Speech and Language Processing
Chapter 6: Hidden Markov Models (HMM)

Ines Rehbein

NCLT, Dublin City University

Outline

1. Hidden Markov Models
   - Introduction
   - Evaluation: the Forward Algorithm
   - Decoding: the Viterbi Algorithm
   - Training: the Forward-Backward Algorithm

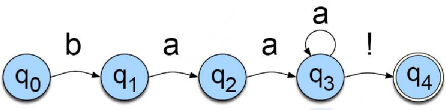
Overview

- HMMs and MEMs are probabilistic sequence classifiers:

  given a sequence of units they compute a probability distribution over possible labels and choose the best label sequence

Background: Extensions of Finite Automata

- The **HMM** is an extensions of a finite automaton
- **Finite automata**: set of states, set of transitions between states taken based on the input



- **Weighted finite-state automata**: each arc is associated with a probability
- **Markov chain**:
  - weighted automaton in which input uniquely determines the transitions
  - can only deal with unambiguous input data

Hidden Markov Model

- HMM can deal with "hidden" data (data which cannot be observed directly)
- HMM consists of:
  - $Q = q_1 \ q_2 \ldots q_N$ — a set of N **states**
  - $A = a_{11} \ a_{12} \ldots a_{n1} \ldots a_{nn}$ — a **transition probability matrix**
  - $O = o_1 \ o_2 \ldots o_T$ — a sequence of **observations**
  - $B = b_i(o_t)$ — a sequence of **observation likelihoods**
  - q_0, q_F — a **start state** and a **final state**
- fully connected (ergodic) HMM vs. left-to-right (Bakis) HMM

First Order HMM

- 2 simplifying assumptions:
  1. **Markov Assumption**: the probability of a particular state is dependent only on the previous state

$$P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}) \tag{1}$$

  2. **Output Independence Assumption**: the probability of an output observation is dependent only on the state that produced the observation

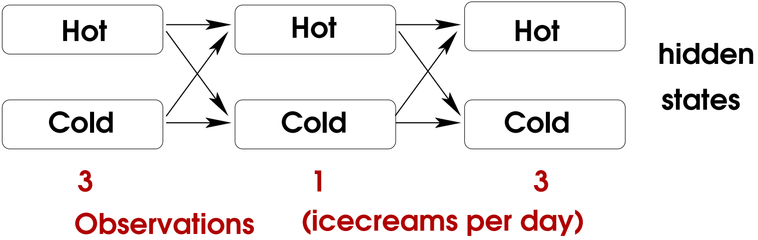
$$P(o_i|q_1...q_i, ..., q_T, o_1, ..., o_i, ..., o_T) = P(o_i|q_i) \tag{2}$$

3 Main Problems

1. **Evaluation**: Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$
   - Which HMM most probably generated a given sequence O?
   - Application: Speech recognition
2. **Decoding**: Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q
   - Application: Tagging
3. **Learning**: Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B
   - Generate an HMM from a sequence of observations
   - Application: Learn the parameters of an HMM (Training)

Evaluation: the Forward Algorithm

- **Task**: We have a sequence of observations O telling us how many icecreams Jason Eisner ate in summer 2007.
  $\Rightarrow$ Determine the probability of an icecream observation without knowing the hidden state sequence (weather conditions)



Evaluation: the Forward Algorithm

- **Simplify** task: compute the probability of an icecream observation O given a state sequence Q:

$$P(O|Q) = \Pi_{i=1}^{T} P(o_i|q_i) \tag{3}$$

- Probability for icecream observation (3, 1, 3) with state sequence (hot hot cold):

$$P(3\ 1\ 3|hot\ hot\ cold) = P(3|hot) \times P(1|hot) \times P(3|cold) \tag{4}$$

Evaluation: the Forward Algorithm

- **Simplify** task: compute the probability of an icecream observation O given a state sequence Q:

$$P(O|Q) = \Pi_{i=1}^{T} P(o_i|q_i) \tag{3}$$

- Probability for icecream observation (3, 1, 3) with state sequence (hot hot cold):

$$P(3\ 1\ 3|hot\ hot\ cold) = P(3|hot) \times P(1|hot) \times P(3|cold) \tag{4}$$

Evaluation: the Forward Algorithm (II)

- But: we don't know the hidden state sequence
  $\Rightarrow$ compute probability of observation (3, 1, 3) by summing over all possible weather sequences, weighted by their probability
- Compute the joint probability of being in a particular weather sequence Q and generating a particular sequence O of ice-cream events:

$$P(O,Q) = P(O|Q) \times P(Q) = \Pi_{i=1}^{n} P(o_i|q_i) \times \Pi_{i=1}^{n} P(q_i|q_{i-1}) \quad (5)$$

$$P(3\ 1\ 3,\ hot\ hot\ cold) = \\ P(3\ 1\ 3,\ cold\ cold\ cold) + P(3\ 1\ 3,\ cold\ cold\ hot) + \\ P(3\ 1\ 3,\ hot\ hot\ cold) \ + ...$$

Evaluation: the Forward Algorithm (II)

- But: we don't know the hidden state sequence
  $\Rightarrow$ compute probability of observation (3, 1, 3) by summing over all possible weather sequences, weighted by their probability
- Compute the joint probability of being in a particular weather sequence Q and generating a particular sequence O of ice-cream events:

$$P(O,Q) = P(O|Q) \times P(Q) = \Pi_{i=1}^{n} P(o_i|q_i) \times \Pi_{i=1}^{n} P(q_i|q_{i-1}) \quad (5)$$

$$\begin{aligned} P(3\ 1\ 3,\ hot\ hot\ cold) = \\ & P(3\ 1\ 3,\ cold\ cold\ cold) + P(3\ 1\ 3,\ cold\ cold\ hot) + \\ & P(3\ 1\ 3,\ hot\ hot\ cold) \quad + \dots \end{aligned}$$

Evaluation: the Forward Algorithm (II)

- But: we don't know the hidden state sequence
  $\Rightarrow$ compute probability of observation (3, 1, 3) by summing over all possible weather sequences, weighted by their probability
- Compute the joint probability of being in a particular weather sequence Q and generating a particular sequence O of ice-cream events:

$$P(O,Q) = P(O|Q) \times P(Q) = \Pi_{i=1}^{n} P(o_i|q_i) \times \Pi_{i=1}^{n} P(q_i|q_{i-1}) \quad (5)$$

$$\begin{aligned} P(3\ 1\ 3,\ hot\ hot\ cold) = \\ & P(3\ 1\ 3,\ cold\ cold\ cold) + P(3\ 1\ 3,\ cold\ cold\ hot) + \\ & P(3\ 1\ 3,\ hot\ hot\ cold) \quad + \ldots \end{aligned}$$

Evaluation: the Forward Algorithm (III)

- For an HMM with N hidden states and T observations $\Rightarrow N^T$ possible hidden sequences
- For a real task N^T is too large to compute the observation likelihood for each hidden state
- Instead: Use **Forward Algorithm** (dynamic programming algorithm with $O(TN^2)$)

Evaluation: the Forward Algorithm (IV)

- Forward Algorithm computes the observation probability by **summing over the probabilities of all possible hidden state paths** that could generate the observation sequence
- Forward Algorithm efficiently stores the probabilities for each path in a **forward trellis**
- Each cell of the trellis $\alpha_t(j)$ represents the probability of being in state j after seeing the first t observations, given the automaton λ
- The value of each cell is computed by summing over the probabilities of every path that could lead to this cell:

$$\alpha_t(J) = P(o_1, o_2 ... o_t, q_t = j|\lambda) \tag{6}$$

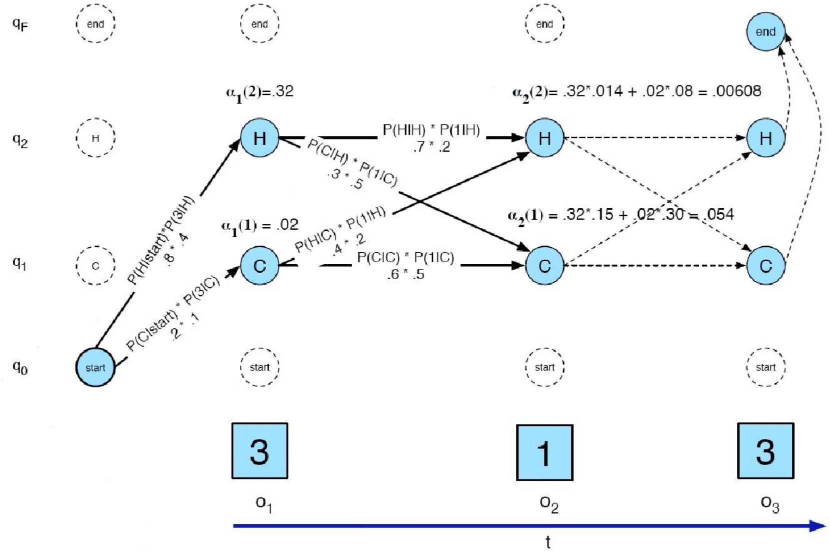
Evaluation: the Forward Algorithm (V)

- For a given state q_j at time t, the value $\alpha_t(j)$ is computed as:

$$\alpha_t(j) = \sum_{i=1}^{N} a_{t-1}(i) a_{ij} b_j(o_t) \tag{7}$$

- $\alpha_{t-1}(i)$ the **previous forward path probability** from the previous time step
- a_{ij} the **transition probability** from previous state q_i to current state q_j
- $b_j(o_t)$ the **state observation likelihood** of the observation symbol o_t given the current state j

Evaluation: the Forward Algorithm (VI)



Evaluation: the Forward Algorithm (VII)

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N+2,T]*
for each state s **from** 1 **to** N **do** ;initialization step
$forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ;recursion step
for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forward[q_F,T] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$$; termination step

return $forward[q_F, T]$

The Forward Algorithm - Summary

- **Task**: Find the probability of a sequence of observations O given an HMM
- We **reduce the complexity** of calculating the probability by creating a **trellis** and calculating **partial probabilities** for each cell in the trellis (the probability of getting to a particular state q at time t)
- The probability of the observation sequence O is computed **recursively** by calculating the partial probabilities at time $t = 1, 2, ..., T$ and adding all α's at $t = T$

Decoding: The Viterbi Algorithm

- **Task**: We have a sequence of icecream observations (3, 1, 3) and an HMM. Find the sequence of hidden states which most likely produced the sequence of observations (3, 1, 3)
- We can't use the Forward Algorithm because there is an **exponentially large** number of state sequences
- **Solution**: Use Viterbi algorithm (dynamic programming algorithm with $O(TN^2)$)

Decoding: The Viterbi Algorithm (II)

- **Idea**: process the observation sequence left-to-right, fill the trellis
- Each cell of the trellis $v_t(j)$ represents the probability that the HMM is in state j after seeing the first t observations and passing through the **most probable** state sequence $q_0, q_1, ..., q_{t-1}$, given the automaton λ
- The value of $v_t(j)$ is computed by recursively taking the most probable path that could lead to this cell:

$$v_t(j) = max_{q_0, q_1, \ldots, q_{t-1}} P(q_0, q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j | \lambda) \quad (8)$$

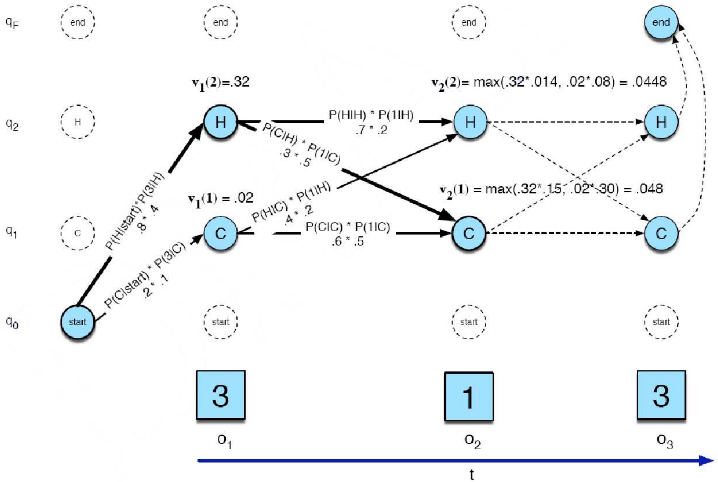
Decoding: The Viterbi Algorithm (III)

- For a given state q_j at time t, the value $v_t(j)$ is computed as:

$$v_t(j) = max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \tag{9}$$

- $v_{t-1}(i)$ the **previous Viterbi path probability** from the previous time step
- a_{ij} the **transition probability** from previous state q_i to current state q_j
- $b_j(o_t)$ the **state observation likelihood** of the observation symbol o_t given the current state j

Computing Likelihood: the Viterbi Algorithm (IV)



Computing Likelihood: the Viterbi Algorithm (VI)

- Find the **most probable** sequence of hidden states given a sequence of observed states
- Exploit **time invariance** of probabilities to reduce complexity by avoiding the necessity for examining every route through the trellis
- Keep a **backward pointer** for each state ($t > 1$), and store probability with each state (probability of having reached the state following the path indicated by the back pointers)
- When the algorithm reaches the states at time t = T, the probabilities for the final states are the probabilities of following the most probable route to that state

Computing Likelihood: the Viterbi Algorithm (V)

function VITERBI(*observations* of len T, *state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
for each state s **from** 1 **to** N **do** ;initialization step
$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ;recursion step
for each state s **from** 1 **to** N **do**
$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s', t-1] * a_{s',s} * b_s(o_t)$
$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s', t-1] * a_{s',s}$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

The Viterbi Algorithm - Summary

- **Viterbi** is identical to the forward algorithm except that it takes the **max** over the previous path probabilities where the **forward** algorithm takes the **sum**
- Forward algorithm computes observation likelihood, Viterbi computes **most probable state sequence**
- Viterbi has **backpointers** to keep track of the path of hidden states that leads to each state
  $\Rightarrow$ trace back the best path that leads to each state (Viterbi **backtrace**)

Training: The Forward-Backward Algorithm (Baum-Welch Algorithm)

- **Task**: We have a sequence of icecream observations O and a set of hidden states H, C
  $\Rightarrow$ Train the HMM and learn the parameters A (transition probabilities) and B (observation likelihood)
- Forward-Backward Algorithm: Overview
  - Make an **initial guess** at the parameters, then assess the approximation and try to reduce the error
  - Compute the **forward probablitiy** of arriving at the state given the approximation and the **backward probability** of generating the final state of the model

Training: The Forward-Backward Algorithm (Baum-Welch Algorithm)

- **Task**: We have a sequence of icecream observations O and a set of hidden states H, C
  $\Rightarrow$ Train the HMM and learn the parameters A (transition probabilities) and B (observation likelihood)
- Forward-Backward Algorithm: Overview
  - Make an **initial guess** at the parameters, then assess the approximation and try to reduce the error
  - Compute the **forward probablitiy** of arriving at the state given the approximation and the **backward probability** of generating the final state of the model

Training: The Forward-Backward Algorithm (II)

- If we would know which hidden states produced the output, we could compute the **maximum likelihood estimate** of the transition probability a_{ij}:

  count the number of times the transition was taken $C(i \rightarrow j)$, and normalise by the total count of all times we took any transition from state i:

$$a_{ij} = \frac{C(i \rightarrow j)}{\sum_{q \in Q} C(i \rightarrow q)} \tag{10}$$

- Can be directly computed for a Markov chain, but not for an HMM (we don't know which path of states was taken to get a particular input)

Training: The Forward-Backward Algorithm (II)

- If we would know which hidden states produced the output, we could compute the **maximum likelihood estimate** of the transition probability a_{ij}:

  count the number of times the transition was taken $C(i \rightarrow j)$, and normalise by the total count of all times we took any transition from state i:

$$a_{ij} = \frac{C(i \rightarrow j)}{\sum_{q \in Q} C(i \rightarrow q)} \tag{10}$$

- Can be directly computed for a Markov chain, but not for an HMM (we don't know which path of states was taken to get a particular input)

Training: The Forward-Backward Algorithm (III)

- Forward-Backward Algorithm uses two intutions to solve the problem:

  1. *iteratively* estimate the counts: start with an estimate for the transition and observation probabilities, then use these estimated probabilities to derive better and better probabilities
  2. estimate probabilities by computing the forward probability for an observation and dividing that probability mass among all the different paths that contributed to this forward probability

Training: The Forward-Backward Algorithm (V)

- Define **backward probability** β: probability of seeing the observations from time $t+1$ to the end, given that we are in state j at time t, given the automaton λ

$$\beta_t(i) = P(o_{t+1}, o_{t+2} \ldots o_T | q_t = i, \lambda) \tag{11}$$

- Sum over all successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and observation probabilities $b_j(o_{t+1})$
  - **Initialisation**: $\beta_T(i) = a_{i,F}, 1 \le i \le N$ (12)
  - **Recursion**:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), 1 \le i \le N, 1 \le t < T \tag{13}$$

  - **Termination**:

$$P(O|\lambda) = \alpha_T(q_F) = \beta_1(0) = \sum_{j=1}^{N} a_{0j} b_j(o_1) \beta_1(j) \tag{14}$$

Training: The Forward-Backward Algorithm (V)

- Define **backward probability** β: probability of seeing the observations from time $t+1$ to the end, given that we are in state j at time t, given the automaton λ

$$\beta_t(i) = P(o_{t+1}, o_{t+2} ... o_T | q_t = i, \lambda) \tag{11}$$

- Sum over all successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and observation probabilities $b_j(o_{t+1})$
  - **Initialisation**: $$\beta_T(i) = a_{i,F}, 1 \leq i \leq N \tag{12}$$
  - **Recursion**: $$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), 1 \leq i \leq N, 1 \leq t < T \tag{13}$$
  - **Termination**: $$P(O|\lambda) = \alpha_T(q_F) = \beta_1(0) = \sum_{j=1}^{N} a_{0j} b_j(o_1) \beta_1(j) \tag{14}$$

Training: The Forward-Backward Algorithm (V)

- **Forward** and **backward probability** help to compute the transition probability a_{ij} and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path is hidden
- Estimate the probability a_{ij} of a particular transition between states i and j:

$$\hat{a}_{ij} = \frac{expected\ \ number\ \ of\ \ transitions\ \ from\ \ state\ \ i\ \ to\ \ state\ \ j}{expected\ \ number\ \ of\ \ transitions\ \ from\ \ state\ \ i}$$

- How do we get the numerator?

If we had an estimate of the probability that a transition $i \rightarrow j$ was taken at time t for each $t \in T$, we could sum over all times t to estimate the total count:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \tag{15}$$

Training: The Forward-Backward Algorithm (V)

- **Forward** and **backward probability** help to compute the transition probability a_{ij} and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path is hidden
- Estimate the probability a_{ij} of a particular transition between states i and j:

$$\hat{a}_{ij} = \frac{expected\ \ number\ \ of\ \ transitions\ \ from\ \ state\ \ i\ \ to\ \ state\ \ j}{expected\ \ number\ \ of\ \ transitions\ \ from\ \ state\ \ i}$$

- How do we get the numerator?

If we had an estimate of the probability that a transition $i \to j$ was taken at time t for each $t \in T$, we could sum over all times t to estimate the total count:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \qquad (15)$$

Training: The Forward-Backward Algorithm (V)

- **Forward** and **backward probability** help to compute the transition probability a_{ij} and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path is hidden
- Estimate the probability a_{ij} of a particular transition between states i and j:

$$\hat{a}_{ij} = \frac{expected\ \ number\ \ of\ \ transitions\ \ from\ \ state\ \ i\ \ to\ \ state\ \ j}{expected\ \ number\ \ of\ \ transitions\ \ from\ state\ \ i}$$

- How do we get the numerator?

If we had an estimate of the probability that a transition $i \to j$ was taken at time t for each $t \in T$, we could sum over all times t to estimate the total count:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \tag{15}$$

Training: The Forward-Backward Algorithm (V)

- **Forward** and **backward probability** help to compute the transition probability a_{ij} and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path is hidden
- Estimate the probability a_{ij} of a particular transition between states i and j:

$$\hat{a}_{ij} = \frac{expected\ \ number\ \ of\ \ transitions\ \ from\ \ state\ \ i\ \ to\ \ state\ \ j}{expected\ \ number\ \ of\ \ transitions\ \ from\ state\ \ i}$$

- How do we get the numerator?

If we had an estimate of the probability that a transition $i \rightarrow j$ was taken at time t for each $t \in T$, we could sum over all times t to estimate the total count:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O, \lambda) \tag{15}$$

Training: The Forward-Backward Algorithm (VI)

- Define ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence O and the HMM λ
- First compute something close to ξ, but including the probability of the observation:

$$not-quite-\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda) \tag{16}$$

- Use forward probability, transition probability, observation likelihood and backward probability:

$$not-quite-\xi_t(i,j) = \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j) \tag{17}$$

- To get ξ from not-quite-ξ, we have to divide by $P(O|\lambda)$, following the laws of probability:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{18}$$

Training: The Forward-Backward Algorithm (VI)

- Define ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence O and the HMM λ
- First compute something close to ξ, but including the probability of the observation:

$$not - quite - \xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda) \tag{16}$$

- Use forward probability, transition probability, observation likelihood and backward probability:

$$not - quite - \xi_t(i,j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \tag{17}$$

- To get ξ from not-quite-ξ, we have to divide by $P(O|\lambda)$, following the laws of probability:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{18}$$

Training: The Forward-Backward Algorithm (VI)

- Define ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence O and the HMM λ
- First compute something close to ξ, but including the probability of the observation:

$$not - quite - \xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda) \qquad (16)$$

- Use forward probability, transition probability, observation likelihood and backward probability:

$$not - quite - \xi_t(i,j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \qquad (17)$$

- To get ξ from not-quite-ξ, we have to divide by $P(O|\lambda)$, following the laws of probability:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \qquad (18)$$

Training: The Forward-Backward Algorithm (VI)

- Define ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence O and the HMM λ
- First compute something close to ξ, but including the probability of the observation:

$$not - quite - \xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda) \tag{16}$$

- Use forward probability, transition probability, observation likelihood and backward probability:

$$not - quite - \xi_t(i,j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \tag{17}$$

- To get ξ from not-quite-ξ, we have to divide by $P(O|\lambda)$, following the laws of probability:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{18}$$

Training: The Forward-Backward Algorithm (VII)

- $P(O|\lambda)$ is simply the forward (or backward) probability of the whole utterance:

$$P(O|\lambda) = a_T(N) = \beta_T(1) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j) \tag{19}$$

- Therefore the final equation for ξ is:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \tag{20}$$

- and the **expected number of transitions** from state i to j is the sum over all t of ξ

Training: The Forward-Backward Algorithm (VII)

- $P(O|\lambda)$ is simply the forward (or backward) probability of the whole utterance:

$$P(O|\lambda) = a_T(N) = \beta_T(1) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j) \tag{19}$$

- Therefore the final equation for ξ is:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \tag{20}$$

- and the **expected number of transitions** from state i to j is the sum over all t of ξ

Training: The Forward-Backward Algorithm (VII)

- To get the **total expected number of transitions** from state i we sum over all transitions out of state i:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)} \tag{21}$$

- We also need to recompute the **observation probability** $\hat{b}_j(v_k)$ (probability of a given symbol v_k from observation vocabulary V, given state j):

$$\hat{b}_j(v_k) = \frac{expected\ number\ of\ times\ in\ state\ j\ and\ observing\ symbol\ v_k}{expected\ number\ of\ times\ in\ state\ j} \tag{22}$$

Training: The Forward-Backward Algorithm (VII)

- To get the **total expected number of transitions** from state i we sum over all transitions out of state i:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)} \tag{21}$$

- We also need to recompute the **observation probability** $\hat{b}_j(v_k)$ (probability of a given symbol v_k from observation vocabulary V, given state j):

$$\hat{b}_j(v_k) = \frac{expected\ number\ of\ times\ in\ state\ j\ and\ observing\ symbol\ v_k}{expected\ number\ of\ times\ in\ state\ j} \tag{22}$$

Training: The Forward-Backward Algorithm (VIII)

- We need to know the probability of being in state j at time t:

$$\gamma_t(j) = P(q_t = j|O, \lambda) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \tag{23}$$

- Now we can compute b:
  - Numerator: sum $\gamma_t(j)$ for all time steps t in which the observation o_t is the symbol v_k
  - Denominator: sum $\gamma_t(j)$ over all time steps t

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{24}$$

Training: The Forward-Backward Algorithm (VIII)

- We need to know the probability of being in state j at time t:

$$\gamma_t(j) = P(q_t = j|O,\lambda) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \tag{23}$$

- Now we can compute b:
  - Numerator: sum $\gamma_t(j)$ for all time steps t in which the observation o_t is the symbol v_k
  - Denominator: sum $\gamma_t(j)$ over all time steps t

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{24}$$

Training: The Forward-Backward Algorithm (IX)

- Now we can **re-estimate** A **and** B from O assuming that we have a previous estimate of A and B
- Forward-Backward Algorithm as a special case of EM algorithm:
  1. E-step (expectation step):
     Compute the expected state occupancy count γ and the expected state transition count ξ, from earlier A and B probabilities
  2. M-step (maximisation step):
     Use γ and ξ to recompute new A and B probabilities

Training: The Forward-Backward Algorithm (X)

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** $HMM=(A,B)$

initialize A and B
iterate until convergence
E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \ \forall\ t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \ \forall\ t,\ i,\text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

Summary

Problem	**Algorithm**	**Complexity**
Evaluation: Calculating $P(q_t = q_i \mid O_1, O_2 \ldots O_t)$	Forward	$O(TN^2)$
Decoding: Computing $Q^* = argmax_Q P(Q \mid O)$	Viterbi	$O(TN^2)$
Learning: Computing $\lambda^* = argmax_\lambda P(O \mid \lambda)$	Forward-Backward (EM)	$O(TN^2)$

- HMMs have proved to be of great value in analysing real systems; their usual drawback is the over-simplification associated with the Markov assumption - that a state is dependent only on predecessors, and that this dependence is time independent.

References

- Jurafsky, D. and J. H. Martin. Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2007.
- Online Tutorial: Hidden Markov Models. (http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html)