Probabilistic Model for Time-series Data: Hidden Markov Model

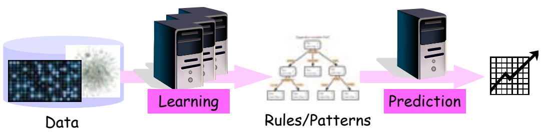
Hiroshi Mamitsuka
Bioinformatics Center
Kyoto University

Outline

- Three Problems for probabilistic models in machine learning
  1. Computing likelihood
  2. Learning
  3. Parsing (prediction)
- Define hidden Markov model (HMM)
- Three problems of HMM
  - Computing likelihood by forward probabilities
  - Learning by Baum-Welch
  - Parsing by Viterbi
- Summary

Probabilistic Model Learning

- An approach of "Machine learning": finding probabilistic patterns/rules from given data



Probabilistic Model Learning

- Probabilistic model: has probabilistic (or probability) parameters estimated from given data
- Unsupervised learning
  - One-class data: No labels attached to given examples
  - Model M gives a score (a likelihood) for a training example X: P(X|M), which should be higher by learning
  - After learning, model M should give a score for an arbitrary example X: P(X|M), which is exactly prediction

Probabilistic Model
Ex: Finite Mixture Model

- Clustering: Grouping examples and assigning a given example to a cluster
- Two variables
  - X: observable variable, corresponding to example
  - Z: latent variable, corresponding to cluster (#clusters given)
- Two probabilistic parameters
  - P(Z): Probability of a cluster
  - P(X|Z): Probability of an example given a cluster
- Likelihood of a given example, i.e. P(X|M):
  - $P(X) = \sum P(X \mid Z) P(Z)$

Probabilistic Model
Ex: Finite Mixture Model

- Learning: Estimating P(X|Z) and p(Z)
- Once learning is done, the objective of FMM is to compute P(Z|X), i.e. probability of the cluster assignment given an example
- Question: How can we compute P(Z|X) from P(X|Z) and P(Z)?
- Answer: Follow the Bayes theorem:

$$P(Z \mid X) = \frac{P(X \mid Z) P(Z)}{\sum_Z P(X \mid Z) P(Z)} \approx P(X \mid Z) P(Z)$$

Three Problems

- Must be solved by a probabilistic model to be used in real-world machine learning applications
  1. Computing likelihood: computing how likely a given example can be generated from a model
  2. Learning: estimating probability parameters of a model from given data
  3. Parsing: finding the most likely set of parameters on an example given a model

Three Problems

1. Computing likelihood
   - Likelihood: P(X|M), score given for an example by the model
   - Computing likelihood can be part of parameter estimation (learning), for example as maximum likelihood is used for learning
2. Learning
   - Parameter estimation, the most significant part
   - Typical example: Maximum likelihood
3. Parsing
   - Prediction and showing the reason of prediction
   - Can be modified from likelihood computation

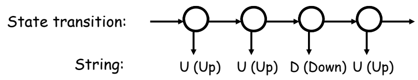
Three Problems: Finite Mixture Model

1. Computing likelihood
   - Computing P(X) due to the probabilistic structure: $L(X) = P(X) = \sum_{Z} P(X \mid Z)P(Z)$
2. Learning
   - Estimate probabilistic parameters: $P(X \mid Z), P(Z)$
3. Parsing
   - Show the cluster which maximizes the likelihood: $\hat{z} = \arg\max_z P(X \mid Z)P(Z)$

Markov Model

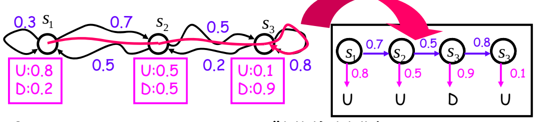
- Markov property
  - Current state depends only on <u>a finite number</u> of past states
  - 1st order Markov property
    - Current state depends on the previous state only

- Markov model (Markov chain): generates a string with Markov property



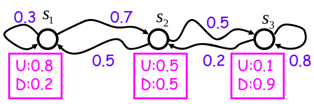
Hidden Markov Model (HMM)

- Defined by a state transition diagram, showing possible state transitions, with
  - State transition probability at an edge
  - Letter generation probability at a node



- Generates a string, say "UUDU," by a state transition path, say $s_1 s_2 s_3 s_3$,
  - with the likelihood of 0.8x0.7x0.5x0.5x0.9x0.8x0.1

1-to-many Correspondence between String and State Transition Paths



UUDU

$S_1S_2S_3S_3$: 0.8x0.7x0.5x0.5x0.9x0.8x0.1

$S_1S_2S_3S_2$: 0.8x0.7x0.5x0.5x0.9x0.2x0.5

$S_1S_1S_2S_3$: 0.8x0.3x0.8x0.7x0.5x0.5x0.1

⋮

Sum = likelihood by the model

Most probable state transition path is "hidden"!

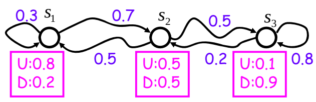
Define HMM More Formally

- Input
  - State transition diagram
    - State in given state set: $S \in S$
    - The size of states: M
  - Data: Strings = **time-series examples**
    - String in given string set: $\sigma \in \Lambda$
    - Maximum length of a string: T
- Two types of probability parameters
  - State transition probability at an edge for states i to j: a_{ij}
  - Letter generation probability at node j (of the $t+1$-th letter): $b_j(\sigma_{t+1})$
- Likelihood of state transition $\pi \in \Xi$ for given string σ : $L(\sigma, \pi)$

Three Problems for HMMs

1. Computing likelihood
   - which is the likelihood given to a string by the model, being equal to the sum of all likelihoods by all state transition paths
2. Learning
   - is to estimate two types of probability parameters, given strings
3. Parsing
   - is to find the state transition path, which gives the maximum likelihood

Computing Likelihood



UUDU

$s_1 s_2 s_3 s_3$: 0.8x0.7x0.5x0.5x0.9x0.8x0.1 $L(\sigma, \pi_1)$

$s_1 s_2 s_3 s_2$: 0.8x0.7x0.5x0.5x0.9x0.2x0.5 $L(\sigma, \pi_2)$

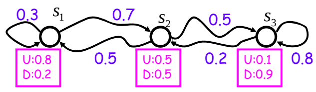
$s_1 s_1 s_2 s_3$: 0.8x0.3x0.8x0.7x0.5x0.5x0.1 $L(\sigma, \pi_3)$

⋮

Sum of the likelihoods of all possible state transition paths = the likelihood given to the string UUDU by the model: $\sum_{\pi \in \Xi} L(\sigma, \pi)$

Computing Likelihood

- Need enumerating all state transition paths, given a string and probability parameters
  - Sum of the likelihoods, each being that for a path
- => combinatorial hardness: $O(M^T)$



UUDU

$s_1 s_2 s_3 s_3$: 0.8x0.7x0.5x0.5x0.9x0.8x0.1

$s_1 s_2 s_3 s_2$: 0.8x0.7x0.5x0.5x0.9x0.2x0.5

$s_1 s_1 s_2 s_3$: 0.8x0.3x0.8x0.7x0.5x0.5x0.1

⋮

- Efficient computation manner needed: Dynamic Programming!

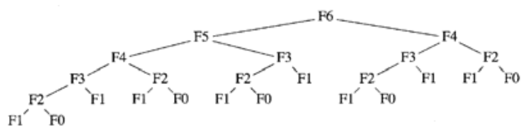
Review: Dynamic Programming

- In the case where subproblems can be solved repeatedly, solve simpler problems first and save the result
- Ex: Fibonacci number: 1, 1, 2, 3, 5, 8, 13, 21, …
  - Recursive algorithm for computing Fibonacci number which looks brief and very nice…

```
Algorithm: fib(n)
{
        if( n <= 1 )
                return 1;
        else
                return fib(n - 1) + fib(n - 2) ;
}
```

Review: Dynamic Programming Example: Fibonacci number

- But this algorithm needs computing all past numbers for each number
- Trace of the recursive calculation of Fibonacci number:



- Makes complexity of *fib*(n) an exponential order!

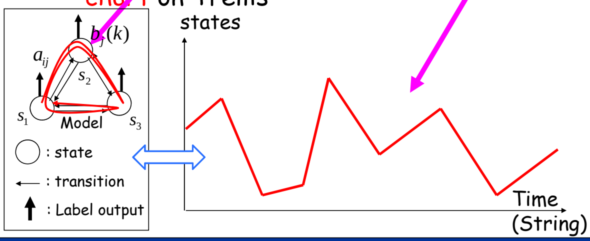
Review: Dynamic Programming Example: Fibonacci number

- Solution for this problem: use a table to save, instead of recursive computation!
- Complexity of new_fib(n): *O(n)*

```
Algorithm: new_fib(n) {
        if( n <= 1 )
                return 1;
        last = 1; nextTolast = 1 ; answer = 1;
        for( i = 2 ; i <= n ; i++ )     {
                answer = last + nextTolast ;
                nextTolast = last ;
                last = answer ;   }
        return answer ; }
```

Trellis

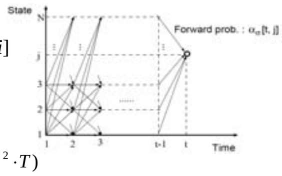
- Two-dimension of Time x States
- Makes easy to understand the dynamic programming process of HMM learning
- A state transition on HMM is a line chart on Trellis



Forward Probability: $\alpha_\sigma[t, j]$

- Given a string, the probability that the current state is *j* and substring [*1..t*] is generated, i.e. the probability covering the first part of the string
- Can be computed by dynamic programming over *t* , due to Markov property
- Updating formula:

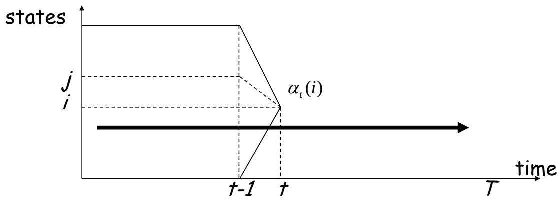
$$\alpha_\sigma[t, j] = \sum_i a_{ij} b_j(\sigma_t) \alpha_\sigma[t-1, i]$$



- Can be computed in $O(M^2 \cdot T)$
  - where M is the size of states and T is the string length

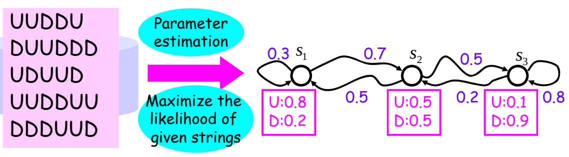
Computing Likelihood with Forward Probabilities

- Compute forward probabilities, incrementing *t*, finally having the likelihood given a string and a model: $\sum_{\pi \in \Xi|} L(\sigma, \pi) = \sum_i \alpha_T(i)$
- Complexity: $O(M^2 \cdot T) \approx O(M^3)$



Training HMM (Learning Parameters of HMM)

- Probability parameters **trained (estimated) from strings (time-series examples)**
- A standard manner is maximum likelihood for given strings, based on EM (Expectation-Maximization) algorithm



EM Algorithm in General

- Notation
  - Observable variable: X
  - Latent variable: Z
  - Parameter set: ϕ
  - Distribution: P
- Purpose
  - Maximize the likelihood of observable variables
  - i.e. obtain parameters which maximize the likelihood: $\hat{\phi} = \arg\max_\phi P_\phi(X)$

EM Algorithm in General

- Notation
  - Observable variable: X
  - Latent variable: Z
  - Parameter set: ϕ
  - Distribution: P
- Q function: $Q(\phi;\phi') = \sum_Z P_\phi(X,Z)\log P_{\phi'}(X,Z)$
- Nice property of Q function:

$$Q(\phi;\phi') > Q(\phi;\phi) \rightarrow P_{\phi'}(X) > P_\phi(X)$$

  - This means if we find ϕ' satisfying $Q(\phi;\phi') > Q(\phi;\phi)$, we can make $P_{\phi'}(X) > P_\phi(X)$

$Q(\phi;\phi') > Q(\phi;\phi) \rightarrow P_{\phi'}(X) > P_\phi(X)$

- Proof: $Q(\phi;\phi') - Q(\phi;\phi)$

$$= \sum_Z P_\phi(X,Z)\log P_{\phi'}(X,Z) - \sum_z P_\phi(X,Z)\log P_\phi(X,Z)$$

$$= \sum_z P_\phi(X,Z)\log\frac{P_{\phi'}(X,Z)}{P_\phi(X,Z)}$$

$$\le \sum_z P_\phi(X,Z)\left(\frac{P_{\phi'}(X,Z)}{P_\phi(X,Z)} - 1\right) \quad (\log x \le x - 1)$$

$$= \sum_z P_{\phi'}(X,Z) - P_\phi(X,Z)$$

$$= P_{\phi'}(X) - P_\phi(X)$$

If $Q(\phi;\phi') - Q(\phi;\phi)$ is positive, $P_{\phi'}(X) - P_\phi(X)$ must be positive. □

EM Algorithm in General

1. Choose initial parameter values
2. Repeat following two steps alternately until convergence
   - E-step: Compute Q function: $Q(\phi;\phi')$
   - M-step: Choose $\phi^{new} = \arg\max_{\phi'} Q(\phi;\phi')$

EM Algorithm for HMM

- Baum-Welch algorithm
- Correspondence
  - Observable variable = string: σ
  - Latent variable = state transition path: $\pi \in \Xi$
  - Distribution = likelihood: L
- Q function: $Q(\phi;\phi')$

$$= \sum_Z P_\phi(X,Z)\log P_{\phi'}(X,Z)$$

$$= \sum_{\pi\in\Xi} L_\phi(\sigma,\pi)\log L_{\phi'}(\sigma,\pi)$$

- Problem: Find $\phi^{new} = \arg\max_{\phi'} Q(\phi;\phi')$

Derivation of Baum-Welch (E-step)

- Assume $\phi = \{a_{ij}\}$
  - meaning that we here focus on state transition probabilities only
- Q function can be derived:

$$Q(\phi;\phi')$$

$$= \sum_{\pi\in\Xi} L_\phi(\sigma,\pi)\log L_{\phi'}(\sigma,\pi)$$

$$= \sum_{\pi\in\Xi} L_\phi(\sigma,\pi)\sum_1^{|\pi|}\log(a'_{\pi_i\pi_{i+1}}) \quad (\pi = \pi_1,\ldots,\pi_{|\pi|})$$

$$= \sum_{i,j}\log(a'_{ij}) \sum_{\pi\in\Xi|i\rightarrow j\in\pi} L_\phi(\sigma,\pi)$$

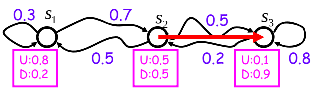
- $\sum_{\pi\in\Xi|i\rightarrow j\in\pi} L_\phi(\sigma,\pi)$ means <u>the expectation value of state transition with states from *i* to *j*</u>

E-step of Baum-Welch

- Expectation value computation needed
  - Count the number of transition paths from state *i* to state *j*
- $E_{P_\sigma}[\#((i,j),\sigma)] = \sum_{\pi\in\Xi|i\rightarrow j\in\pi} L(\sigma,\pi)$
  - Enumerate all state transition paths, having the transition from state *i* to state *j*
- Is enumerating all these state transition paths possible???

Expectation Value Computation

- Enumerating all possible paths having certain state transition
- => combinatorial hardness! : $O(M^T)$



UUDU

$s_1 s_2 s_3 s_3$: 0.8x0.7x0.5x0.5x0.9x0.8x0.1

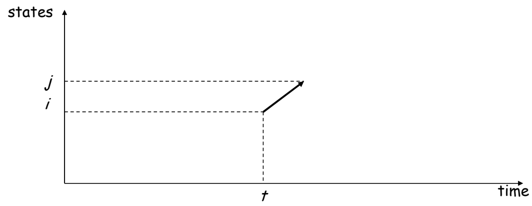
$s_1 s_2 s_3 s_2$: 0.8x0.7x0.5x0.5x0.9x0.2x0.5

$s_1 s_1 s_2 s_3$: 0.8x0.3x0.8x0.7x0.5x0.5x0.1

⋮

Computing Expectation Value for States *i* to *j*

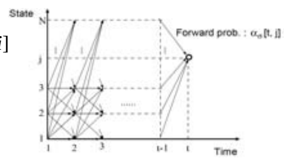
- We want to know #paths having states *i* to *j*
- First, we fix *t*, ...



Forward Probability Again

- $\alpha_\sigma[t, j]$:Given a string, the probability that the current state is *j* and substring [*1..t*] is generated, i.e. the probability covering the first part of the string
- Can be computed by dynamic programming over *t*
- Updating formula:

$$\alpha_\sigma[t, j] = \sum_i a_{ij} b_j(\sigma_i) \alpha_\sigma[t-1, i]$$

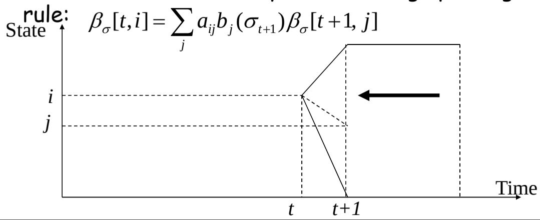


- Can be computed in $O(M^2 \cdot T)$

Backward Probability: $\beta_\sigma[t, i]$

- Given a string, the probability that the current state is *i* and substring [*t...n*] is generated, i.e. the probability covering the last part of the string
- Can be computed by dynamic programming over *t* in the reverse direction, by the following updating rule:

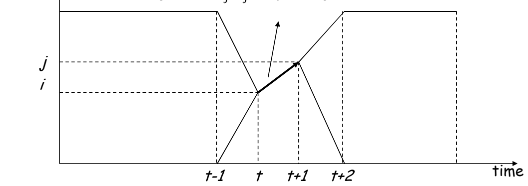
$$\beta_\sigma[t, i] = \sum_j a_{ij} b_j(\sigma_{t+1}) \beta_\sigma[t+1, j]$$



Computing Expectation Value for Transition of States *i* to *j* at *t*

- Forward probabilities cover all possible state transition paths at state *i* and time *t* for the first part of given string
- Backward probabilities cover all possible state transition paths at state *j* and time *t+1* for the last part of given string
- By using these two, we can have the expectation value of the state transition paths with state *i* to *j*

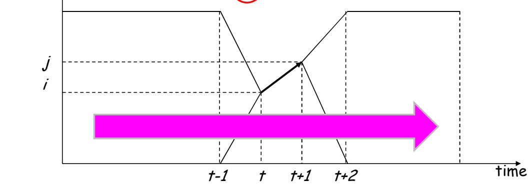
$$\alpha_\sigma[t, i] a_{ij} b_j(\sigma_{t+1}) \beta_\sigma[t+1, j]$$



Computing Expectation Value for Transition of States i to j

- We can further sum the following over all possible *t*: $\alpha_\sigma[t, i] a_{ij} b_j(\sigma_{t+1}) \beta_\sigma[t+1, j]$

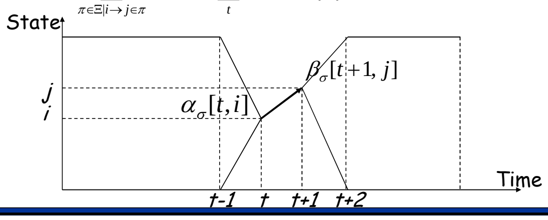
$$\sum_{\pi \in \Xi | i \to j \in \pi} L(\sigma, \pi) = \sum_t \alpha_\sigma[t, i] a_{ij} b_j(\sigma_{t+1}) \beta_\sigma[t+1, j]$$



E-step of Baum-Welch

- E-step is to compute Q function, but Baum-Welch instead the expectation values can be computed
- That is, expectation values on the state transition from *i* to *j* :

$$\sum_{\pi\in\Xi|i\to j\in\pi} L(\sigma,\pi) = \sum_t \alpha_\sigma[t,i]\, a_{ij}\, b_j(\sigma_{t+1})\, \beta_\sigma[t+1,j]$$



Baum-Welch Algorithm

1. Choose initial values for probability parameters
2. Repeat E- and M-steps alternately
   - E-step:
     - Computes expectation values (#counts) for each state transition (or letter generation)
   - M-step:
     - Updates probability parameters using expectation values

Derivation of Baum-Welch (M-step)

- Derived Q function:

$$Q(\phi;\phi') = \sum_{i,j} \log(a'_{ij}) \sum_{\pi\in\Xi|i\to j\in\pi} L_\phi(\sigma,\pi)$$

  - The problem is to maximize

$$f(x_1,\dots,x_K) = \sum_i^K c_i \log(x_i)$$

  - This problem is maximized by $x_i = \frac{c_i}{\sum_i^K c_i}$ if $\sum_i x_i = 1, x_i \ge 0$

- This directly derives the updating rule of M-step:

$$\hat{a}_{ij} = \frac{\sum_{\pi\in\Xi|i\to j\in\pi} L_\phi(\sigma,\pi)}{\sum_{\pi\in\Xi} L_\phi(\sigma,\pi)} = \frac{\sum_t \alpha_t(i) a_{ij} b_j(\sigma_{t+1})\beta_{t+1}(j)}{\sum_{i,j,t} \alpha_t(i) a_{ij} b_j(\sigma_{t+1})\beta_{t+1}(j)} = \frac{\sum_t \alpha_t(i) a_{ij} b_j(\sigma_{t+1})\beta_{t+1}(j)}{\sum_t \alpha_t(i)\beta_t(i)}$$

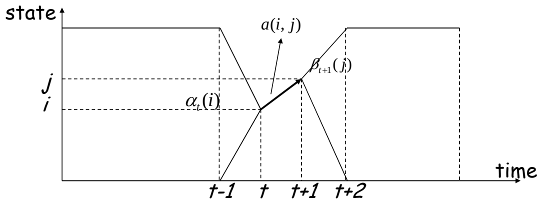
M-step of Baum-Welch

- Update state transition probability by using the expectation value and the likelihood

$$\hat{a}_{ij} = \frac{\sum_t \alpha_t(i) a_{ij} b_j(\sigma_{t+1})\beta_{t+1}(j)}{\sum_t \alpha_t(i)\beta_t(i)}$$

Likelihood of all paths with $i \to j$

Likelihood of all paths



Baum-Welch Algorithm

1. Choose initial values for probability parameters
2. Iterates E- and M-steps alternately until convergence
   - E-step:
     1. Compute <u>forward probabilities</u>: $\alpha_\sigma[t,i]$
     2. Compute <u>backward probabilities</u>: $\beta_\sigma[t,j]$
     3. Compute the <u>expectation value</u> of state transition from *i* to *j* using forward and backward probabilities:

$$E_{P_\sigma}[\#((i,j),\sigma] \propto \sum_t \alpha_\sigma[t,i]\, a_{ij}\, b_j(\sigma_{t+1})\, \beta_\sigma[t+1,j]$$

   - M-step:
     1. Update <u>transition probability</u> a_{ij} using expectation values:

$$\hat{a}_{ij} = \frac{E_{P_\sigma}[\#((i,j),\sigma]}{\sum_j E_{P_\sigma}[\#((i,j),\sigma]}$$

Summary of Baum-Welch

- Algorithm for estimating probability parameters of HMM
  - i.e. Algorithm for training HMM
- EM (Expectation-Maximization) algorithm, meaning that the solution is local optimum of maximum likelihood
- Makes simple enumeration efficient by dynamic programming: $O(M^T) \to O(M^3)$

Parsing for HMM

- Given a string, we can compute likelihoods for all possible state transition paths
- Among them, we call the state transition which gives the maximum the maximum likelihood path, which is exactly the solution of parsing
- Question: How can we compute that efficiently?

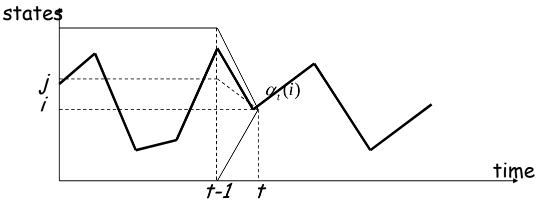
Parsing for HMM

- Question: How can we compute that efficiently?
- If we try to enumerate all possible state transition paths, computational hardness again!
- Solution:
  - Remember forward probabilities
  - Replace $\sum$ with `max'
  - Keep the maximum path

$$\alpha_{t+1}(j) = \sum_i \alpha_t(i) a_{ij} b_j(\sigma_{t+1}) \Rightarrow \alpha_{t+1}(j) = \max_i \alpha_t(i) a_{ij} b_j(\sigma_{t+1})$$

Parsing for HMM

- Viterbi Algorithm
  - Computing maximum at each time (letter) and remember the previous state so that the maximum path is traceable finally



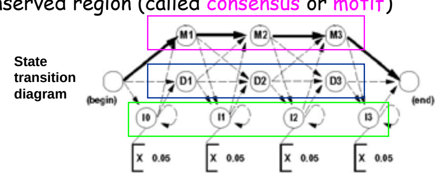
$$\alpha_{t+1}(j) = \sum_i \alpha_t(i) a_{ij} b_j(\sigma_{t+1}) \Rightarrow \alpha_{t+1}(j) = \max_i \alpha_t(i) a_{ij} b_j(\sigma_{t+1})$$

Three Problems for Hidden Markov Model

1. Computing likelihood:
   - Computing forward probabilities until the last letter of a given string
2. Learning
   - Maximizing the likelihood by Baum-Welch, an EM (Expectation-Maximization) algorithm
3. Parsing
   - Viterbi algorithm, a modification of computing forward probabilities

Example: Profile HMM

- Allows to align multiple string (amino acid sequences) to find conserved region (called consensus or motif)

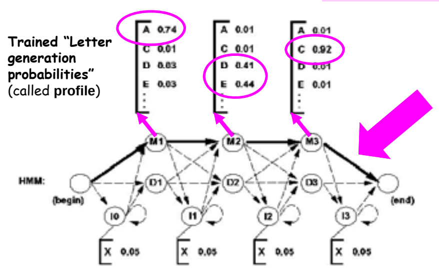


- Three types of states:
  - M: **normal** state, for important (conserved) amino acids
  - D: any letter not generated, for amino acids deletion
  - I: a letter generated according to a **fixed uniform** distribution, for unimportant (unconserved) amino acids

Training Profile HMM

Training strings example:

ADTC
WAEC
VEC
ADC
AEC

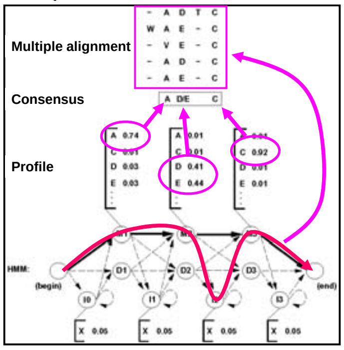


Consensus by Profile HMM

- Find consensus from M states
- Have multiple alignment by checking the most likely state path
  - Ex. ADTC:

    A (M1:0.74) →
    D (M2:0.41) →
    T (I2:0.05) →
    C (M3:0.92)

Parsing!



Final Remark

- Three Problems for probabilistic models in machine learning
  1. Computing likelihood
  2. Learning
  3. Parsing (prediction)
- Define hidden Markov model (HMM)
- Three problems of HMM
  - Computing likelihood by forward probabilities
  - Learning by Baum-Welch
  - Parsing by Viterbi
- Example: Profile HMM