The three basic HMM problems

(These slides follow the classic formulation by Ferguson, as published by Rabiner and Juang, as adapted by Manning and Schutze. Note the change in notation from the last lecture!!)
Parameters of an HMM

- **States**: A set of states $S = s_1, \ldots, s_n$

- **Transition probabilities**: $A = a_{1,1}, a_{1,2}, \ldots, a_{n,n}$
  Each $a_{i,j}$ represents the probability of transitioning from state $s_i$ to $s_j$.

- **Emission probabilities**: a set $B$ of functions of the form $b_i(o_t)$ which is the probability of observation $o_t$ being emitted by $s_i$

- **Initial state distribution**: $\pi_i$ is the probability that $s_i$ is a start state
The Three Basic HMM Problems

- **Problem 1 (Evaluation):** Given the observation sequence $O = o_1, \ldots, o_T$ and an HMM model $\lambda = (A, B, \pi)$, how do we compute the probability of $O$ given the model?

- **Problem 2 (Decoding):** Given the observation sequence $O$ and an HMM model $\lambda$, how do we find the state sequence that best explains the observations?

- **Problem 3 (Learning):** How do we adjust the model parameters $\lambda = (A, B, \pi)$, to maximize $P(O \mid \lambda)$?
Problem 1: Probability of an Observation Sequence

• Q: What is $P(O | \lambda)$?

• A: the sum of the probabilities of all possible state sequences in the HMM.

• Naïve computation is very expensive. Given $T$ observations and $N$ states, there are $N^T$ possible state sequences.
  • (for $T=10$ and $N=10$, 10 billion different paths!!)

• Solution: linear time dynamic programming!
The Crucial Data Structure: **The Trellis**
Forward Probabilities: $\alpha$

- For a given HMM $\lambda$, for some time $t$, what is the probability that the partial observation $o_1 \ldots o_t$ has been generated and that the state at time $t$ is $i$?

$$\alpha_t(i) = P(o_1 \ldots o_t, q_t = s_i \mid \lambda)$$

- Forward algorithm computes $\alpha_t(i)$, $0 < i < N$, $0 < t < T$ in time $O(N^2 T)$ using the trellis.
Forward Algorithm: Induction step

\[ \alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_{j}(o_t) \]
Forward Algorithm

- **Initialization** (probability that $o_1$ has been generated and that the state is $i$ at time $t=1$):

  $$\alpha_1(i) = \pi_i b_i(o_1) \quad 1 \leq i \leq N$$

- **Induction**:

  $$\alpha_t(j) = \left[ \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \right] b_j(o_t) \quad 2 \leq t \leq T, \quad 1 \leq j \leq N$$

- **Termination**:

  $$P(O | \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$
Forward Algorithm Complexity

- Naïve approach requires exponential time to evaluate all $N^T$ state sequences
- Forward algorithm using dynamic programming takes $O(N^2T)$ computations
Backward Probabilities: $\beta$

- For a given HMM $\lambda$, given that the state is $i$ at time $t$, what is the probability that the partial observation $o_{t+1} \ldots o_T$ will be generated?

$$\beta_t(i) = P(o_{t+1} \ldots o_T \mid q_t = s_i, \lambda)$$

- Analogous to forward probability, just in the other direction: 
  
  **Backward algorithm computes** $\beta_t(i)$, $0<i<N$, $0<t<T$ 
  
  **in time** $0(N^2T)$ **using the trellis**
Backward Probabilities

\[
\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)
\]
Backward Algorithm

- Initialization:

\[ \beta_T(i) = 1, \quad 1 \leq i \leq N \]

- Induction (ワークイングバックワード) :

\[ \beta_t(i) = \left[ \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \right] T - 1 \geq t \geq 1, 1 \leq i \leq N \]

- Termination:

\[ P(O \mid \lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i) \]
Problem 2: Decoding

- The Forward algorithm gives the *sum of all paths* through an HMM efficiently.
- Here, we want to find the *highest probability path*.
- We want to find the state sequence \( Q = q_1 \ldots q_T \), such that

\[
Q = \arg\max_{Q'} P(Q' | O, \lambda)
\]
Viterbi Algorithm

- Just like the forward algorithm, but instead of *summing* over transitions from incoming states, compute the *maximum*

- Forward:

  \[
  \alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)
  \]

- Viterbi Recursion:

  \[
  \delta_t(j) = \max_{1 \leq i \leq N} \delta_{t-1}(i) a_{ij} b_j(o_t)
  \]
Core Idea of Viterbi Algorithm

\[ a_{Nk} \]

\[ a_{1k} \]

22/02/2002

\( O_1 \), \( O_2 \), \( O_{t-1} \), \( O_t \), \( O_{t+1} \), \( O_{T+1} \), \( O_T \)
Not quite what we want…. 

- Viterbi recursion computes the maximum probability path to state \( j \) at time \( t \) given that the partial observation \( o_1 \ldots o_t \) has been generated

\[
\delta_t(j) = \max_{1 \leq i \leq N} \delta_{t-1}(i) a_{ij} b_j(o_t)
\]

- But we want the path itself that gives the maximum probability

- Solution:
  1. Keep backpointers
  2. Find \( \arg \max_j \delta_T(j) \)
  3. Chase backpointers from state \( j \) at time \( T \) to find state sequence (backwards)
Viterbi Algorithm

- **Initialization:** \( \delta_1(i) = \pi_i b_j(o_1) \quad 1 \leq i \leq N \)

- **Induction:**
  \[
  \delta_t(j) = \left[ \max_{1 \leq i \leq N} \delta_{t-1}(i) a_{ij} \right] b_j(o_t) \quad 2 \leq t \leq T, 1 \leq j \leq N
  \]
  \[
  \psi_t(j) = \left[ \arg \max_{1 \leq i \leq N} \delta_{t-1}(i) a_{ij} \right] \quad (\text{Backpointers})
  \]

- **Termination:** \( q_T^* = \arg \max_{1 \leq i \leq N} \delta_T(i) \quad (\text{Final state!}) \)

- **Backpointer path:** \( q_t^* = \psi_{t+1}(q_{t+1}) \quad t = T-1, \ldots, 1 \)
Problem 3: Learning

- Up to now we’ve assumed that we know the underlying model \( \lambda = (A, B, \pi) \)

- Often these parameters are estimated on annotated training data, but:
  - Annotation is often difficult and/or expensive
  - Training data is different from the current data

- We want to maximize the parameters with respect to the current data, i.e., we’re looking for a model \( \lambda' \), such that \( \lambda' = \arg \max_\lambda P(O | \lambda) \)
Problem 3: Learning

• Unfortunately, there is no known way to analytically find a global maximum, i.e., a model $\lambda'$, such that $\lambda' = \arg \max_\lambda P(O | \lambda)$

• But it is possible to find a local maximum

• Given an initial model $\lambda$, we can always find a model $\lambda'$, such that $P(O | \lambda') \geq P(O | \lambda)$
Forward-Backward (Baum-Welch) algorithm

Key Idea: *parameter re-estimation* by hill-climbing

FB algorithm *iteratively re-estimates the parameters yielding a new $\lambda'$ at each iteration*

1. *Initialize $\lambda$ to a random set of values*
2. *Estimate $P(O \mid \lambda)$, filling out the trellis for both the Forward and the Backward algorithms*
3. *Reestimate $\lambda$ using both trellises, yielding a new estimate $\lambda'$*

**Theorem:** $P(O \mid \lambda') \geq P(O \mid \lambda)$
Parameter Re-estimation

- Three parameters need to be re-estimated:
  - Initial state distribution: $\pi_i$
  - Transition probabilities: $a_{i,j}$
  - Emission probabilities: $b_i(o_t)$
Re-estimating *Transition* Probabilities: Step 1

- What’s the probability of being in state $s_i$ *at time t* and going to state $s_j$, given the current model and parameters?

$$\xi_t(i, j) = P(q_t = s_i, q_{t+1} = s_j | O, \lambda)$$
Re-estimating Transition Probabilities: Step 1

\[ \xi_t(i,j) = \frac{\alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)} \]
Re-estimating *Transition* Probabilities: Step 2

- The intuition behind the re-estimation equation for transition probabilities is

\[ \hat{a}_{i,j} = \frac{\text{expected number of transitions from state } s_i \text{ to state } s_j}{\text{expected number of transitions from state } s_i} \]

- Formally:

\[
\hat{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j'=1}^{N} \xi_t(i,j')} \]

\[
\hat{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j'=1}^{N} \xi_t(i,j')} \]

\[ \sum_{t=1}^{T-1} \xi_t(i,j) \]

\[ \sum_{t=1}^{T-1} \sum_{j'=1}^{N} \xi_t(i,j') \]
Re-estimating *Transition* Probabilities

- **Defining**

\[ \gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j) \]

as the probability of being in state \( s_i \) at time \( t \),
given the complete observation \( O \)

- **We can say:**

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

\[ \sum_{t=1}^{T-1} \gamma_t(i) \]
Re-estimating *Initial State* Probabilities

- **Initial state distribution:** $\pi_i$ is the probability that $s_i$ is a start state

- **Re-estimation is easy:**
  
  $\hat{\pi}_i = \text{expected number of times in state } s_i \text{ at time } t$

- **Formally:**
  
  $\hat{\pi}_i = \gamma_1(i)$
Re-estimation of *Emission* Probabilities

- Emission probabilities are re-estimated as
  \[
  \hat{b}_i(k) = \frac{\text{expected number of times in state } s_i \text{ and observe symbol } v_k}{\text{expected number of times in state } s_i}
  \]

- Formally:
  \[
  \hat{b}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t, v_k) \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}
  \]
  where \( \delta(o_t, v_k) = 1, \text{ if } o_t = v_k, \text{ and } 0 \text{ otherwise} \)

- Note that \( \delta \) here is the Kronecker delta function and is not related to the \( \delta \) in the discussion of the Viterbi algorithm!!
The Updated Model

- Coming from \( \lambda = (A, B, \pi) \) we get to \( \lambda' = (\hat{A}, \hat{B}, \hat{\pi}) \) by the following update rules:

\[
\hat{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}
\]

\[
\hat{b}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t, v_k) \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}
\]

\[
\hat{\pi}_i = \gamma_1(i)
\]
Expectation Maximization

- The forward-backward algorithm is an instance of the more general EM algorithm
  - The E Step: Compute the forward and backward probabilities for a given model
  - The M Step: Re-estimate the model parameters