Hidden Markov Models

Dr. Holger Fröhlich

SS2016

CpG-Islands

- Given 4 nucleotides: probability of occurrence is ~ 1/4. Thus, probability of occurrence of a dinucleotide is ~ 1/16.
- However, the frequencies of dinucleotides in DNA sequences vary widely.
- In particular, *CG* is typically underepresented (frequency of *CG* is typically < 1/16)

Why *CpG*-Islands?

- *CG* is the least frequent dinucleotide because *C* in *CG* is easily *methylated and* has the tendency to mutate into T afterwards
- However, the methylation is suppressed around genes in a genome. So, *CG* appears at relatively high frequency within these *CpG* islands
- So, finding the *CpG* islands in a genome is an important problem
- How can we detect patterns in DNA or protein sequences?

CpG Islands and the "Fair Bet Casino"

- The *CpG* islands problem can be modeled after a problem named ***"The Fair Bet Casino"***
- The game is to flip coins, which results in only two possible outcomes: **H**ead or **T**ail.
- The **F**air coin will give **H**eads and **T**ails with same probability ½.
- The **B**iased coin will give **H**eads with prob. ¾.

The "Fair Bet Casino" (cont'd)

- Thus, we define the probabilities:
  - $P(H|F) = P(T|F) = ½$
  - $P(H|B) = ¾$, $P(T|B) = ¼$
  - The crooked dealer chages between Fair and Biased coins with probability 10%

The Fair Bet Casino Problem

- **Input:** A sequence $y = y_1 y_2 y_3 \ldots y_T$ of coin tosses (***H*** or ***T***) made by two possible coins (***F*** or ***B***).
  - Note: I will equivalently use the notation y_t and y(t) for a particular event at time t.

- **Output:** A sequence $x = x_1\, x_2\, x_3 \ldots\, x_T$, with each x_i being either *F* or *B* indicating that y_t is the result of tossing the Fair or Biased coin respectively.

P(y|fair coin) vs. P(y|biased coin)

- Suppose first that dealer never changes coins. Some definitions:
  - P(y|fair coin): prob. of the dealer using the *F* coin and generating the outcome *y*.
  - P(*y*|biased coin): prob. of the dealer using the *B* coin and generating outcome *y*.

P(*y*|fair coin) vs. P(*y*|biased coin)

- P(*y*|fair coin)=P($y_1 \ldots y_T$|fair coin)

  $\Pi_i p\,(y_i|\text{fair coin}) = (1/2)^T$

- P(*y*|biased coin)= P($y_1 \ldots y_T$|biased coin)=

$\Pi_i p\,(y_i|\text{biased coin}) = (3/4)^k(1/4)^{T-k} = 3^k/4^T$

  - *k* - the number of *H*eads in *x*.

Log-odds Ratio

- We define ***log-odds ratio*** as follows:

$$\log_2(\mathrm{P}(y \mid \text{fair coin}) / \mathrm{P}(y \mid \text{biased coin})) = \Sigma^{k}_{i=1} \log_2(p^{+}(y_i) / p^{-}(y_i)) = T - k \log_2 3$$

- At which point is the log-odds ratio 0, i.e. fair and bias coin have the same probability?

$$\mathrm{P}(y \mid \text{fair coin}) = \mathrm{P}(y \mid \text{biased coin})$$

$$1/2^T = 3^k/4^T$$

when $k = T / \log_2 3$ ($k \sim 0.67T$)

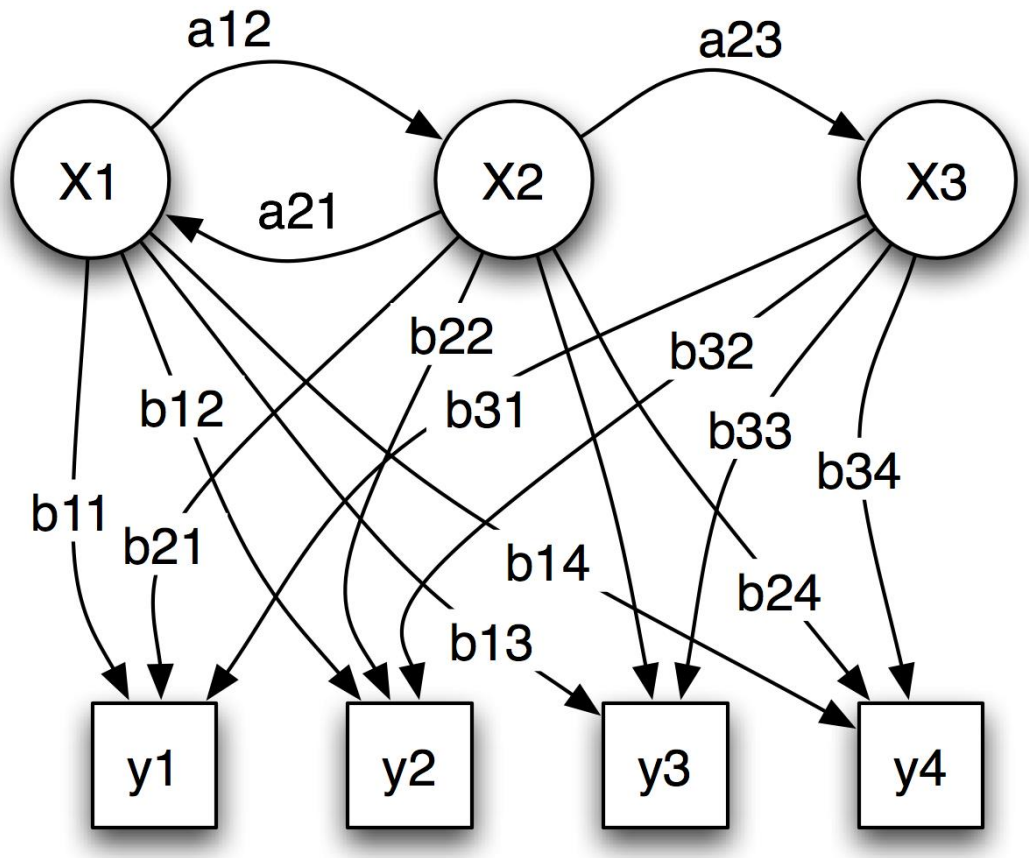
Fair-Bet Casino with Coin Switching

- The above described calculation assumes that the dealer never switches coins (fair to biased or biased to fair).
- What changes, if the dealer is switching coins with some probability?
  - Observed number of heads depends on type of coin (called *state*) used!
  - In other words: P(H | fair coin) is different from P(H | biased coin)

Hidden Markov Model (HMM)

- Can be viewed as an abstract machine with *k hidden* states that emits symbols from an alphabet Σ.
- Each state has its own probability distribution, and the machine switches between states according to this probability distribution.
- While in a certain state, the machine makes 2 decisions:
  - What state should I move to next?
  - What symbol - from the alphabet Σ - should I emit?

HMM - Example

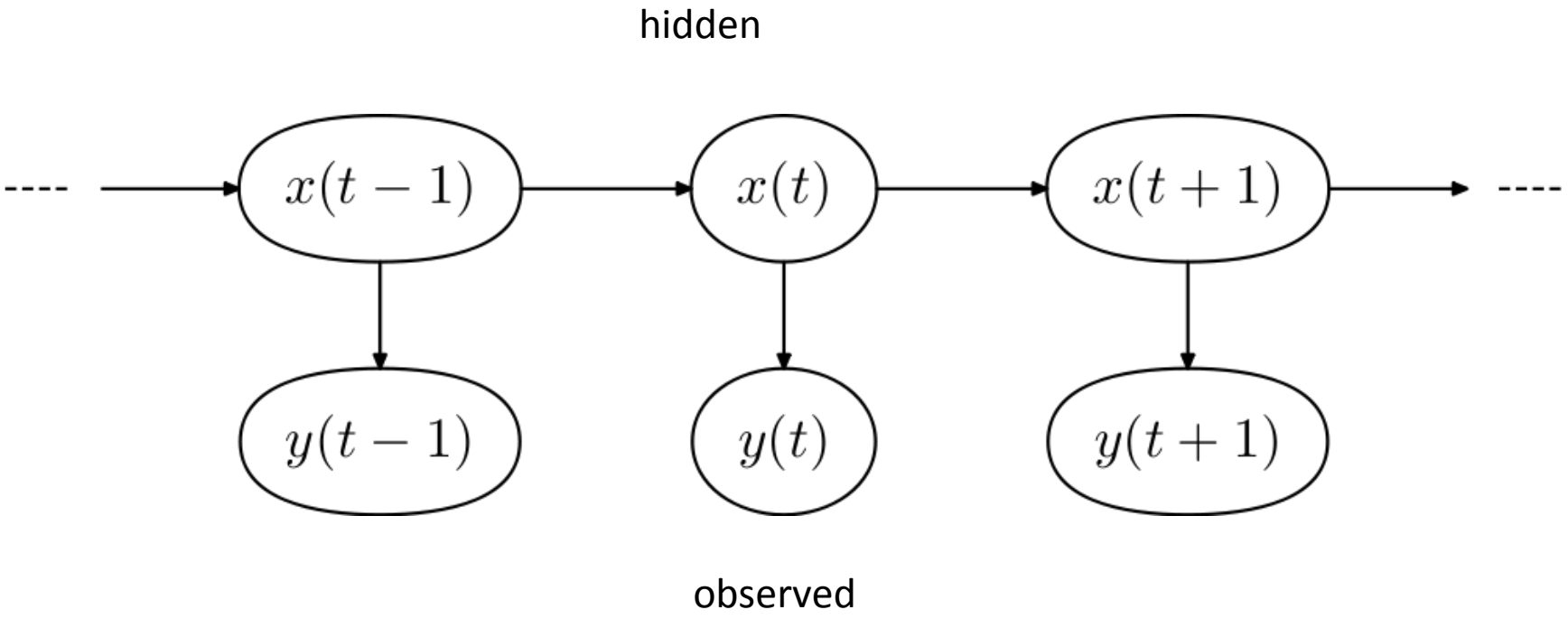


hidden

observed

a: transition probabilities
b: emission probabilities

Temporal View



Why “Hidden” Markov Machine?

- Observers can see the emitted symbols of an HMM but have *no ability to know which state the HMM is currently in*.
- ➔ Goal 1: infer most likely hidden states based on given sequence of emitted symbols
- ➔ Goal 2: calculate probability of a particular output sequence
- ➔ Goal 3: infer state transitions and output probabilities (i.e. learn model from data)

HMM Representation

- A HMM is represented by a tuple (Σ, Q, A, B)

Σ: set of emission characters.

Ex.: Σ = {H, T} for coin tossing

Σ = {1, 2, 3, 4, 5, 6} for dice tossing

Q: set of hidden states, each emitting symbols from Σ.

Q={F, B} for coin tossing

HMM Parameters

A = (a_{kl}): a |Q| x |Q| matrix of probability of changing from state *k* to state *l*.

$a_{FF} = 0.9$ $a_{FB} = 0.1$

$a_{BF} = 0.1$ $a_{BB} = 0.9$

B = (b_{kq}): a |Q| x |Σ| matrix of probability of emitting symbol *q* while being in state *k*.

$b_{FH} = ½$ $b_{FT} = ½$

$b_{BH} = ¼$ $b_{BT} = ¾$

Example: HMM for Fair Bet Casino

- The *Fair Bet Casino* in *HMM* terms:

  Σ = {T, H} (T for ***T***ails and H ***H***eads)

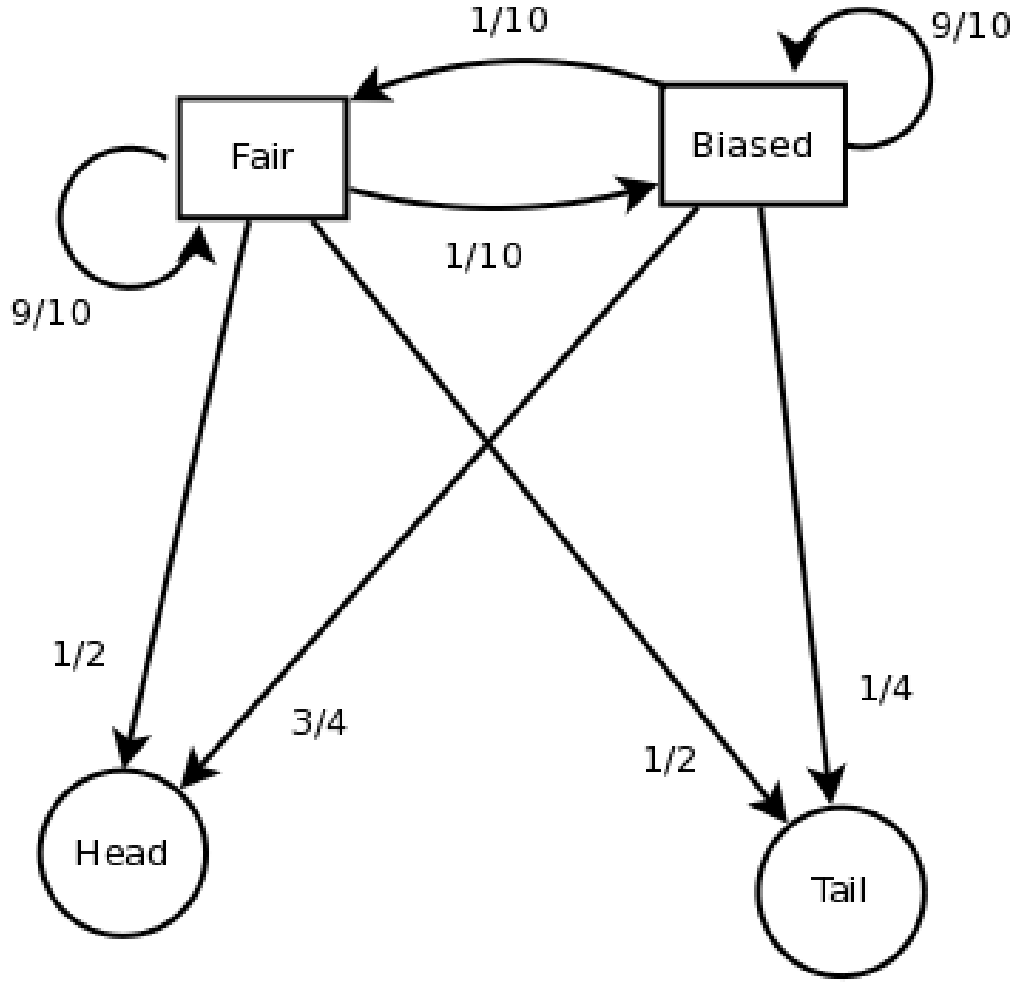
  Q = {*F, B*} – *F* for Fair & *B* for Biased coin.

- Transition Probabilities *A;* Emission Probabilities *B*

	Fair	Biased
Fair	$a_{FF} = 0.9$	$a_{FB} = 0.1$
Biased	$a_{BF} = 0.1$	$a_{BB} = 0.9$

	Tails	Heads
Fair	$b_{FT} = ½$	$b_{FH} = ½$
Biased	$b_{BT} = ¼$	$b_{BH} = ¾$

Representation as Graphical Model



Graphical Models

- HMMs are so-called **graphical models** defining probability distributions via graph structures.
- Other graphical models:
  - Bayesian networks (more general than HMMs)
  - Markov Random Fields (undirected graphical models)
  - Factor graphs (bipartite graphical models)
- Graphical models are frequently used in machine learning and widely applied in Bioinformatics
- General idea: edges describe **conditional statistical variable dependences**

(Conditional) Independence

- Statistical independence:

$$\Pr(X, Y) = \Pr(X)\Pr(Y) \text{ (probabilities)}$$

$$p(A, B) = p(A)p(B) \text{ (densities)}$$

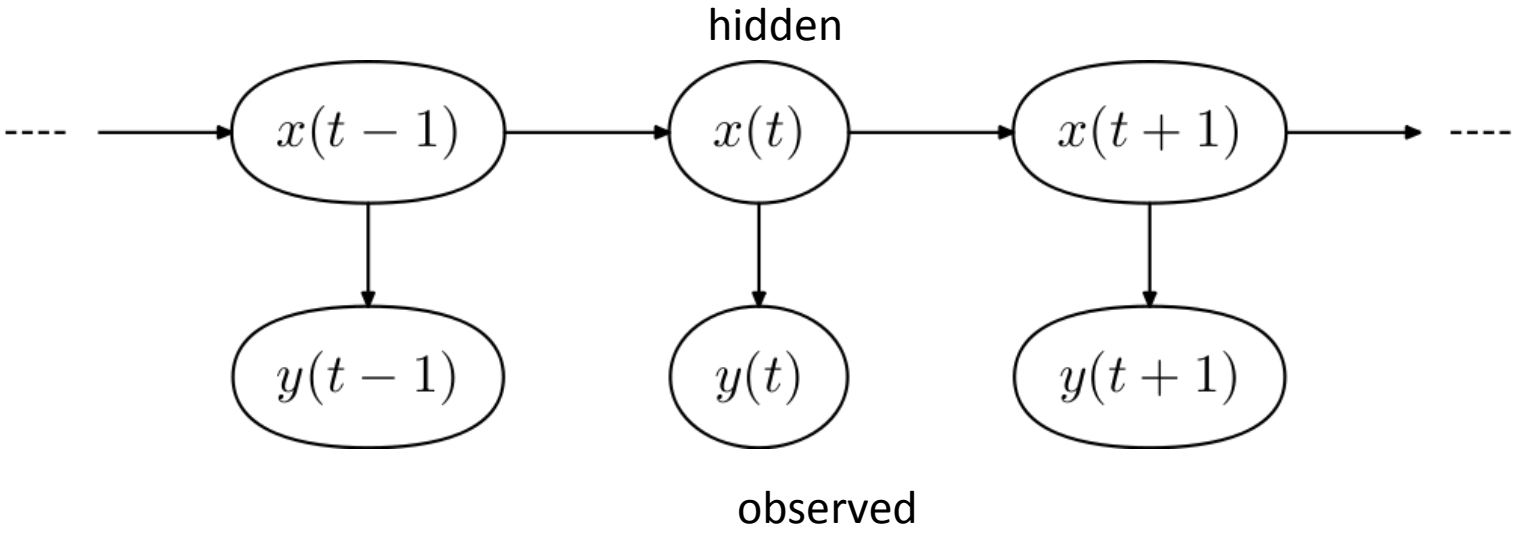
- Conditional independence:

$$\Pr(X, Y \mid W) = \Pr(X \mid W)\Pr(Y \mid W)$$

$$p(A, B \mid C) = p(A \mid C)p(B \mid C)$$

- Short notation: $X \perp\!\!\!\perp Y \mid W$ or $X \perp Y \mid W$

Conditional Independence in HMMs



- observation y(t) only depends on state x(t)
  - It is conditionally independent from all other observations
- Hidden state x(t) only depends on state x(t-1)
  - It is conditionally independent from all other states (**1st order Markov property**)

Hidden Paths

- A *path* $x = x_1 \ldots x_T$ in the HMM is defined as a sequence of hidden states.
- Consider path x = FFFBBBBBFFF and sequence y = THTHHHTHTTH

Probability that y_i was emitted from state x_i

y		T	H	T	H	H	H	T	H	T	T	H
x	=	F	F	F	B	B	B	B	B	F	F	F
$P(y_i \mid x_i)$		½	½	½	¾	¾	¾	¼	¾	½	½	½
$P(x_{i-1} \rightarrow x_i)$		½	9/10	9/10	1/10	9/10	9/10	9/10	9/10	1/10	9/10	9/10

Transition probability from state x_{i-1} to state x_i

Likelihood of output sequence for <u>given</u> hidden state sequence

- P(*y*,X=x): Probability to observe *y* with given hidden state sequence *x (*drawn from random variable X*)* in HMM model M

$$P(y, X = x) = \prod_{t=1}^{T} P(y(t) \mid x(t)) P(x(t-1) \rightarrow x(t))$$

$$= \prod_{t=1}^{T} b_{x(t),y(t)} a_{x(t-1),x(t)}$$

- $a_{x(0),x(1)}$: probability to move from an initial fictitious state x(0) to x(1) (= probability $\pi_{x(1)}$ to be initially in state x(1)).

Decoding Problem

- **Goal:** Find most probable hidden path of states for given observations.
- **Input:** Sequence of observations $y = y_1 \ldots y_T$ generated by an HMM $M(\Sigma, Q, A, B)$
- **Output:** A path that maximizes $P(y,X=x)$ over all possible paths $x \rightarrow$ *maximum likelihood estimate*

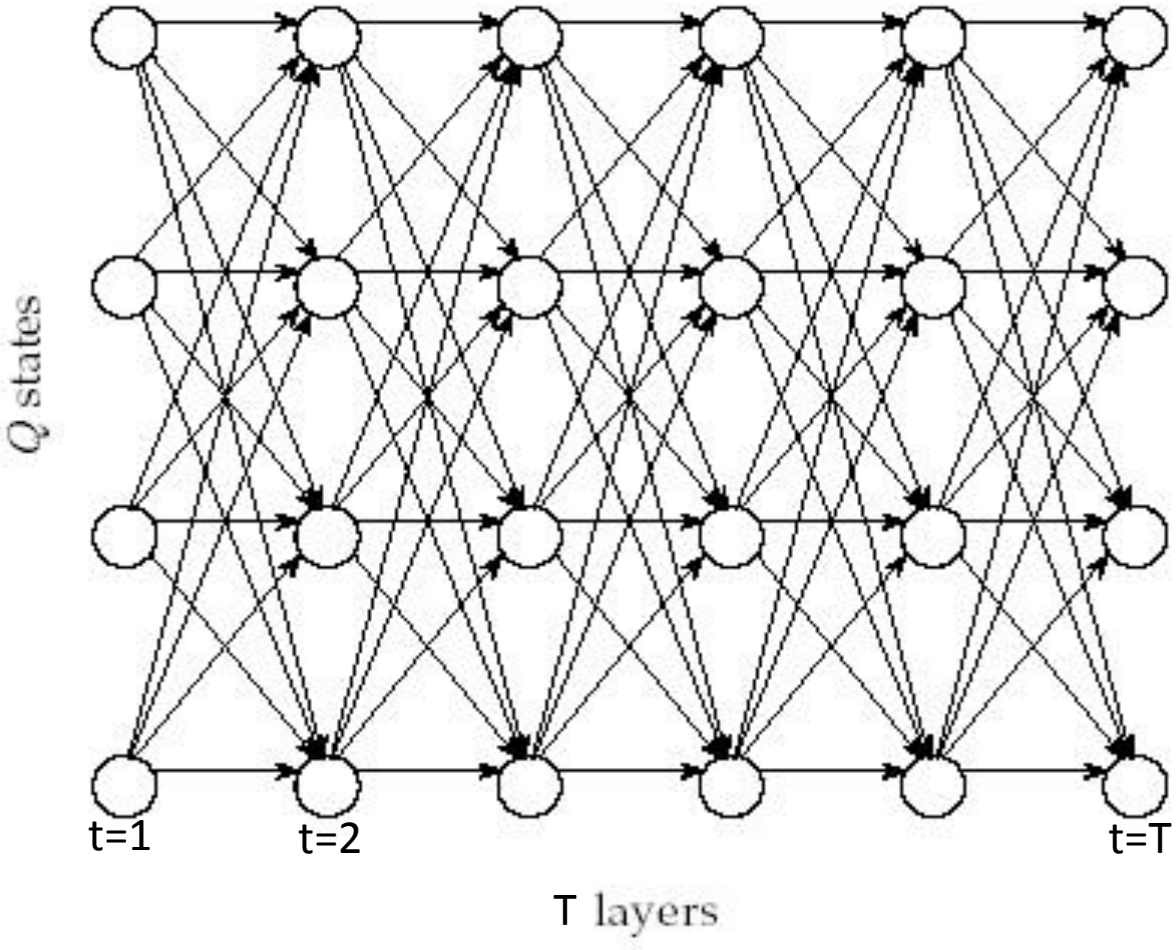
Decoding Problem

- How can we find the x that **maximizes** P(y,x)? (maximum a-posteriori estimate)
- x is a discrete set ➔ cannot compute derivatives
- Naive solution: enumerate all possible hidden state sequences and look for the one with highest log-likelihood
- Problem: $|Q|^T$ possible hidden state sequences ➔ impossible exhaustive search

Dynamic Programming

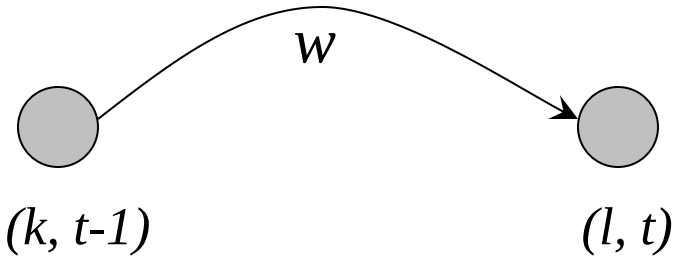
- Andrew Viterbi (1967): Dynamic Programming to solve the *Decoding Problem*
- Every choice of $x = x_1 \ldots x_T$ corresponds to a path in an edit graph.
- The only valid direction in the graph is *eastward.*
- This graph has $|Q|^2(T-1)$ edges.

Edit Graph for Decoding Problem - unfolding over time



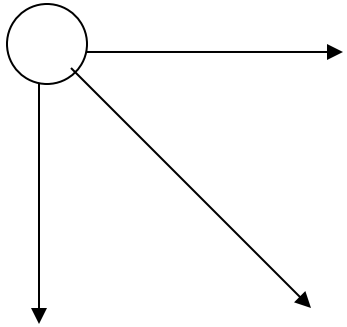
Edge Weights in Edit Graph

t-th term = $b_{x(t),y(t)} \cdot a_{x(t-1),x(t)} = b_{l,y(t)} \cdot a_{kl}$ *for* $x(t-1)=k, x(t)=l$

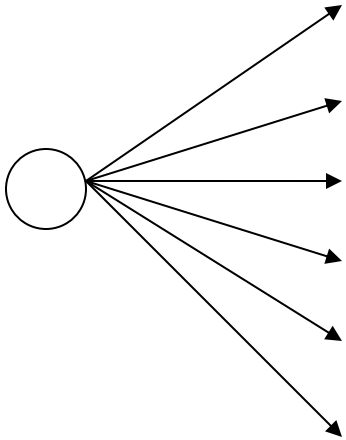


The weight $w = b_{l,y(t)} \cdot a_{kl}$

Decoding Problem vs. Alignment Problem



Valid directions in the *alignment problem.*



Valid directions in the *decoding problem.*

Decoding Problem as Finding a Longest Path in a DAG

- The *Decoding Problem* is reduced to finding a longest path in the *directed acyclic graph (DAG)* edit graph.

- Notes: the length of the path is defined as the *product* of its edges' weights, not the *sum*.

Decoding Problem and Dynamic Programming Recurrence

- Let $s_{l,t}$ denote the probability of the most likely path, which is ending in vertex (l,t).
- We have:

$s_{l,t}$ = $\max_{k \in Q} \{s_{k,t-1} \cdot$ weight of edge between *(k,t-1)* and *(l,t)*$\}$

$= \max_{k \in Q} \{s_{k,t-1} \cdot a_{kl} \cdot b_{l,y(t)} \}$

$= b_{l,y(t)} \cdot \max_{k \in Q} \{s_{k,t-1} \cdot a_{kl}\}$

Numerical Issues

- The value of the product can become extremely small, which leads to overflowing.
- To avoid overflowing, use log value instead.

$$s_{l,t} = \log b_{l,y(t)} + \max_{k \in Q} \{s_{k,t-1} + \log(a_{kl})\}$$

Viterbi Algorithm (Pseudocode)

Algorithm Viterbi (y, A, B, π)

$s_{k,1} = \log \pi_k + \log b_{k,y(1)}$ for k=1,…,|Q| // initialization

for t = 2,…,T:

  for l = 1,…,Q:

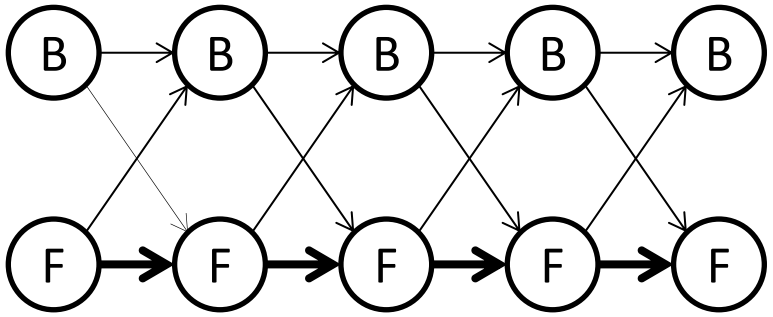
    $s_{l,t} = \log b_{l,y(t)} + \max_{k \in Q} \{s_{k,t-1} + \log(a_{kl})\}$

j = arg $\max_k s_{k,T}$ Likelihood of optimal path

path = backtracking path from $s_{j,T}$

return {path, $s_{j,T}$}

Example: Viterbi Algorithm for Fair Bet Casino Problem



	F	B
F	a_{FF} = 0.9	a_{FB} = 0.1
B	a_{BF} = 0.1	a_{BB} = 0.9

Observed sequence

H T H T T

Log – likelihood (s)

F	-1.39 ←	-2.18 ←	-2.98 ←	-3.78 ←	-4.58
B	-0.98	-2.47	-2.86	-4.36	-5.85

	T	H
F	b_{FT} = ½	b_{FH} = ½
B	b_{BT} = ¼	b_{BH} = ¾

Viterbi sequence

F F F F F

Initial state probabilities:
$\pi_F = 0.5$, $\pi_B = 0.5$

Likelihood of viterbi sequence: -4.58

What is the probability to observe a sequence under a given HMM model?

- Recall: for given hidden state sequence x we have:

$$P(y, X = x) = \prod_{t=1}^{T} P(y(t) \mid x(t)) P(x(t-1) \rightarrow x(t))$$

- P(y) can thus be computed by marginalization:

$$P(y) = \sum_{x \in X} P(y, X = x)$$

- **Problem:** sum runs over all possible hidden state sequences!
- This is practically infeasible to compute: $|Q|^T$ possible hidden state sequences

Solution: Forward Algorithm

- $f_{k,t}$ (*forward probability*) = probability of emitting the *prefix* $y_1 \ldots y_t$ and reaching the state $x(t) = k$.

$$f_{k,t} = P(y(1),\ldots, y(t), x(t) = k)$$

$$\overset{\text{1st order Markov property}}{=} \left(\sum_{l \in Q} \left[P(y(1),\ldots, y(t-1), x(t-1) = l) * a_{lk} \right] \right) * P(y(t) \mid x(t) = k)$$

$$P(y(t) \mid x(t) = k) \uparrow b_{k,y(t)}$$

Sum over all possible transitions that could lead to state k

Forward Algorithm

- Recurrences for the forward algorithm:

$$f_{k,1} = \pi_k b_{k,y(1)}$$

$$f_{k,t+1} = b_{k,y(t+1)} \sum_{l \in Q} f_{l,t} a_{lk}$$

- Remark for implementation: Need to replace „max“ by sum in the Viterbi algorithm - be careful with logarithms (log-sum-exp trick):

$$log\left(\sum_{i=1}^{n} p_i\right) = log\ p_{\max} + log\left(\sum_{i=1}^{n} e^{log\ p_i - log\ p_{\max}}\right)$$

What is the probability to observe a sequence under a given HMM model?

- Note that

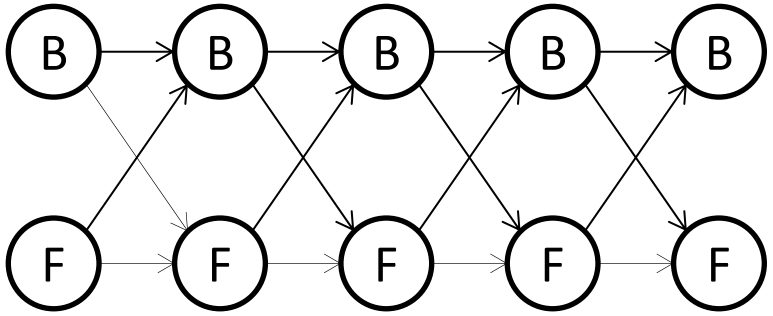
$$f_{k,T} = P(y(1),\ldots, y(T), x(T) = k)$$

$$\Rightarrow$$

$$P(y) = P(y(1),\ldots, y(T)) = \sum_{k \in Q} f_{k,T}$$

- Probability of sequence y can be computed by summing forward probabilities over all hidden states at the last time point.
  - There are just Q summands!

Example: Forward Algorithm for Fair Bet Casino Problem



Observed sequence

H T H T T

Log forward probabilities

F	-1.39	-2.03	-2.76	-3.44	-4.18
B	-0.98	-2.4	-2.64	-4.04	-5.35

Probability to observe sequence under given HMM model: exp(-4.18) + exp(-5.35) = 0.02

	F	B
F	a_{FF} = 0.9	a_{FB} = 0.1
B	a_{BF} = 0.1	a_{BB} = 0.9

	T	H
F	b_{FT} = ½	b_{FH} = ½
B	b_{BT} = ¼	b_{BH} = ¾

Initial state probabilities: $\pi_F = 0.5$, $\pi_B = 0.5$

Forward-Backward Problem

Given: a sequence of coin tosses generated by an HMM.

Goal: find the probability that the dealer was using a biased coin at a particular time, i.e. $P(x(t)=k \mid y)$.

Backward Algorithm

- *Forward probability* is only <u>one</u> factor affecting $P(x(t) = k|y)$.

- Sequence of transitions and emissions that the HMM undergoes between $x(t+1)$ and $x(T)$ also affect $P(x(t)= k|y)$.

forward y_t backward

Backward Algorithm (cont'd)

- *Backward probability* $p_{k,t}$ = probability of emitting the *suffix* $y_{t+1} \ldots y_T$ given $x(t) = k$

$$p_{k,t} = P(y(t+1),\ldots, y(T) \mid x(t) = k)$$

$$= \sum_{l \in Q} a_{kl} * P(y(t+1) \mid x(t+1) = l) * P(y(t+2),\ldots, y(T) \mid x(t+1) = l)$$

- Recurrences for the *backward algorithm*:

$$p_{k,T} = 1$$

$$p_{k,t} = \sum_{l \in Q} a_{kl} b_{l,y(t+1)} p_{l,t+1}$$

Forward-Backward Probabilities

- The probability that the dealer used a biased coin at any moment t:

$$P(x(t)=k|y) = \frac{P(y, x(t)=k)}{P(y)} = \frac{f_{k,t} \cdot p_{k,t}}{P(y)}$$

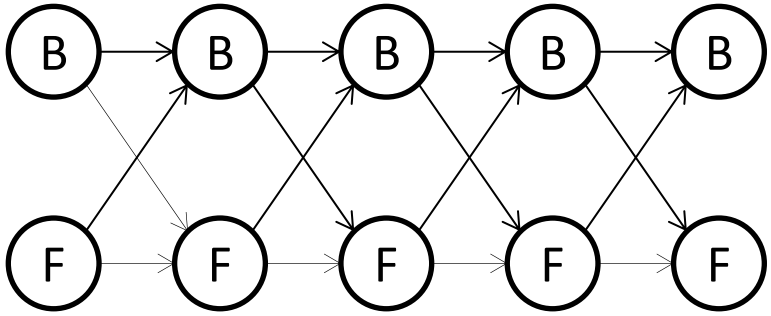
$$P(y) = \sum_{k \in Q} P(x(t)=k, y) = \sum_{k \in Q} f_{k,t} \cdot p_{k,t} \quad \text{for } t = 1,\ldots,T$$

$$= \sum_{k \in Q} f_{k,T}$$

$$= \sum_{k \in Q} \pi_k b_{k,y(1)} p_{k,1}$$

Why?

Example: Backward Algorithm for Fair Bet Casino Problem



Observed sequence

H T H T T

Log backward probabilities

F	-3.01	-2.25	-1.51	-0.74	0
B	-3.87	-2.68	-2.46	-1.29	0

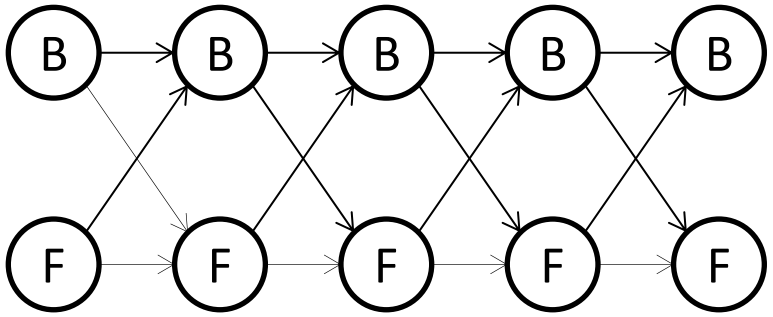
Probability to observe sequence under given HMM model: exp(-3.01)*0.5*0.5 + exp(-3.87)*0.75*0.5 = 0.02

	F	B
F	a_{FF} = 0.9	a_{FB} = 0.1
B	a_{BF} = 0.1	a_{BB} = 0.9

	T	H
F	b_{FT} = ½	b_{FH} = ½
B	b_{BT} = ¼	b_{BH} = ¾

Initial state probabilities: $\pi_F = 0.5$, $\pi_B = 0.5$

Example: Fw./Bw. Algorithm for Fair Bet Casino Problem



Observed sequence

H T H T T

Posterior state probabilities

F	0.61	0.69	0.7	0.76	0.76
B	0.39	0.31	0.3	0.24	0.24

	F	B
F	a_{FF} = 0.9	a_{FB} = 0.1
B	a_{BF} = 0.1	a_{BB} = 0.9

	T	H
F	b_{FT} = ½	b_{FH} = ½
B	b_{BT} = ¼	b_{BH} = ¾

Initial state probabilities:
$\pi_F = 0.5$, $\pi_B = 0.5$

Forward-Backward algorithm allows for estimating posterior state probabilities!

Viterbi vs.Posterior Decoding

- Viterbi: most likely hidden path: arg $\max_x$ P(y,x)
  - Issue: there may exist several path with (almost) identical maximum likelihood
  - Viterbi returns only one
- Posterior decoding: look at state assignment at particular position
  - Path {k | k = arg $\max_l$ P(x(t)=l | y)} is **not** the same
  - It may not even correspond to a legitimate path!

HMM Parameter Estimation

- So far, we have assumed that the transition and emission probabilities are known.

- However, in most HMM applications, the probabilities are not known.

HMM Parameter Estimation Problem

- Given
  - HMM with states and alphabet (emission characters)
  - Training sequence Y
- Find HMM parameters Θ (that is, A, B) that maximize

$$P(Y \mid \Theta)$$

Assume Hidden Paths to be Known

A_{kl} = # of times each $k \rightarrow l$ is taken in the training sequences

$B_{k,c}$ = # of times c is emitted from state k in the training sequence

Compute a_{kl} and $b_{k,c}$ as ML estimators:

$$a_{kl} = A_{kl} / \sum_{l'} A_{kl'}$$

$$b_{k,c} = B_{k,c} / \sum_{b'} B_{k,b'}$$

ML Estimates of Parameters

- a_{kl} and $b_{k,c}$ are relative frequencies of finite number of observations in A and B
- a_{kl} and $b_{k,c}$ may not reflect the true probabilities (given infinite number of observation)
- Some state *k* may even not appear in training sequence. This means $A_{kl} = 0$ for every state *l.*
- ➔ HMM parameters are overfitted (do not generalize well)

Pseudocounts

- Use predetermined pseudocounts r_{kl} and $R_{k,c}$.

  A_{kl} = # of transitions $k \rightarrow l + r_{kl}$

  $B_{k,c}$ = # of emissions of b from $k + R_{k,c}$

- Interpretation in the Bayesian inference framework:
  - A_{kl} , $B_{k,c}$ can be viewed as samples drawn from multinomial distributions with parameters a_{kl} and $b_{k,c}$
  - Estimating of a_{kl} and $b_{k,c}$ via relative frequencies corresponds to a maximum likelihood estimate
  - Adding of pseudocounts corresponds to using a Dirichlet prior

Unknown paths: Baum-Welch Algorithm

Idea (Baum & Welch, 1966):

1. Guess initial values for parameters.

2. Estimate forward and backward probabilities.

3. Calculate expected values for A and B given these probabilities → new parameters.

3. Repeat until convergence (likelihood does not change significantly).

Properties of Baum-Welch Algorithm

- Algorithm is guarenteed to improve log-likelihood in each step
- BUT: can run into local optima
  - Try different start values
- BW algorithm is a special Expectation Maximization (EM) algorithm
  - Widely used technique in statistics:
    - Clustering
    - Mixture models
    - Missing value imputation
    - …

Updating Parameter Estimates: temporary variables

- Suppose we have calculated forward and backward probabilities
- Posterior probability to be in state k at time t:

$$\gamma_k(t) := P(x(t) = k \mid y) = \frac{f_{k,t}\, p_{k,t}}{\sum_{l \in Q} f_{l,T}}$$

Slide 54

- Probability for l→k transition at time t-1:

$$\begin{aligned}\xi_{lk}(t) &:= P(x(t-1) = l, x(t) = k \mid y) \\ &= P(x(t) = k \mid x(t-1) = l, y) P(x(t-1) = l, y) \\ &= \ldots \\ &\overset{\text{(several steps)}}{=} \frac{f_{l,t-1} a_{lk} b_{k,y(t)} p_{k,t}}{\sum_{k \in Q} f_{k,T}}\end{aligned}$$

Updating Parameter Estimates: Expectation Values

- Estimate expected number of transitions between *k* and *l* (can add pseudocounts):

$$\mathrm{E}[A_{kl} \mid Y] = \sum_{t=1}^{T} P(x(t-1) = l, x(t) = k \mid y)$$

$$\Rightarrow \mathrm{ML\ estimate\ of}\ a_{kl}:$$

$$\hat{a}_{kl} = \frac{\sum_{t=1}^{T} P(x(t-1) = l, x(t) = k \mid y)}{\sum_{t=1}^{T} \underbrace{\sum_{q \in Q} P(x(t-1) = l, x(t) = q \mid y)}_{= P(x(t-1) = l \mid y)}} = \frac{\sum_{t=1}^{T} \xi_{lk}(t)}{\sum_{t=1}^{T} \gamma_l(t)}$$

(posterior state probs)

Updating Parameter Estimates (cont.)

- Estimate expected number of times that *c* is emitted in state *k* (can add pseudocounts):

$$\mathrm{E}[B_{kc} \mid Y] = \sum_{t=1}^{T} P(x(t) = k \mid y)\delta_{y(t),c}$$

$$\delta_{y(t),c} = \begin{cases} 1 & y(t) = c \\ 0 & otherwise \end{cases}$$

$$\Rightarrow \mathrm{ML\ estimate\ of}\ b_{kc}:$$

$$\hat{b}_{kc} = \frac{\sum_{t=1}^{T} P(x(t) = k \mid y)\delta_{y(t),c}}{\sum_{t=1}^{T} P(x(t) = k \mid y)} = \frac{\sum_{t=1}^{T} \gamma_k(t)\delta_{y(t),c}}{\sum_{t=1}^{T} \gamma_k(t)}$$

Once Again: Baum-Welch Algorithm

Idea (Baum, L. E. and Petrie, T., 1966):

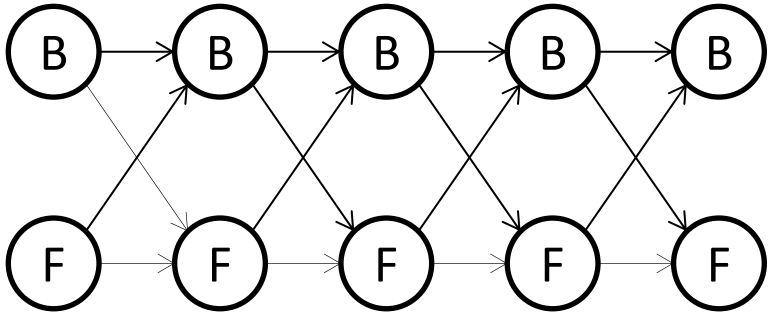
1. Guess initial values for parameters.

2. Estimate forward and backward probabilities.

3. Calculate expected values for A and B given these probabilities ➔ new parameters.

3. Repeat until convergence (likelihood does not change significantly).

Example: BW-Algorithm for Fair Bet Casino Problem (initialization)



Observed sequence

H T H T T

Posterior state probabilities γ

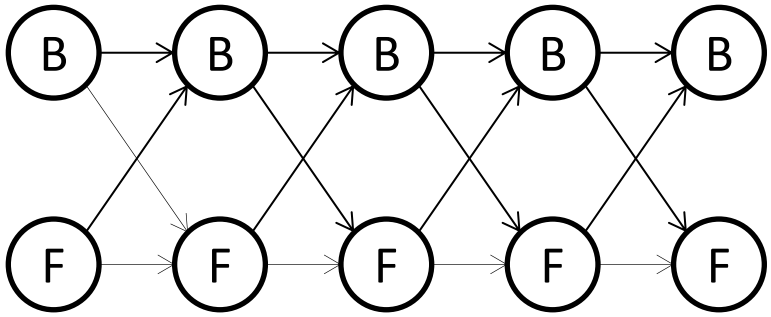
F	0.28	0.07	0.14	0.03	0.04
B	0.72	0.93	0.86	0.97	0.96

	F	B
F	a_{FF} = 0.75	a_{FB} = 0.25
B	a_{BF} = 0.25	a_{BB} = 0.75

	T	H
F	b_{FT} = *0.5*	b_{FH} = *0.5*
B	b_{BT} = *0.1*	b_{BH} = *0.9*

Initial state probabilities:
$\pi_F = 0.5$, $\pi_B = 0.5$

Example: BW-Algorithm for Fair Bet Casino Problem (1 iteration)



Observed sequence

H T H T T

Posterior state probabilities γ

F	0.57	0.29	0.33	0.22	0.21
B	0.43	0.71	0.37	0.78	0.79

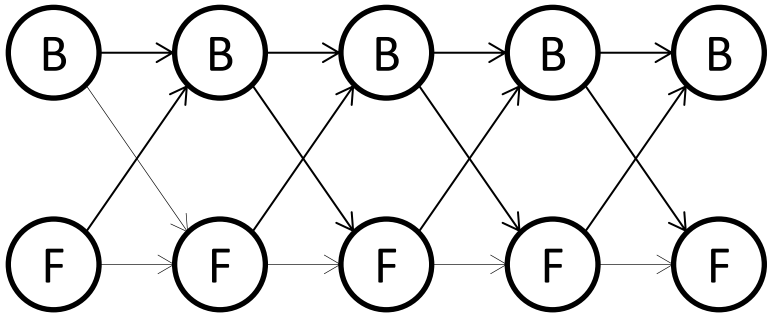
Log-likelihood difference: 0.91

	F	B
F	a_{FF} = 0.44	a_{FB} = 0.56
B	a_{BF} = 0.4	a_{BB} = 0.6

	T	H
F	b_{FT} = 0.56	b_{FH} = 0.44
B	b_{BT} = 0.6	b_{BH} = 0.4

Initial state probabilities:
$\pi_F = 0.5$, $\pi_B = 0.5$
Pseudo-count: 1

Example: BW-Algorithm for Fair Bet Casino Problem (2 iterations)



Observed sequence

H T H T T

Posterior state probabilities γ

F	0.57	0.35	0.43	0.32	0.31
B	0.43	0.65	0.57	0.68	0.69

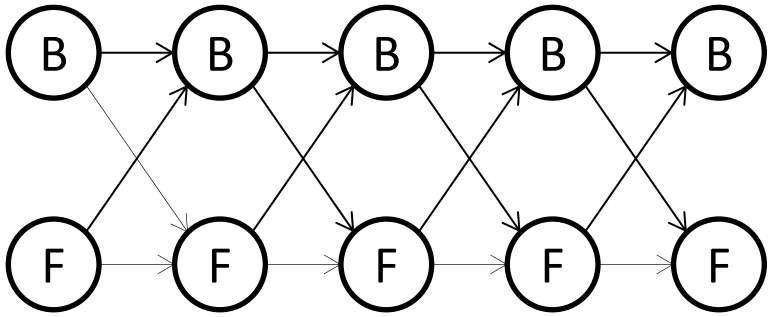
Log-likelihood difference: 0.21

	F	B
F	a_{FF} = 0.46	a_{FB} = 0.54
B	a_{BF} = 0.33	a_{BB} = 0.67

	T	H
F	b_{FT} = *0.52*	b_{FH} = *0.48*
B	b_{BT} = *0.39*	b_{BH} = *0.61*

Initial state probabilities:
$\pi_F = 0.5$, $\pi_B = 0.5$
Pseudo-count: 1

Example: BW-Algorithm for Fair Bet Casino Problem (100 iterations)



Observed sequence

H T H T T

Posterior state probabilities γ

F	0.5	0.5	0.5	0.5	0.5
B	0.5	0.5	0.5	0.5	0.5

	F	B
F	a_{FF} = 0.5	a_{FB} = 0.5
B	a_{BF} = 0.5	a_{BB} = 0.5

	T	H
F	b_{FT} = *0.56*	b_{FH} = *0.44*
B	b_{BT} = *0.56*	b_{BH} = *0.44*

Log-likelihood difference: 7.6E-10

Initial state probabilities:
$\pi_F = 0.5$, $\pi_B = 0.5$
Pseudo-count: 1

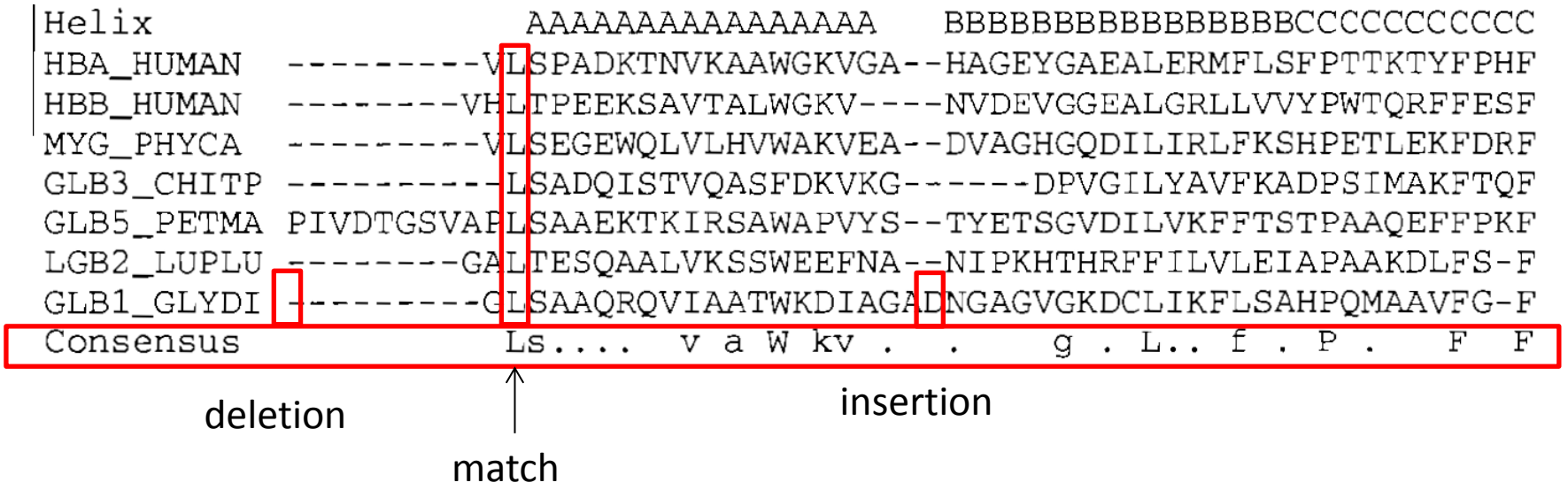
Alternative to BW: Viterbi Training

- BW: estimate forward and backward probabilities, then derive expected parameters
- Viterbi training:
  1. Estimate Viterbi path x*
  2. Treat x* as known path and estimate parameters
- Advantage: Exact convergence
- Disadvantage: Does **not** maximize $P(y \mid \Theta)$
- Typically performs less well than BW

Motivation: Finding Distant Members of a Protein Family

- A distant cousin of functionally related sequences in a protein family may have weak pair-wise similarities with each member of the family and thus fail significance test.
- However, they may have weak similarities with ***many*** members of the family.
- The goal is to align a sequence to ***all*** members of the family at once.
- Family of related proteins can be represented by their multiple alignment and the corresponding alignment profile.

Multiple Alignments



Profile Representation of Multiple Alignments

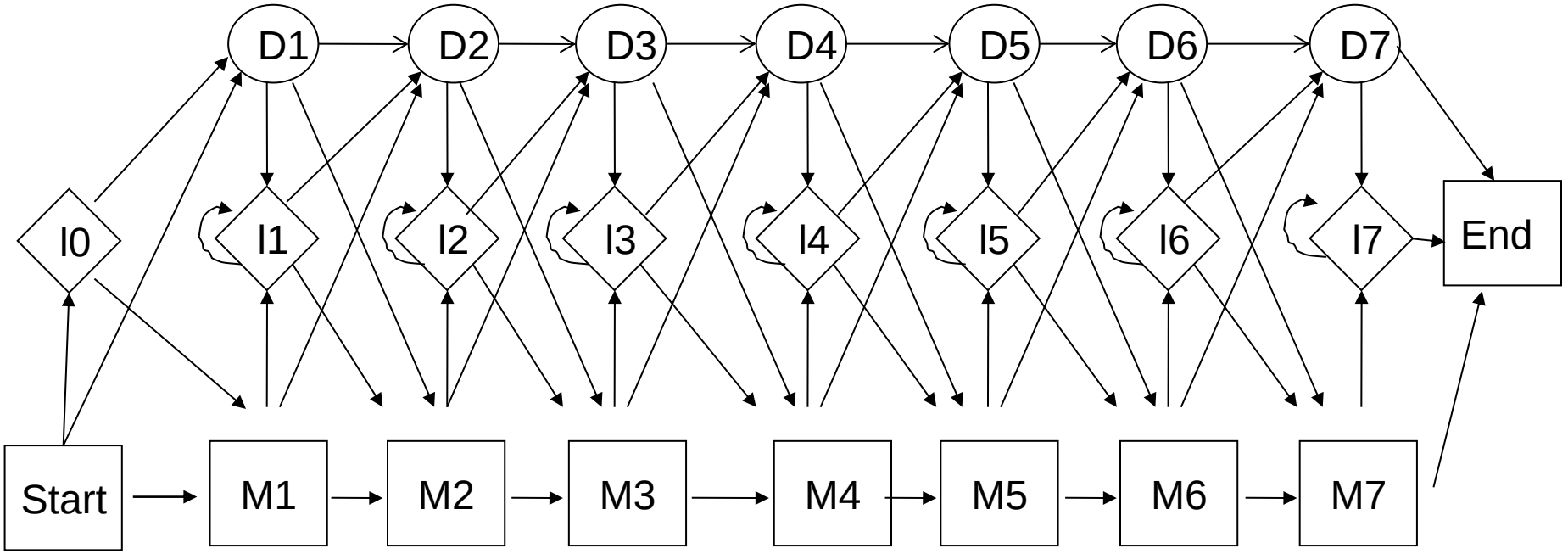
	1	2	3	...
A	0.1	0.2	0.2	
R	0.3	0.05	0.05	
…				

Protein family can be represented by a *20·n* profile representing frequencies of amino acids.

What are Profile HMMs ?

- A **Profile HMM** is a probabilistic representation of a multiple alignment.
- **Model can be used to find and score potential matches of new sequences.**
- Advantages of Profile HMMs compared to conventional alignments:
  - Higher quality
  - Probabilistic scoring scheme
- Disadvantage: higher computation time

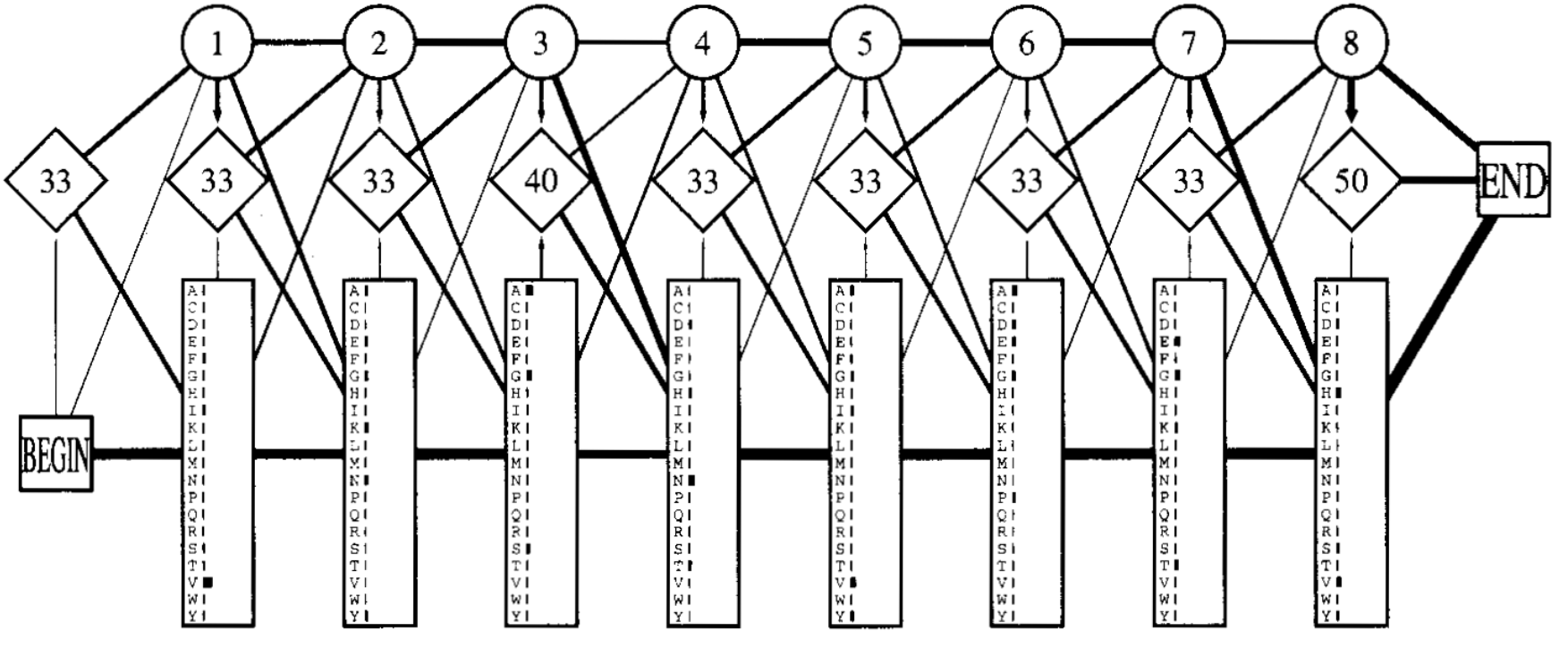
Profile HMM



- Number of states depends on length of consensus sequence

- Given query sequence x, for each position i in consensus sequence there are 3 possible states:
  - match (M_i): sequence x matches with consensus sequence at position i
  - insertion (I_i): in sequence x a letter is inserted after position i (i.e. gap in profile)
  - deletion (D_i): in sequence x a letter is deleted at position i

Example



Numbers in the middle row indicate transition probabilities between insertion states in percentages.

Interpretation of Transition Probabilities

- Let x be a query sequence we want to align against our HMM
- $log(a_{MI})+log(a_{IM})$ = cost for inserting additional letters in x ➔ gap opening in profile

- $log(a_{II})$ = penalty for gap extension in profile

- $log(a_{DI}) + log(a_{DM})$= costs for deleting characters in x ➔ gap opening in x

Building a Profile HMM from Multiple Alignments

1. Construct multiple alignment and its profile
2. Assign each column in the consensus sequence with sufficient fraction of letters to a *Match* state. Add I*nsertion* and *Deletion* states.
   - Insertion state I_1 models an insertion after the first column
   - Deletion state D_1 models a deletion in first column
3. Estimate the emission probabilities according to amino acid counts in column. Different positions in the protein will have different emission probabilities.
4. Estimate the transition probabilities between *Match, Deletion* and *Insertion* states via relative frequencies of matches and indels in each column

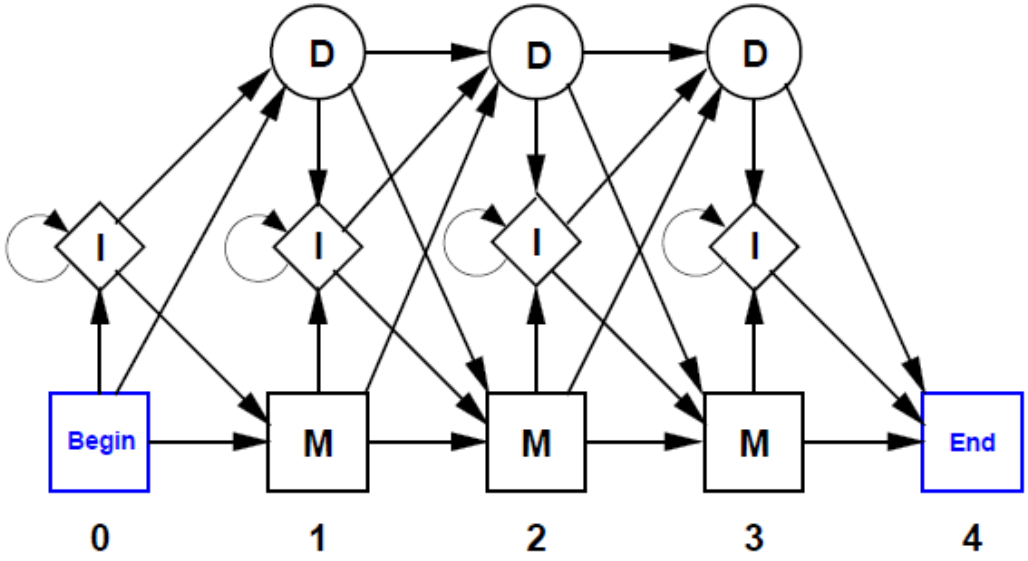
Example

1. Multiple sequence alignment:

```
bat     V E V - - L
rat     - E - E V L
cat     L E V - E -
gnat    L - L E L -
goat    - E V - L -
          . 1 2 . . 3
```

Only columns with sufficient fraction of matching letters are modeled as match states.

2. Profile HMM



State annotation:

```
bat     I M M D D M
rat     D M D I I M
cat     I M M D I D
gnat    I D M I I D
goat    D M M D I D
        0 1 2 . . 3
```

How to estimate parameters?

1. Estimate emission probability for letter a in match state z (maximum likelihood):

$$\hat{b}_{za} = \frac{B_{za}}{\sum_{e \in \Sigma} B_{ze}}$$

$$B_{za} = \#\,\text{observed emissions in state z}$$

2. Estimate transition probability from state k to l:

$$\hat{a}_{kl} = \frac{A_{kl}}{\sum_{r \in Q} A_{kr}}$$

$$A_{kl} = \#\,\text{observed transitions from k to l}$$

Previous Example

$$b_{M_1 E} = \frac{4}{4}$$

$$b_{I_0 V} = \frac{1}{3}$$

$$b_{I_0 L} = \frac{2}{3}$$

State annotation:

```
bat    I M M D D M
rat    D M D I I M
cat    I M M D I D
gnat   I D M I I D
goat   D M M D I D
       0 1 2 . . 3
```

$$a_{M_1 M_2} = \frac{\#\,\text{matches in column 1 that lead to matches in column 2}}{\text{all matches in column 1}} = \frac{3}{4}$$

$$a_{M_1 D_2} = \frac{\#\,\text{matches in column 1 that lead to deletions in column 2}}{\text{all matches in column 1}} = \frac{1}{4}$$

$$a_{D_2 M_3} = \frac{\#\,\text{deletions in column 2 that lead to matches in column 3}}{\text{all deletions in column 2}} = \frac{0}{1}$$

Other Example

Observed emission and transition frequencies:

		0	1	2	3
Match-emissions	E		4	-	-
	L		-	-	1
	V		-	3	-
	…		-	-	-
Insert emissions	E	-	-	3	-
	L	2	-	1	-
	V	1	-	1	-
	…	-	-	-	-
State-transitions	M-M	2	3	2	4
	M-D	-	1	-	-
	M-I	3	-	1	-
	I-M	2	-	2	-
	I-D	1	-	1	-
	I-I	-	-	2	-
	D-M		-	-	1
	D-D		1	-	-
	D-I		-	2	-

ML estimates of emission and transition probabilities:

		0	1	2	3
Match emissions	E		1.0	-	-
	L		-	-	1.0
	V		-	1.0	-
	…		-	-	-
Insert emissions	E	-	-	0.6	-
	L	0.67	-	0.2	-
	V	0.33	-	0.2	-
	…	-	-	-	-
State transitions	M-M	0.4	0.75	0.67	1.0
	M-D	-	0.25	-	-
	M-I	0.6	-	0.33	-
	I-M	0.67	-	0.4	-
	I-D	0.33	-	0.2	-
	I-I	-	-	0.4	-
	D-M		-	-	1.0
	D-D		1.0	-	-
	D-I		-	1.0	-

Pseudo-Counts

- **Problem**: ML estimates are 0 in many cases
- Earlier discussed solution: add k pseudo-counts (e.g. k=1)
- Modified estimates for emission and transition probabilities:

$$\hat{b}_{za} = \frac{B_{za} + k}{\sum_{e \in \Sigma} B_{ze} + |\Sigma| k}$$

$$\hat{a}_{kl} = \frac{A_{kl} + k}{\sum_{r \in Q} A_{kr} + 3k}$$

$$k = \text{pseudo-count}$$

Remark: 3k, because there are 3 possible states (M, I, D) at each position

Example (cont'd)

ML estimates of emission and transition probabilities:

		0	1	2	3
Match emissions	E		1.0	-	-
	L		-	-	1.0
	V		-	1.0	-
	…		-	-	-
Insert emissions	E	-	-	0.6	-
	L	0.67	-	0.2	-
	V	0.33	-	0.2	-
	…	-	-	-	-
State transitions	M-M	0.4	0.75	0.67	1.0
	M-D	-	0.25	-	-
	M-I	0.6	-	0.33	-
	I-M	0.67	-	0.4	-
	I-D	0.33	-	0.2	-
	I-I	-	-	0.4	-
	D-M		-	-	1.0
	D-D		1.0	-	-
	D-I		-	1.0	-

With one pseudo-count:

		0	1	2	3
Match-emissions	E		5/24	1/23	1/21
	L		1/24	1/23	2/21
	V		1/24	4/23	1/21
	…		1/24	1/23	1/21
Insert emissions	E	1/23	1/20	4/25	1/20
	L	3/23	1/20	2/25	1/20
	V	2/23	1/20	2/25	1/20
	…	1/23	1/20	1/25	1/20
State-transitions	M-M	3/8	4/7	1/2	5/7
	M-D	1/8	2/7	1/6	1/7
	M-I	1/2	1/7	1/3	1/7
	I-M	1/2	1/3	3/8	1/3
	I-D	1/3	1/3	2/8	1/3
	I-I	1/6	1/3	3/8	1/3
	D-M		1/4	1/5	2/4
	D-D		1/2	1/5	1/4
	D-I		1/4	3/5	1/4

Variants

- Letter specific pseudo-count: requires sufficiently large training data
- Use of substitution matrices (heuristic approach):

$$\hat{b}_{za} = \frac{B_{za} + P_{za}}{\sum_{e \in \Sigma} B_{ze} + P_{ze}}$$

$$P_{za} = \sum_{e \in \Sigma} P(a \mid e) \frac{B_{za}}{\sum_{e \in \Sigma} B_{ze}}$$

$P(a \mid e)$: substitution prob.

ML estimate of emission probability

Variants for Parameter Estimation of Profile HMMs

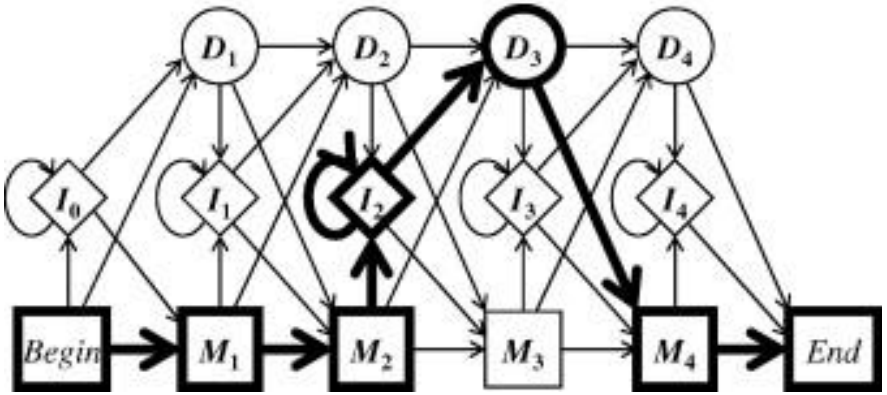
- **Option 1 (so far):** Start from multiple alignment (see previous example)
  - Use observed counts in each column to estimate emission and transition probabilities in a ML way
  - Problem: depends on alignment quality
- **Option 2**: Start from unaligned sequences
  - Use Baum-Welch algorithm
  - Problems: may get stuck in local optima, time consuming
- **Option 3**: Start with aligned sequences and use Baum-Welch
  - Advantage: starts from good initial solution ➔ higher chance to reach global optimum; reduces dependency on initial alignment

Profile HMM Alignment

- We can use a profile HMM to perform probabilistic sequence alignment.
  - How likely is it that a given sequence has been generated by the profile HMM sequence model?

  ➔ Forward algorithm
- Alignment of sequence against profile HMM: find most probable path of a sequence through HMM model → Viterbi‘s algorithm

Aligning a Sequence against a Profile HMM



- What is the most likely hidden path (sequence of M, I and D states) for ACGGT, allowing for possible insertions and deletions?

A C G G – T

M_1 M_2 I_2 I_2 D_3 M_4

- Decoding problem → Viterbi's algorithm

Profile HMM Alignment – aligning a sequence against a profile HMM

- Define $v^M_j(i)$ as the logarithmic likelihood score of the best path matching $y_1..y_i$ to profile HMM ending with y_i , which is emitted by the state M_j.

$$v_j^M(i) = \log P(y(1),..., y(i) \mid x(i) = M_j)$$

- $v^I_j(i)$ and $v^D_j(i)$ are defined similarly.

Profile HMM Alignment: Viterbi Algorithm Recurrences

$$v_j^M(i) = \log \frac{e_{M_j}(y_i)}{p(y_i)} + \max \begin{cases} v_{j-1}^M(i-1) + \log(a_{M_{j-1},M_j}) & \text{Match/mismatch} \\ v_{j-1}^I(i-1) + \log(a_{I_{j-1},M_j}) & \text{Gap end (profile)} \\ v_{j-1}^D(i-1) + \log(a_{D_{j-1},M_j}) & \text{Gap end (sequence)} \end{cases}$$

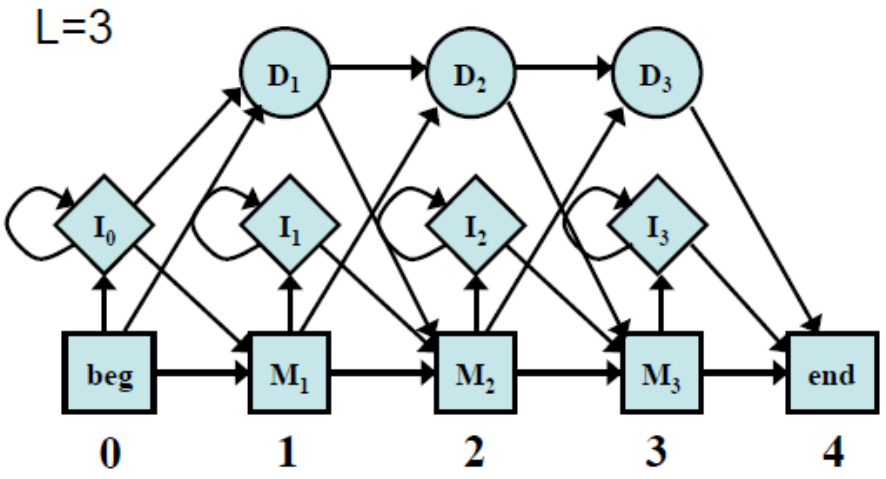
$$v_j^I(i) = \log \frac{e_{I_j}(y_i)}{p(y_i)} + \max \begin{cases} v_j^M(i-1) + \log(a_{M_j,I_j}) & \text{Gap initialization (profile)} \\ v_j^I(i-1) + \log(a_{I_j,I_j}) & \text{Gap extension (profile)} \\ v_j^D(i-1) + \log(a_{D_j,I_j}) & \text{Gap initialization (profile)} \end{cases}$$

$$v_j^D(i) = \max \begin{cases} v_{j-1}^M(i) + \log(a_{M_{j-1},D_j}) & \text{Gap initialization (sequence)} \\ v_{j-1}^I(i) + \log(a_{I_{j-1},D_j}) & \text{Gap initialization (sequence)} \\ v_{j-1}^D(i) + \log(a_{D_{j-1},D_j}) & \text{Gap extension (sequence)} \end{cases}$$

Comparison to Dynamic Programming based Profile Alignment

- Advantage: position and sequence / profile specific gap opening and extension penalties
- No need to pre-specify these penalties
- They are parameters of the HMM and learned from training sequences

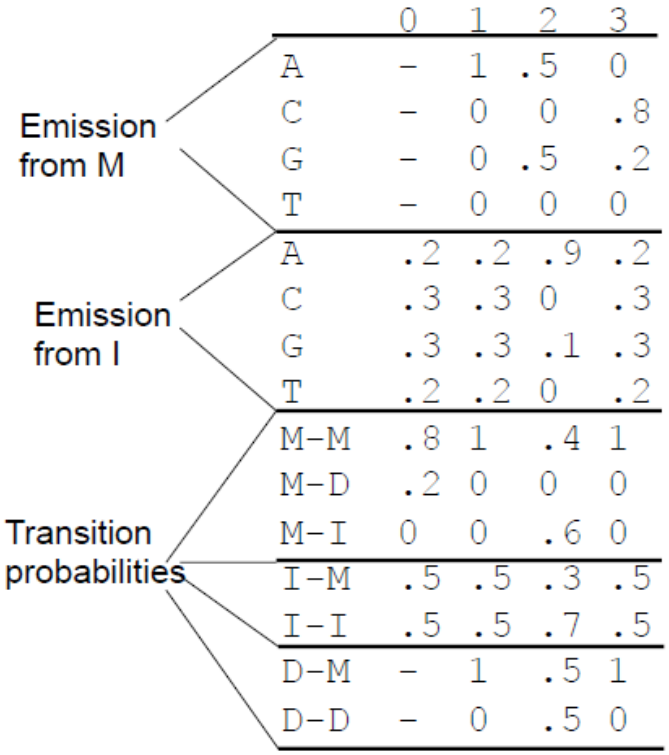
Example: Profile HMM and Query



Query: AGG (N=3)

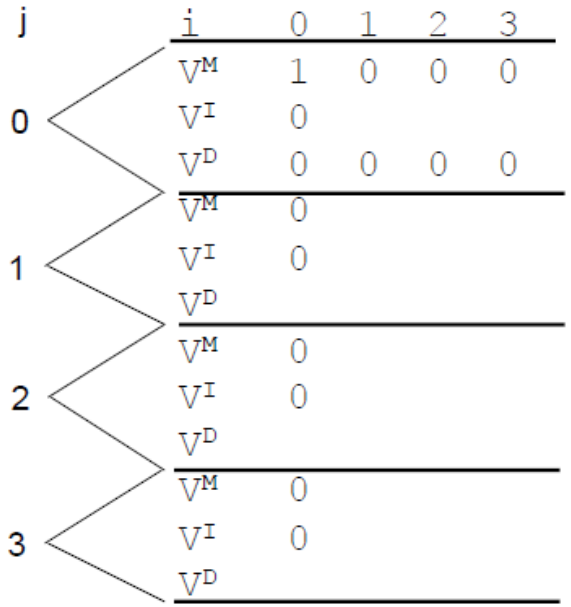
		0	1	2	3
Emission from M	A	-	1	.5	0
	C	-	0	0	.8
	G	-	0	.5	.2
	T	-	0	0	0
Emission from I	A		.2	.2	.9	.2
	C		.3	.3	0	.3
	G		.3	.3	.1	.3
	T		.2	.2	0	.2
Transition probabilities	M-M	.8	1	.4	1
	M-D	.2	0	0	0
	M-I	0	0	.6	0
	I-M	.5	.5	.3	.5
	I-I	.5	.5	.7	.5
	D-M	-	1	.5	1
	D-D	-	0	.5	0

Example: Initialization



	0	1	2	3
A	-	1	.5	0
C	-	0	0	.8
G	-	0	.5	.2
T	-	0	0	0
A	.2	.2	.9	.2
C	.3	.3	0	.3
G	.3	.3	.1	.3
T	.2	.2	0	.2
M-M	.8	1	.4	1
M-D	.2	0	0	0
M-I	0	0	.6	0
I-M	.5	.5	.3	.5
I-I	.5	.5	.7	.5
D-M	-	1	.5	1
D-D	-	0	.5	0

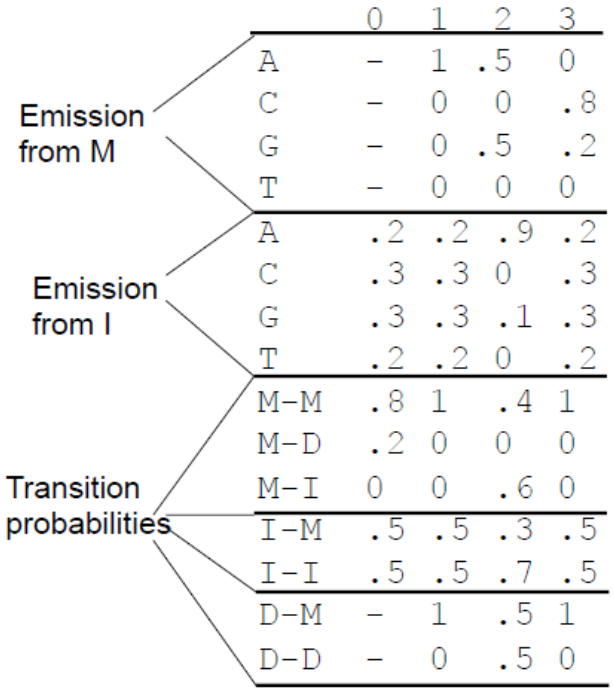
Initialization



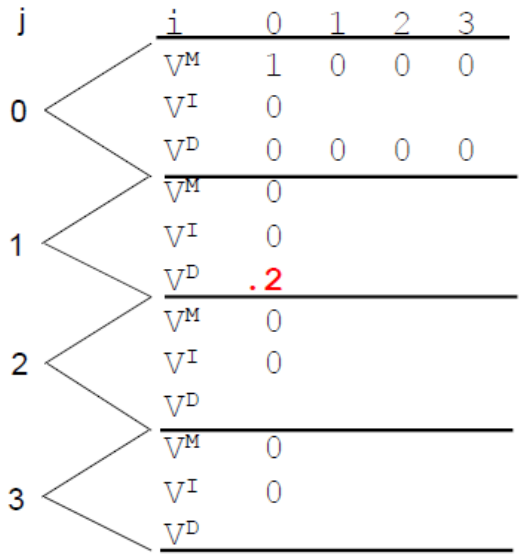
i	0	1	2	3
V^M	1	0	0	0
V^I	0			
V^D	0	0	0	0
V^M	0			
V^I	0			
V^D				
V^M	0			
V^I	0			
V^D				
V^M	0			
V^I	0			
V^D				

Query: X=AGG (N=3)

Example: Step 1



	0	1	2	3
A	-	1	.5	0
C	-	0	0	.8
G	-	0	.5	.2
T	-	0	0	0
A	.2	.2	.9	.2
C	.3	.3	0	.3
G	.3	.3	.1	.3
T	.2	.2	0	.2
M-M	.8	1	.4	1
M-D	.2	0	0	0
M-I	0	0	.6	0
I-M	.5	.5	.3	.5
I-I	.5	.5	.7	.5
D-M	-	1	.5	1
D-D	-	0	.5	0

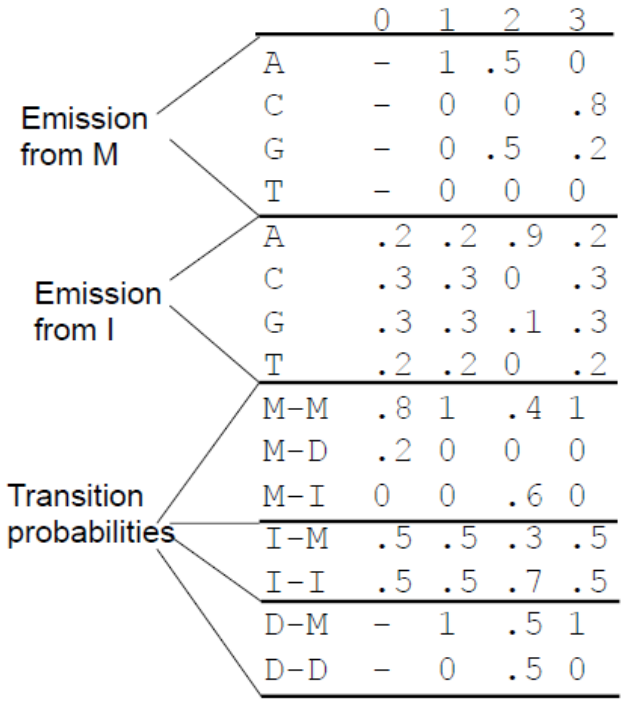


i	0	1	2	3
V^M	1	0	0	0
V^I	0			
V^D	0	0	0	0
V^M	0			
V^I	0			
V^D	.2			
V^M	0			
V^I	0			
V^D				
V^M	0			
V^I	0			
V^D				

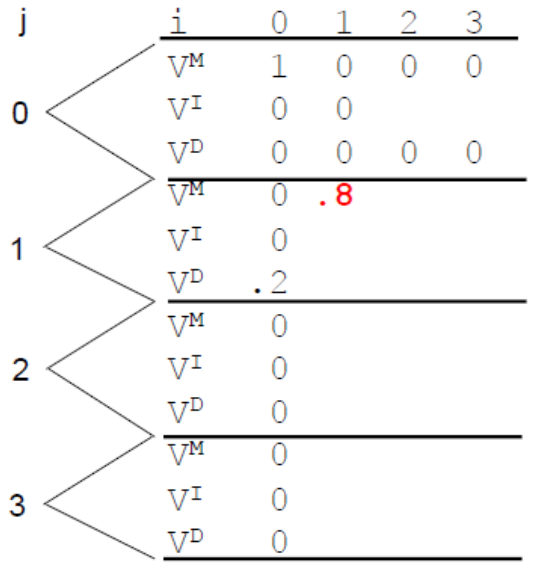
Query: X=AGG (N=3)

$$V_1^D(0) = \max\left(V_0^M(0)a_{M_0D_1}, V_0^D(0)a_{D_0D_1}\right)$$
$$= \max(0.2, 0) = 0.2$$

Example: Step 2



	0	1	2	3
A	-	1	.5	0
C	-	0	0	.8
G	-	0	.5	.2
T	-	0	0	0
A	.2	.2	.9	.2
C	.3	.3	0	.3
G	.3	.3	.1	.3
T	.2	.2	0	.2
M-M	.8	1	.4	1
M-D	.2	0	0	0
M-I	0	0	.6	0
I-M	.5	.5	.3	.5
I-I	.5	.5	.7	.5
D-M	-	1	.5	1
D-D	-	0	.5	0

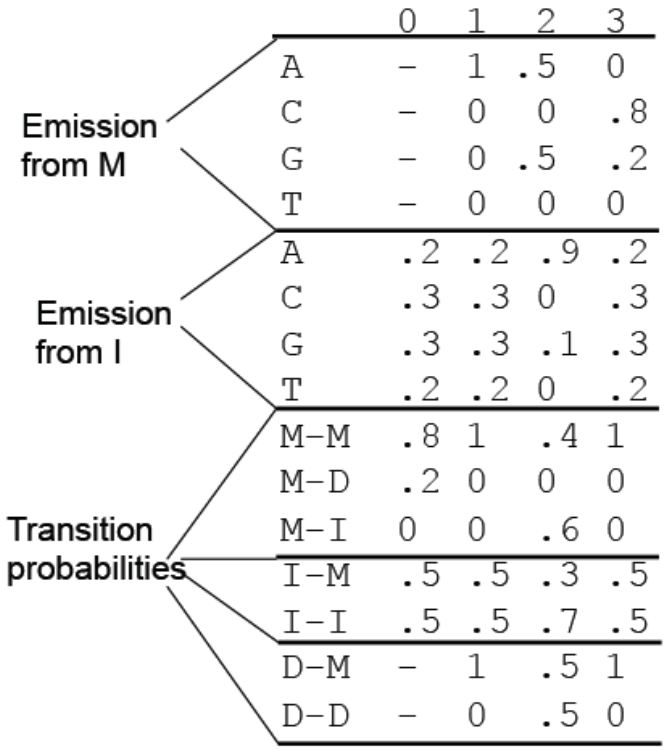


i	0	1	2	3
V^M	1	0	0	0
V^I	0	0		
V^D	0	0	0	0
V^M	0	.8		
V^I	0			
V^D	.2			
V^M	0			
V^I	0			
V^D	0			
V^M	0			
V^I	0			
V^D	0			

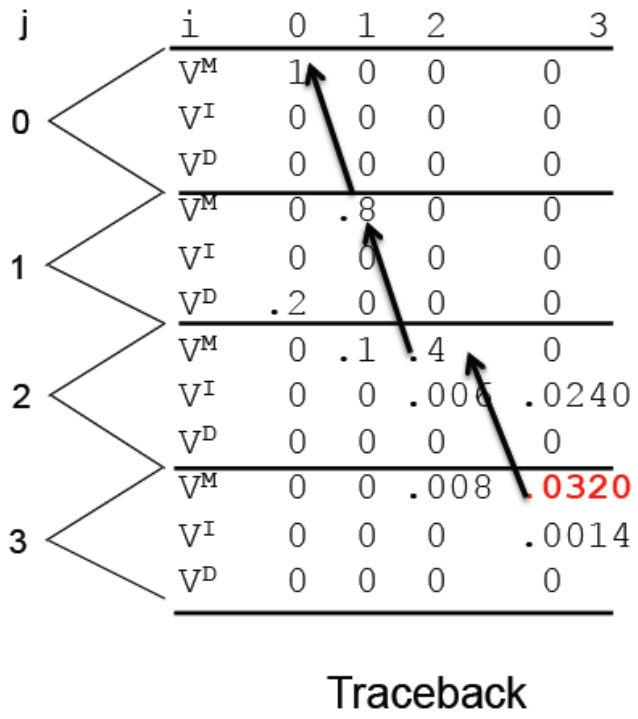
Query: X=AGG (N=3)

$$V_1^M(1) = e_{M_1}(x_1) \cdot \max\left(V_0^M(0)a_{M_0M_1}, V_0^I(0)a_{I_0M_1}, V_0^D(0)a_{D_0M_1}\right)$$
$$= 1 \times \max(1 \times 0.8, 0 \times 0.5, 0 \times 1) = 0.8$$

Example: Final iteration, traceback



	0	1	2	3
A	–	1	.5	0
C	–	0	0	.8
G	–	0	.5	.2
T	–	0	0	0
A	.2	.2	.9	.2
C	.3	.3	0	.3
G	.3	.3	.1	.3
T	.2	.2	0	.2
M–M	.8	1	.4	1
M–D	.2	0	0	0
M–I	0	0	.6	0
I–M	.5	.5	.3	.5
I–I	.5	.5	.7	.5
D–M	–	1	.5	1
D–D	–	0	.5	0



i	0	1	2	3
V^M	1	0	0	0
V^I	0	0	0	0
V^D	0	0	0	0
V^M	0	.8	0	0
V^I	0	0	0	0
V^D	.2	0	0	0
V^M	0	.1	.4	0
V^I	0	0	.006	.0240
V^D	0	0	0	0
V^M	0	0	.008	**.0320**
V^I	0	0	0	.0014
V^D	0	0	0	0

Query: X=AGG (N=3)

Extensions of Profile HMMs and Popular Implementations

- Profile HMMs in the shown variant represent global alignments
- Extensions to local alignments are possible
  - Match states do not span the entire profile, but only a local region

- Popular Profile HMM implementation: HMMER (Sean Eddy, 1996)

Application Example: PFAM

- Pfam describes ***protein domains*** (http://pfam.xfam.org)
- Each protein domain family in Pfam has:
  - S*eed alignment*: manually verified multiple alignment of a representative set of sequences.
  - profile *HMM* built from the seed alignment for further database searches.
  - *Full alignment* generated automatically from the HMM
- Details about HMM model generation (HMMER) provided
- All models can be downloaded

- Biological background information for each protein domain available

What you should know and being able to apply

- What is the purpose of a HMM and how is it defined?
  - Being able to define / draw a HMM for a given task
- What is the purpose of Viterbi‘s algorithm and the forward-backward algorithm? How do these algorithms roughly work?
- How can HMM parameters be estimated?
- What is a profile HMM and for which purposes is it used? How is such a HMM trained and applied?

Acknowledgements

- A few slides were adopted and modified from www.bioalgorithms.info. We thank the authors for their great work.
- Further sources:
  - N. Jones, P. Pevzner, An Introduction to Bioinformatics Algorithms, MIT Press, 2004
  - Durbin, Eddy, Krogh, Mitchson, Biological Sequence Analysis, Cambridge University Press, 2002