion

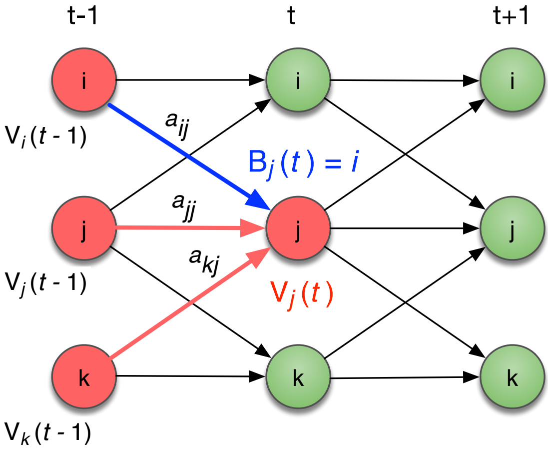
$$V_j(t) = \max_i V_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$

Likelihood of the most probable path



Viterbi Recursion

Backpointers to the previous state on the most probable path



2. Decoding: The Viterbi algorithm

- Initialisation

$$
\begin{aligned}
V_0(0) &= 1 \\
V_j(0) &= 0 \qquad \text{if } j \neq 0 \\
B_j(0) &= 0
\end{aligned}
$$

2. Decoding: The Viterbi algorithm

- Initialisation

$$V_0(0) = 1$$
$$V_j(0) = 0 \qquad \text{if } j \neq 0$$
$$B_j(0) = 0$$

- Recursion

$$V_j(t) = \max_{i=1}^{J} V_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$
$$B_j(t) = \arg\max_{i=1}^{J} V_i(t-1) a_{ij} b_j(\mathbf{x}_t)$$

2. Decoding: The Viterbi algorithm

- Initialisation

$$
\begin{aligned}
V_0(0) &= 1 \\
V_j(0) &= 0 \qquad \text{if } j \neq 0 \\
B_j(0) &= 0
\end{aligned}
$$

- Recursion

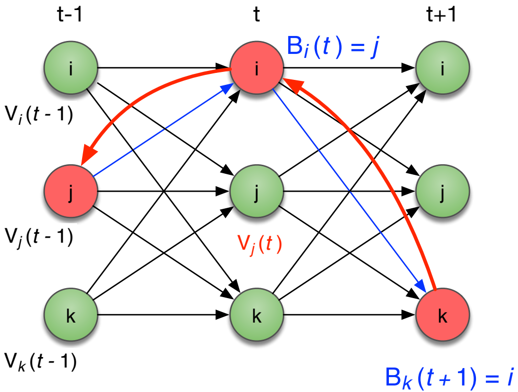
$$
\begin{aligned}
V_j(t) &= \max_{i=1}^{J} V_i(t-1) a_{ij} b_j(\mathbf{x}_t) \\
B_j(t) &= \arg\max_{i=1}^{J} V_i(t-1) a_{ij} b_j(\mathbf{x}_t)
\end{aligned}
$$

- Termination

$$
\begin{aligned}
P^* = V_E &= \max_{i=1}^{J} V_T(\,i\,) a_{iE} \\
s_T^* = B_E &= \arg\max_{i=1}^{J} V_i(T) a_{iE}
\end{aligned}
$$

Viterbi Backtrace

Backtrace to find the state sequence of the most probable path



3. Training: Forward-Backward algorithm

- Goal: Efficiently estimate the parameters of an HMM $\mathcal{M}$ from an observation sequence

3. Training: Forward-Backward algorithm

- Goal: Efficiently estimate the parameters of an HMM $\mathcal{M}$ from an observation sequence
- Assume single Gaussian output probability distribution

$$b_j(\mathbf{x}) = p(\mathbf{x} \mid j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

3. Training: Forward-Backward algorithm

- Goal: Efficiently estimate the parameters of an HMM $\mathcal{M}$ from an observation sequence
- Assume single Gaussian output probability distribution

$$b_j(\mathbf{x}) = p(\mathbf{x} \mid j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

- Parameters $\mathcal{M}$:
  - Transition probabilities a_{ij}:

$$\sum_j a_{ij} = 1$$

  - Gaussian parameters for state j:
    mean vector $\boldsymbol{\mu}_j$; covariance matrix $\boldsymbol{\Sigma}_j$

Viterbi Training

- If we knew the state-time alignment, then each observation feature vector could be assigned to a specific state

Viterbi Training

- If we knew the state-time alignment, then each observation feature vector could be assigned to a specific state
- A state-time alignment can be obtained using the most probable path obtained by Viterbi decoding

Viterbi Training

- If we knew the state-time alignment, then each observation feature vector could be assigned to a specific state
- A state-time alignment can be obtained using the most probable path obtained by Viterbi decoding
- Maximum likelihood estimate of a_{ij}, if $C(\, i \to j \,)$ is the count of transitions from i to j

$$\hat{a}_{ij} = \frac{C(\, i \to j \,)}{\sum_k C(\, i \to k \,)}$$

Viterbi Training

- If we knew the state-time alignment, then each observation feature vector could be assigned to a specific state
- A state-time alignment can be obtained using the most probable path obtained by Viterbi decoding
- Maximum likelihood estimate of a_{ij}, if $C(\, i \to j\,)$ is the count of transitions from i to j

$$\hat{a}_{ij} = \frac{C(\, i \to j\,)}{\sum_k C(\, i \to k\,)}$$

- Likewise if Z_j is the set of observed acoustic feature vectors assigned to state j, we can use the standard maximum likelihood estimates for the mean and the covariance:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{\mathbf{x} \in Z_j} \mathbf{x}}{|Z_j|}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{\mathbf{x} \in Z_j} (\mathbf{x} - \hat{\boldsymbol{\mu}}_j)(\mathbf{x} - \hat{\boldsymbol{\mu}}_j)^T}{|Z_j|}$$

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability
- *State occupation probability*: The probability $\gamma_j(t)$ of occupying state j at time t given the sequence of observations.

  Compare with component occupation probability in a GMM

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability
- *State occupation probability*: The probability $\gamma_j(t)$ of occupying state j at time t given the sequence of observations.

  Compare with component occupation probability in a GMM
- We can use this for an iterative algorithm for HMM training: the EM algorithm (whose adaption to HMM is called 'Baum-Welch algorithm')

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability
- *State occupation probability*: The probability $\gamma_j(t)$ of occupying state j at time t given the sequence of observations.
  Compare with component occupation probability in a GMM
- We can use this for an iterative algorithm for HMM training: the EM algorithm (whose adaption to HMM is called 'Baum-Welch algorithm')
- Each iteration has two steps:

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability
- *State occupation probability*: The probability $\gamma_j(t)$ of occupying state j at time t given the sequence of observations.

  Compare with component occupation probability in a GMM
- We can use this for an iterative algorithm for HMM training: the EM algorithm (whose adaption to HMM is called 'Baum-Welch algorithm')
- Each iteration has two steps:

  E-step estimate the state occupation probabilities (Expectation)

EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability
- *State occupation probability*: The probability $\gamma_j(t)$ of occupying state j at time t given the sequence of observations.

  Compare with component occupation probability in a GMM
- We can use this for an iterative algorithm for HMM training: the EM algorithm (whose adaption to HMM is called 'Baum-Welch algorithm')
- Each iteration has two steps:

  E-step estimate the state occupation probabilities (Expectation)

  M-step re-estimate the HMM parameters based on the estimated state occupation probabilities (Maximisation)

Backward probabilities

- To estimate the state occupation probabilities it is useful to define (recursively) another set of probabilities—the *Backward probabilities*

  $$\beta_j(t) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|\, q_t = j, \mathcal{M})$$

  The probability of future observations given a the HMM is in state j at time t

Backward probabilities

- To estimate the state occupation probabilities it is useful to define (recursively) another set of probabilities—the *Backward probabilities*

$$\beta_j(t) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|\, q_t = j, \mathcal{M})$$

  The probability of future observations given a the HMM is in state j at time t
- These can be recursively computed (going backwards in time)

Backward probabilities

- To estimate the state occupation probabilities it is useful to define (recursively) another set of probabilities—the *Backward probabilities*

  $$\beta_j(t) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|\, q_t = j, \mathcal{M})$$

  The probability of future observations given a the HMM is in state j at time t
- These can be recursively computed (going backwards in time)
  - Initialisation

    $$\beta_i(T) = a_{iE}$$

Backward probabilities

- To estimate the state occupation probabilities it is useful to define (recursively) another set of probabilities—the *Backward probabilities*

  $$\beta_j(t) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|\, q_t = j, \mathcal{M})$$

  The probability of future observations given a the HMM is in state j at time t
- These can be recursively computed (going backwards in time)
  - Initialisation

    $$\beta_i(T) = a_{iE}$$

  - Recursion

    $$\beta_i(t) = \sum_{j=1}^{J} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_j(t+1) \quad \text{for } t = T-1, \ldots, 1$$

Backward probabilities

- To estimate the state occupation probabilities it is useful to define (recursively) another set of probabilities—the *Backward probabilities*

$$\beta_j(t) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|\, q_t = j, \mathcal{M})$$

  The probability of future observations given a the HMM is in state j at time t
- These can be recursively computed (going backwards in time)
  - Initialisation

$$\beta_i(T) = a_{iE}$$

  - Recursion

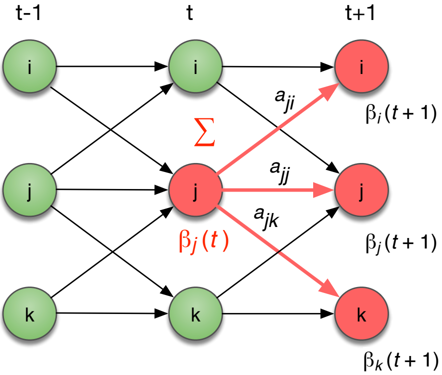
$$\beta_i(t) = \sum_{j=1}^{J} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_j(t+1) \quad \text{for } t = T-1, \ldots, 1$$

  - Termination

$$p(\mathbf{X} \,|\, \mathcal{M}) = \beta_0(0) = \sum_{j=1}^{J} a_{0j} b_j(\mathbf{x}_1) \beta_j(1) = \alpha_E$$

Backward Recursion

$$\beta_j(t) = p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \mid q_t = j, \mathcal{M}) = \sum_{j=1}^{J} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_j(t+1)$$



State Occupation Probability

- The **state occupation probability** $\gamma_j(t)$ is the probability of occupying state j at time t given the sequence of observations
- Express in terms of the forward and backward probabilities:

$$\gamma_j(t) = P(q_t = j \,|\, \mathbf{X}, \mathcal{M}) = \frac{1}{\alpha_E} \alpha_j(\,t\,) \beta_j(\,t\,)$$

recalling that $p(\mathbf{X} \,|\, \mathcal{M}) = \alpha_E$

- Since

$$\begin{aligned}
\alpha_j(t)\beta_j(t) &= p(\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = j \,|\, \mathcal{M}) \\
&\qquad p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T \,|q_t = j, \mathcal{M}) \\
&= p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_T, q_t = j \,|\, \mathcal{M}) \\
&= p(\mathbf{X}, q_t = j \,|\, \mathcal{M})
\end{aligned}$$

$$P(q_t = j \,|\, \mathbf{X}, \mathcal{M}) = \frac{p(\mathbf{X}, q_t = j \,|\, \mathcal{M})}{p(\mathbf{X} \,|\, \mathcal{M})}$$

Re-estimation of Gaussian parameters

- The sum of state occupation probabilities through time for a state, may be regarded as a "soft" count
- We can use this "soft" alignment to re-estimate the HMM parameters:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} \gamma_j(t)\boldsymbol{x}_t}{\sum_{t=1}^{T} \gamma_j(t)}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} \gamma_j(t)(\boldsymbol{x}_t - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_j)^T}{\sum_{t=1}^{T} \gamma_j(t)}$$

Re-estimation of transition probabilities

- Similarly to the state occupation probability, we can estimate $\xi_{i,j}(t)$, the probability of being in i at time t and j at $t+1$, given the observations:

$$\begin{aligned}\xi_t(i,j) &= P(q_t = i, q_{t+1} = j \mid \mathbf{X}, \mathcal{M}) \\ &= \frac{p(q_t = i, q_{t+1} = j, \mathbf{X} \mid \mathcal{M})}{p(\mathbf{X} \mid \mathcal{M})} \\ &= \frac{\alpha_i(t) a_{ij} b_j(\mathbf{x}_{t+1}) \beta_j(t+1)}{\alpha_E}\end{aligned}$$

- We can use this to re-estimate the transition probabilities

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} \xi_{i,j}(t)}{\sum_{k=1}^{J} \sum_{t=1}^{T} \xi_{i,k}(t)}$$

Pulling it all together

- Iterative estimation of HMM parameters using the EM algorithm. At each iteration
  - E step: For all time-state pairs
    1. Recursively compute the forward probabilities $\alpha_j(t)$ and backward probabilities $\beta_j(t)$
    2. Compute the state occupation probabilities $\gamma_j(t)$ and $\xi_{i,j}(t)$
  - M step: Based on the estimated state occupation probabilities re-estimate the HMM parameters: mean vectors $\boldsymbol{\mu}_j$, covariance matrices $\boldsymbol{\Sigma}_j$ and transition probabilities a_{ij}
- The application of the EM algorithm to HMM training is sometimes called the Forward-Backward algorithm or Baum-Welch algorithm

Extension to a corpus of utterances

- We usually train from a large corpus of R utterances
- If $\mathbf{x}_t^r$ is the t th frame of the r th utterance $\mathbf{X}^r$ then we can compute the probabilities $\alpha_j^r(t)$, $\beta_j^r(t)$, $\gamma_j^r(t)$ and $\xi_{i,j}^r(t)$ as before
- The re-estimates are as before, except we must sum over the R utterances, eg:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_j^r(t) \boldsymbol{x}_t^r}{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_j^r(t)}$$

Extension to a corpus of utterances

- We usually train from a large corpus of R utterances
- If $\mathbf{x}_t^r$ is the t th frame of the r th utterance $\mathbf{X}^r$ then we can compute the probabilities $\alpha_j^r(t)$, $\beta_j^r(t)$, $\gamma_j^r(t)$ and $\xi_{i,j}^r(t)$ as before
- The re-estimates are as before, except we must sum over the R utterances, eg:

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_j^r(t) \boldsymbol{x}_t^r}{\sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_j^r(t)}$$

- In addition, we usually employ "*embedded training*", in which fine tuning of phone labelling with "*forced Viterbi alignment*" or forced alignment is involved. (For details see Section 9.7 in Jurafsky and Martin's SLP)

Extension to Gaussian mixture model (GMM)

- The assumption of a Gaussian distribution at each state is very strong; in practice the acoustic feature vectors associated with a state may be strongly non-Gaussian
- In this case an M-component Gaussian mixture model is an appropriate density function:

$$b_j(\mathbf{x}) = p(\mathbf{x} \,|\, q = j) = \sum_{m=1}^{M} c_{jm} \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$$

  Given enough components, this family of functions can model any distribution.
- Train using the EM algorithm, in which the component estimation probabilities are estimated in the E-step

EM training of HMM/GMM

- Rather than estimating the state-time alignment, we estimate the component/state-time alignment, and component-state occupation probabilities $\gamma_{jm}(t)$: the probability of occupying mixture component m of state j at time t.

  ($\xi_{tm}(j)$ in Jurafsky and Martin's SLP)

- We can thus re-estimate the mean of mixture component m of state j as follows

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{t=1}^{T} \gamma_{jm}(t) \boldsymbol{x}_t}{\sum_{t=1}^{T} \gamma_{jm}(t)}$$

  And likewise for the covariance matrices (mixture models often use diagonal covariance matrices)

- The mixture coefficients are re-estimated in a similar way to transition probabilities:

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} \gamma_{jm}(t)}{\sum_{m'=1}^{M} \sum_{t=1}^{T} \gamma_{jm'}(t)}$$

Doing the computation

- The forward, backward and Viterbi recursions result in a long sequence of probabilities being multiplied
- This can cause floating point *underflow* problems
- In practice computations are performed in the log domain (in which multiplies become adds)
- Working in the log domain also avoids needing to perform the exponentiation when computing Gaussians

Summary: HMMs

- HMMs provide a generative model for statistical speech recognition
- Three key problems
  1. Computing the overall likelihood: the Forward algorithm
  2. Decoding the most likely state sequence: the Viterbi algorithm
  3. Estimating the most likely parameters: the EM (Forward-Backward) algorithm
- Solutions to these problems are tractable due to the two key HMM assumptions
  1. Conditional independence of observations given the current state
  2. Markov assumption on the states

References: HMMs

- Gales and Young (2007). “The Application of Hidden Markov Models in Speech Recognition”, *Foundations and Trends in Signal Processing*, **1** (3), 195–304: section 2.2.
- Jurafsky and Martin (2008). *Speech and Language Processing* (2nd ed.): sections 6.1–6.5; 9.2; 9.4. (Errata at `http://www.cs.colorado.edu/~martin/SLP/Errata/SLP2-PIEV-Errata.html`)
- Rabiner and Juang (1989). “An introduction to hidden Markov models”, *IEEE ASSP Magazine*, **3** (1), 4–16.
- Renals and Hain (2010). “Speech Recognition”, *Computational Linguistics and Natural Language Processing Handbook*, Clark, Fox and Lappin (eds.), Blackwells.