

Hidden Markov models

Torgeir R. Hvidsten

Markov models

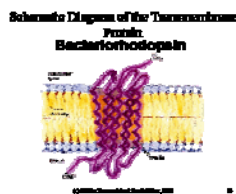
- Application: Classification
- Task: Model a set of sequences of unequal length probabilistically.

ATGCT ← Unequal length. PSSM won't do it!
 GTGCTAC
 GCTAC
 ATGCTGG

Sequences from N letter alphabet, could be nucleotides {A,G,C,T}, amino acids {V,P,A,L, ...}, codons {AUG, GAC, ...} et cetera.

0th order solution

- Assign a probability to each letter (e.g. its frequency). N parameters needed
- Assume independent observations:
 - $P(A|GCTG) = P(A)$
- Problem:
 - $P(AGCT) = P(A)P(G)P(C)P(T) = P(GTCA)$, i.e. **permutation invariant**
 - Order carries a lot of information in biology, e.g.
 - GC-islands
 - Hydrophobic core vs hydrophilic surface of globular proteins
 - Transmembrane proteins
 - et cetera

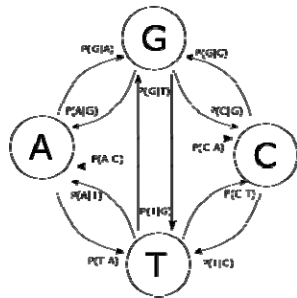


1st order solution

Assume each letter depends on the preceding.

- $P(AGTTCGT) = P(A)P(G|A)P(T|G)P(C|T)P(G|C)P(T|G)$
- No longer permutation invariant.
- $N \times N$ parameters (+ N for start probabilities)
- Note: Higher order models often intractable
 - k th order model requires N^k parameters

Graphical representation



T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

5

Summary: Marko models

- 1st order Markov models incorporates 1st order dependencies (i.e. $P(X|Y)$)
- Can be represented as a weighted, directed graph.
- Application:
 - Comparing two sequence families:
 - Choose architecture(s)
 - Estimate $P(X|Y)$ for all X, Y in alphabet for each of the sequence families
 - For classification, calculate the probability of each of the models generating the sequence
 - Practical note: For long sequences numerical overflow is a problem. Solution: work with logarithms.

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

6

Hidden Markov models

- Applications: Sequence alignment, gene detection et cetera
- Originated in speech recognition

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

7

CG-islands

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

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

8

Why CG-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 *CG* islands
- So, finding the *CG* islands in a genome is an important problem (gene finding)

CG-islands and the “Fair Bet Casino”

- The *CG* 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: **Head** or **Tail**
- The **Fair** coin will produce **Heads** and **Tails** with the same probability $\frac{1}{2}$
- The **Biased** coin will produce **Heads** with prob. $\frac{3}{4}$

The “Fair Bet Casino” (cont'd)

Thus, we define the probabilities:

- $P(H|F) = P(T|F) = \frac{1}{2}$
- $P(H|B) = \frac{3}{4}$, $P(T|B) = \frac{1}{4}$
- The crooked dealer changes between Fair and Biased coins with probability 0.1

The Fair Bet Casino problem

- **Input:** A sequence $\mathbf{x} = x_1 x_2 x_3 \dots x_n$ of coin tosses (either *H* or *T*) made by two possible coins (*F* or *B*)
- **Output:** A sequence $\boldsymbol{\pi} = \pi_1 \pi_2 \pi_3 \dots \pi_n$, with each π_i being either *F* or *B* indicating that x_i is the result of tossing the Fair or Biased coin, respectively
- **Problem:** Any observed outcome of coin tosses could have been generated by any sequence of states!

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

Suppose first that the dealer never changes coins

Some definitions:

- **P(x|fair coin)**: prob. of the dealer using the *F* coin and generating the outcome **x**
- **P(x|biased coin)**: prob. of the dealer using the *B* coin and generating outcome **x**

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

➤ $P(\mathbf{x} | \text{fair coin}) = P(x_1 \dots x_n | \text{fair coin}) =$

$$\prod_{i=1, n} P(x_i | \text{fair coin}) = (1/2)^n$$

➤ $P(\mathbf{x} | \text{biased coin}) = P(x_1 \dots x_n | \text{biased coin}) =$

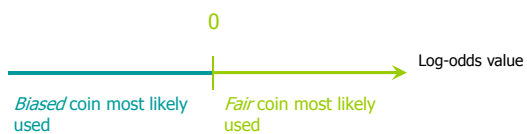
$$\prod_{i=1, n} P(x_i | \text{biased coin}) = (3/4)^k (1/4)^{n-k} = 3^k / 4^n$$

where *k* is the number of **H**eads in **x** (and *n-k* is the number of **T**ails)

Log-odds ratios

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

$$\log_2(P(\mathbf{x} | \text{fair coin}) / P(\mathbf{x} | \text{biased coin}))$$



Computing log-odds ratio in sliding windows

$$x_1 x_2 \boxed{x_3 x_4 x_5 x_6 x_7} x_8 \dots x_n$$

Consider a *sliding window* of the outcome sequence, find the log-odds for this short window

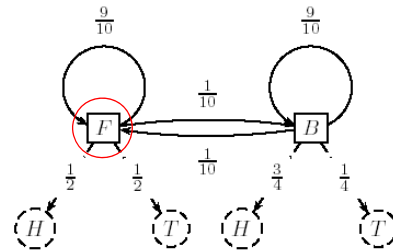
Disadvantages:

- the length of CG-island is not known in advance
- different windows may classify the same position differently

Hidden Markov Model (HMM)

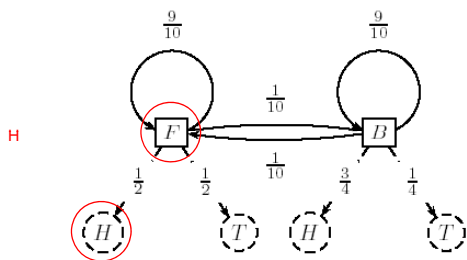
- Can be viewed as an abstract machine with k *hidden* states that emits symbols from an alphabet Σ
- Each state has its own probability distribution, and the machine switches between states according to this probability distribution
- While in a certain state, the machine makes 2 decisions:
 - What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

HMM for Fair Bet Casino



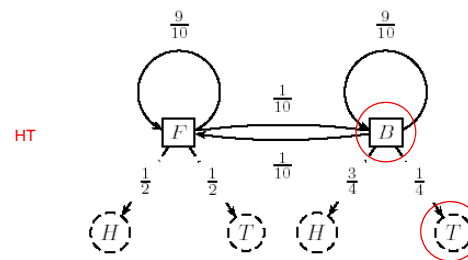
HMM model for the *Fair Bet Casino* Problem

HMM for Fair Bet Casino (cont'd)



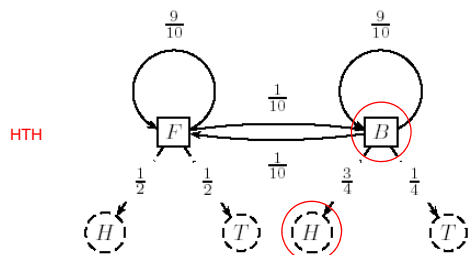
HMM model for the *Fair Bet Casino* Problem

HMM for Fair Bet Casino (cont'd)



HMM model for the *Fair Bet Casino* Problem

HMM for Fair Bet Casino (cont'd)



HMM model for the *Fair Bet Casino* Problem

Why “Hidden”?

- Observers can see the emitted symbols of an HMM but have **no ability to know which state the HMM is currently in**
- Thus, the goal is to infer the most likely hidden states of an HMM based on the given sequence of emitted symbols.

HMM parameters

- Σ : set of emission characters
 $\Sigma = \{H, T\}$ for coin tossing
 $\Sigma = \{A, C, G, T\}$ for the CG-island problem
- Q : set of hidden states, each emitting symbols from Σ
 $Q = \{F, B\}$ for coin tossing
 $Q = \{\text{CG-island, not CG-island}\}$ for the CG-island problem

HMM Parameters (cont'd)

- $A = (a_{kl})$: a $|Q| \times |Q|$ matrix of probability of changing from state k to state l
 - *transition probabilities*
- $E = (e_k(b))$: a $|Q| \times |\Sigma|$ matrix of probability of emitting symbol b while being in state k
 - *emission probabilities*

HMM for Fair Bet Casino

The *Fair Bet Casino* in HMM terms:

$\Sigma = \{0, 1\}$ – 0 for **T**ails and 1 for **H**eads

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

Transition Probabilities \mathcal{A} Emission Probabilities E

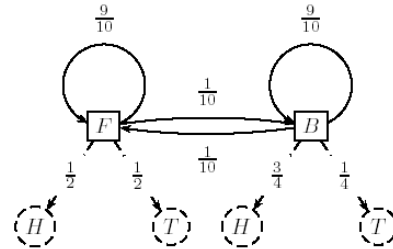
	Fair	Biased
Fair	$a_{FF} = 0.9$	$a_{FB} = 0.1$
Biased	$a_{BF} = 0.1$	$a_{BB} = 0.9$

	Tails(0)	Heads(1)
Fair	$e_F(0) = 1/2$	$e_F(1) = 1/2$
Biased	$e_B(0) = 3/4$	$e_B(1) = 1/4$

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

25

HMM for Fair Bet Casino (cont'd)



HMM model for the *Fair Bet Casino* Problem

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

26

Hidden Paths

➤ A *path* $\pi = \pi_1 \dots \pi_n$ in the HMM is defined as a sequence of states

➤ Consider path $\pi = \text{FFFBBBBBFFF}$ and sequence $\mathbf{x} = 01011101001$

		Probability that x_j was emitted from state π_j										
\mathbf{x}		0	1	0	1	1	1	0	1	0	0	1
π	\neq	F	F	F	B	B	B	B	B	F	F	F
$P(x_j \pi_j)$		1/2	1/2	1/2	3/4	3/4	3/4	1/4	3/4	1/2	1/2	1/2
$P(\pi_{i-1} \rightarrow \pi_i)$		1/2	9/10	9/10	1/10	9/10	9/10	9/10	9/10	1/10	9/10	9/10
		Transition probability from state π_{i-1} to state π_i										

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

27

$P(\mathbf{x}, \pi)$ Calculation

$P(\mathbf{x}, \pi) = P(\mathbf{x} | \pi) P(\pi)$: Probability that sequence \mathbf{x} was generated by the path π :

$$\begin{aligned}
 P(\mathbf{x}, \pi) &= P(\pi_0 \rightarrow \pi_1) \cdot \prod_{i=1}^n P(x_i | \pi_i) \cdot P(\pi_i \rightarrow \pi_{i+1}) \\
 &= a_{\pi_0, \pi_1} \cdot \prod_{i=1}^n e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_{i+1}} \\
 &= \prod_{i=0}^n e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i, \pi_{i+1}}
 \end{aligned}$$

where π_0 and π_{n+1} are fictitious initial and terminal states *begin* and *end*

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

28

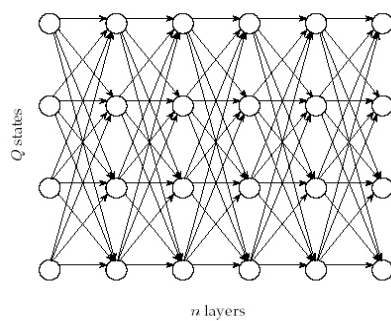
Decoding problem

- **Goal:** Find an optimal hidden path of states given observations
- **Input:** Sequence of observations $\mathbf{x} = x_1 \dots x_n$ generated by an HMM: $M(\Sigma, Q, A, E)$
- **Output:** A path that maximizes $P(\mathbf{x}, \pi)$ over all possible paths π

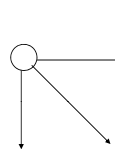
Building an edit graph for the Decoding problem

- Andrew Viterbi used the Manhattan grid model to solve the *Decoding Problem*
- Every choice of $\pi = \pi_1 \dots \pi_n$ corresponds to a path in the graph
- The only valid direction in the graph is *eastward*
- This graph has $|Q|^2(n-1)$ edges

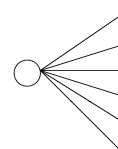
Edit graph for Decoding problem



Decoding problem vs. Alignment problem



Valid directions in the *alignment problem*



Valid directions in the *decoding problem*.

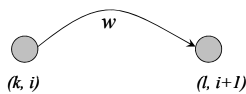
Decoding problem: reformulated

- The *Decoding problem* is reduced to finding a longest path in the *directed acyclic graph (DAG)*
- **Notes:** the length of the path is defined as the *product* of its edges' weights, not the *sum*.

Decoding problem (cont'd)

- Every path in the graph has the probability $P(\mathbf{x}, \boldsymbol{\pi})$
- The Viterbi algorithm finds the path that maximizes $P(\mathbf{x}, \boldsymbol{\pi})$ among all possible paths
- The Viterbi algorithm runs in $O(n|Q|^2)$ time

Weights of edges

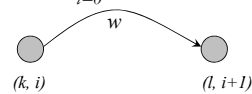


The weight w is given by:

???

Weights of edges

$$P(\mathbf{x}, \boldsymbol{\pi}) = \prod_{i=0}^n e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i, \pi_{i+1}}$$

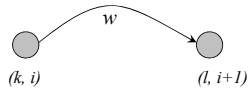


The weight w is given by:

??

Weights of edges

$$i\text{-th term} = e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i, \pi_{i+1}}$$

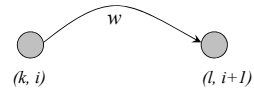


The weight w is given by:

?

Weights of edges

$$i\text{-th term} = e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i, \pi_{i+1}} = e_l(x_{i+1}) \cdot a_{kl} \text{ for } \pi_i = k, \pi_{i+1} = l$$



The weight $w = e_l(x_{i+1}) \cdot a_{kl}$

Decoding problem and Dynamic programming

Define $s_{k,i}$ as the probability of emitting the prefix $x_1 \dots x_i$ and reaching the state k

$$s_{l,i+1} = \max_{k \in Q} \{s_{k,i} \cdot \text{weight of edge between } (k,i) \text{ and } (l,i+1)\} =$$

$$\max_{k \in Q} \{s_{k,i} \cdot a_{kl} \cdot e_l(x_{i+1})\} =$$

$$e_l(x_{i+1}) \cdot \max_{k \in Q} \{s_{k,i} \cdot a_{kl}\}$$

Decoding problem (cont'd)

► Initialization:

$$- s_{begin,0} = 1$$

$$- s_{k,0} = 0 \text{ for } k \neq \text{begin.}$$

► Let π^* be the optimal path. Then,

$$P(\mathbf{x}, \pi^*) = \max_{k \in Q} \{s_{k,n} \cdot a_{k,end}\}$$

Viterbi algorithm

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

$$s_{l,i+1} = \log e_l(x_{i+1}) + \max_{k \in Q} \{s_{k,i} + \log(a_{kl})\}$$

Remember: $\log(ab) = \log a + \log b$

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

Forward algorithm

- Define $f_{k,i}$ (*forward probability*) as the probability of emitting the prefix x_1, \dots, x_i and reaching the state $\pi = k$
- The recurrence for the forward algorithm:

$$f_{k,i} = e_k(x_i) \cdot \sum_{l \in Q} f_{l,i-1} \cdot a_{lk}$$

Remember the Viterbi algorithm:

$$s_{l,i+1} = e_l(x_{i+1}) \cdot \max_{k \in Q} \{s_{k,i} + \log(a_{kl})\}$$

Backward algorithm

- However, *forward probability* is not the only factor affecting $P(\pi_i = k|x)$

- The sequence of transitions and emissions that the HMM undergoes between π_{i+1} and π_n also affect $P(\pi_i = k|x)$

forward x_i backward

Backward algorithm (cont'd)

- Define *backward probability* $b_{k,i}$ as the probability of being in state $\pi_i = k$ and emitting the *suffix* $x_{i+1} \dots x_n$
- The recurrence for the *backward algorithm*:

$$b_{k,i} = \sum_{l \in Q} e_l(x_{i+1}) \cdot b_{l,i+1} \cdot a_{kl}$$

Backward-forward algorithm

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

$$P(\pi_i = k | x) = \frac{P(x, \pi_i = k)}{P(x)} = \frac{f_{ki} \cdot b_{ki}}{P(x)}$$

where:

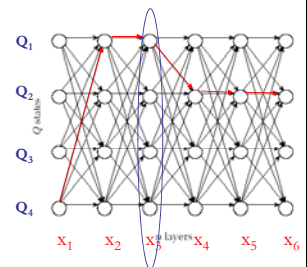
$P(x, \pi_i = k) = \sum_{\pi} P(x, \pi)$ with $\pi_i = k$: the sum of probabilities of all paths with $\pi_i = k$
 $P(x) = \sum_{\pi} P(x, \pi)$: the sum of probabilities over all paths

Summary

- HMM defined by
 - Σ : set of emission characters
 - Q : set of hidden states
 - Transition probabilities
 - Emission probabilities
- Finding most probable path for a given sequence can be done with the Viterbi algorithm
- The total probability of the sequence (for matching) calculated with the forward algorithm
- Probability of being in a state at a given time requires the backward algorithm in addition

Summary

- Every *layer* i emit one symbol x_i
- Every *path* from layer 1 to layer n has probability $P(x, \pi)$
- Note that the path tells us which hidden state in layer i that emitted x_i



Summary

- Probability of sequence and path:
 - $P(\mathbf{x}, \pi) = P(\mathbf{x} | \pi) P(\pi) = \prod_i e_{\pi_{i+1}(x_{i+1})} \cdot a_{\pi_i, \pi_{i+1}}$
- Viterbi gives the path π^* that maximizes $P(\mathbf{x}, \pi)$:
 - $s_{i+1} = e_i(x_{i+1}) \cdot \max_{k \in Q} \{s_{ik} \cdot a_{ik}\}$
- Forward algorithm sums the probability of \mathbf{x} over all paths:
 - $\sum_{\pi} P(\mathbf{x}, \pi) = \sum_{\pi} P(\mathbf{x} | \pi) P(\pi) = P(\mathbf{x})$
 - $f_{k,i} = e_k(x_i) \cdot \sum_l f_{l,i-1} \cdot a_{lk}$
- Forward+backwards sums the probability of \mathbf{x} over all paths with $\pi_i = k$
 - $\sum_{\pi \text{ with } \pi_i = k} P(\mathbf{x}, \pi)$

Summary

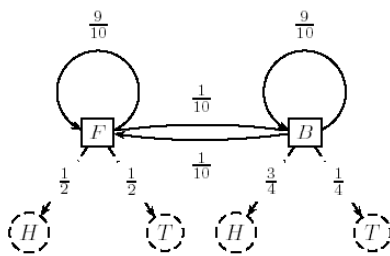
- A *path* $\pi = \pi_1 \dots \pi_n$ in the HMM is defined as a sequence of states
- Consider path $\pi = \text{FFFBBBBBFFF}$ and sequence $\mathbf{x} = 01011101001$

Probability that x_i was emitted from state π_i

x	0	1	0	1	1	1	0	1	0	0	1
π	F	F	F	B	B	B	B	B	F	F	F
$P(x_i \pi_i)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$P(\pi_{i-1} \rightarrow \pi_i)$	$\frac{1}{2}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$

Transition probability from state π_{i-1} to state π_i

Summary



HMM model for the *Fair Bet Casino Problem*

Finding distant members of a protein family

- A distant cousin of functionally related sequences in a protein family may have weak pairwise similarities with each member of the family and thus fail significance test using e.g. BLAST
- 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 profile

Profile representation of protein families

Aligned DNA sequences can be represented by a $4 \cdot n$ profile matrix reflecting the frequencies of nucleotides in every aligned position.

A	.72	.14	0	0	.72	.72	0	0
T	.14	.72	0	0	0	.14	.14	.86
G	.14	.14	.86	.44	0	.14	0	0
C	0	0	.14	.56	.28	0	.86	.14

Protein family can be represented by a $20 \cdot n$ profile representing frequencies of amino acids.

HMMs

- HMMs can also be used for aligning a sequence against a protein family
- Conserved positions in the family corresponds to n sequentially linked *match* states M_1, \dots, M_n in the **profile HMM**
- HMMs handle gaps better than profiles do

Building a profile HMM

- Multiple alignment is used to construct the HMM model
- Assign each column to a *Match* state in HMM. Add *Insertion* and *Deletion* state
- Estimate the emission probabilities according to amino acid counts in columns
- Estimate the transition probabilities between *Match*, *Deletion* and *Insertion* states

```

VTLSCTGSSSNIGAG-NHVKWYQQLPG
VTLSCTGTSSNIGS--ITVNWYQQLPG
LRLSCSSSGPIFSS--YAMYWVRQA--
LSLTCVSG-SFDD--YSTWVRQP--
PEVTCVVVD-SHEDPQVKFNWYVDG--
ATLVCLISDFYPGA--VTVAWKADS--
AALGCLVKD-FPER--VTVSWNSG--
VSLTCLVKGFYPSD-LAVEWESNG--
    
```

Match state 3:

$$c_{M3}(L) = 5/8$$

$$c_{M3}(I) = 3/8$$

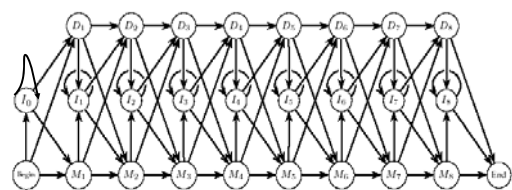
Modeled as insertions

Transitions from match state 9 to 10

$$a_{M9,M10} = 5/8$$

$$a_{3B,D10} = 3/8$$

Profile HMM



A profile HMM

Note: penalties in HMMs

Different penalties for opening a gap and extending the gap is naturally implemented in HMM

- $\log(a_{MI}) + \log(a_{IM}) = \text{gap initiation penalty}$
- $\log(a_{II}) = \text{gap extension penalty}$

Emission probabilities for insertions

Probability of emitting a symbol a at an insertion state I_j :

$$e_{I_j}(a) = p(a)$$

where $p(a)$ is the frequency of the occurrence of the symbol a in all the sequences.

Profile HMM alignment

➤ Define $v_j^M(i)$ as the logarithmic likelihood score of the best path for matching $x_1 \dots x_i$ to a profile HMM ending with x_i emitted by the state M_j

➤ $v_j^I(i)$ and $v_j^D(i)$ are defined similarly

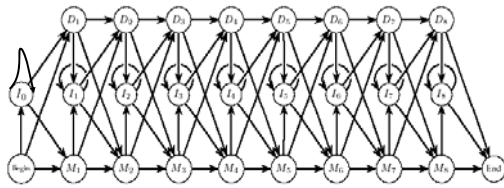
Profile HMM alignment: Dynamic programming

$$v_j^M(i) = \log(e_{M_j}(x_i)) + \max \begin{cases} v_{j-1}^M(i-1) + \log(a_{M_{j-1}M_j}) \\ v_{j-1}^I(i-1) + \log(a_{I_{j-1}M_j}) \\ v_{j-1}^D(i-1) + \log(a_{D_{j-1}M_j}) \end{cases}$$

$$v_j^I(i) = \log(e_{I_j}(x_i)) + \max \begin{cases} v_j^M(i-1) + \log(a_{M_j I_j}) \\ v_j^I(i-1) + \log(a_{I_j I_j}) \\ v_j^D(i-1) + \log(a_{D_j I_j}) \end{cases}$$

$$v_j^D(i) = \max \begin{cases} v_j^M(i) + \log(a_{M_j D_j}) \\ v_j^I(i) + \log(a_{I_j D_j}) \\ v_j^D(i) + \log(a_{D_j D_j}) \end{cases}$$

Profile HMM



A profile HMM

Making a collection of HMM for protein families

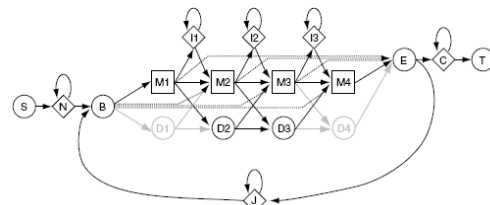
- Use Blast to separate a protein database into families of related proteins
- Construct a multiple alignment for each protein family
- Construct a profile HMM model and optimize the parameters of the model (transition and emission probabilities)
- Align the target sequence against each HMM to find the best fit between a target sequence and an HMM

Pfam

- Pfam describes **protein domains**
- Each protein domain family in Pfam has:
 - *Seed alignment*: manually verified multiple alignment of a representative set of sequences
 - *HMM* built from the seed alignment for further database searches
 - *Full alignment* generated automatically from the HMM
- The distinction between seed and full alignments facilitates Pfam updates
 - Seed alignments are stable resources
 - Full alignments can be updated with newly found amino acid sequences

Pfam

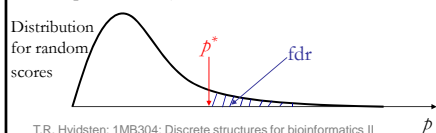
Pfam uses a tool called HMMER with the following architecture:



Scoring matches

Given a protein sequence \mathbf{x} and an HMM, what is a significant score?

- The score from the forward algorithm: $p^* = \log p(\mathbf{x})$
- Generate 1000 random sequences and score them:
 $P_{rand\ 1}, P_{rand\ 2}, \dots, P_{rand\ 1000}$
- Fit a distribution to the random scores and calculate the false discover rate (fdr)
- $E\text{-score} = fdr \cdot \text{Size of query database}$ (the expected number of false positive hits)



T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

65

HMM parameter estimation

- So far, we have (mostly) assumed that the transition and emission probabilities are known
- However, in most HMM applications, the probabilities are not known. It's very hard to estimate the probabilities.

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

66

HMM parameter estimation problem

Given

- HMM with **states** and **alphabet** (emission characters)
- Independent **training sequences** x^1, \dots, x^m

Find HMM parameters Θ (that is, $a_{kb}, e_k(b)$) that **maximize**

$$P(x^1, \dots, x^m \mid \Theta),$$

the joint probability of the training sequences.

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

67

Maximize the likelihood

$P(x^1, \dots, x^m \mid \Theta)$ as a function of Θ is called the **likelihood** of the model

The training sequences are assumed independent, therefore

$$P(x^1, \dots, x^m \mid \Theta) = \prod_i P(x^i \mid \Theta)$$

The parameter estimation problem seeks Θ that realizes

$$\max_{\Theta} \prod_i P(x^i \mid \Theta)$$

In practice the **log likelihood** is computed to avoid underflow errors

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

68

Two situations

Known paths for training sequences

- CG islands are marked on training sequences
- One evening the casino dealer allows us to see when he changes dice
- A multiple alignment of the protein family is given

Unknown paths

- CG islands are not marked
- Do not see when the casino dealer changes dice
- A multiple alignment of the protein family is *not* given

Known paths

A_{kl} = # of transitions from state k to state l in the training sequences

$E_k(b)$ = # of times b is emitted from state k in the training sequences

Compute a_{kl} and $e_k(b)$ as maximum likelihood estimators:

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

$$e_k(b) = E_k(b) / \sum_{b' \in \Sigma} E_k(b')$$

Pseudocounts

- Some state k may not appear in any of the training sequences. This means $A_{kl} = 0$ for every state l and a_{kl} cannot be computed with the given equation
- To avoid this **overfitting** use predetermined **pseudocounts** r_{kl} and $r_k(b)$.

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

$$E_k(b) = \# \text{ of emissions of } b \text{ from } k + r_k(b)$$

- The pseudocounts reflect our prior biases about the probability values

Unknown paths: Viterbi training

Idea: use Viterbi decoding to compute **the most probable paths** for training sequences \mathbf{x}

- **Start** with some guess for initial parameters
- **Iterate** :
 - Compute the most probable path π^* for each \mathbf{x} using the current parameters (Viterbi algorithm)
 - Stop if no change in π^*
 - Determine A_{kl} and $E_k(b)$ using the computed paths
 - Compute new parameters a_{kl} and $e_k(b)$ using the same formulas as before

Viterbi training analysis

- The algorithm **converges precisely**
 - There are a finite number of possible paths
 - New parameters are uniquely determined by the current π^*
 - There may be several paths for \mathbf{x} with the same probability, hence to decide convergence one must compare the new π^* with all previous paths having the highest probability
- Does **not maximize the likelihood** $\Pi_i P(\mathbf{x}^i | \theta)$ but the contribution to the likelihood of the most probable path $\Pi_i P(\mathbf{x}^i | \theta, \pi^*)$
- In general performs less well than Baum-Welch

Unknown paths: Baum-Welch

Idea:

1. Guess initial values for parameters.
art and experience, not science
2. Estimate new (better) values for parameters.
how ?
3. Repeat until stopping criteria is met.
what criteria ?

Better values for parameters

Would need the A_{kl} and $E_k(b)$ values but cannot count (the path is unknown) and do not want to use a most probable path

For all states k, l , symbol b and training sequence x

Compute A_{kl} and $E_k(b)$ as **expected values**, given the current parameters

Notation

For any sequence of characters \mathbf{x} emitted along some **unknown path** π , denote by $\pi_i = k$ the assumption that the state at position i (in which x_i is emitted) is k

Probabilistic setting for $A_{k,l}$

Given x^1, \dots, x^m consider a **discrete probability space** with **elementary events**

$$\varepsilon_{k,l} = \text{"}k \rightarrow l \text{ is taken in } x^1, \dots, x^m \text{"}$$

For each x in $\{x^1, \dots, x^m\}$ and each position i in x let $Y_{x,i}$ be a **random variable** defined by

$$Y_{x,i}(\varepsilon_{k,l}) = \begin{cases} 1, & \text{if } \pi_i = k \text{ and } \pi_{i+1} = l \\ 0, & \text{otherwise} \end{cases}$$

Define $Y = \sum_x \sum_i Y_{x,i}$ random variable that counts # of times the event $\varepsilon_{k,l}$ happens in x^1, \dots, x^m .

The meaning of $A_{k,l}$

Let $A_{k,l}$ be the **expectation of Y**

$$\begin{aligned} E(Y) &= \sum_x \sum_i E(Y_{x,i}) = \sum_x \sum_i P(Y_{x,i} = 1) = \\ &= \sum_x \sum_i P(\{\varepsilon_{k,l} \mid \pi_i = k \text{ and } \pi_{i+1} = l\}) = \\ &= \sum_x \sum_i P(\pi_i = k, \pi_{i+1} = l \mid x) \end{aligned}$$

Need to compute $P(\pi_i = k, \pi_{i+1} = l \mid x)$

Probabilistic setting for $E_k(b)$

Given x^1, \dots, x^m consider a **discrete probability space** with **elementary events**

$$\varepsilon_{k,b} = \text{"}b \text{ is emitted in state } k \text{ in } x^1, \dots, x^m \text{"}$$

For each x in $\{x^1, \dots, x^m\}$ and each position i in x let $Y_{x,i}$ be a **random variable** defined by

$$Y_{x,i}(\varepsilon_{k,b}) = \begin{cases} 1, & \text{if } x_i = b \text{ and } \pi_i = k \\ 0, & \text{otherwise} \end{cases}$$

Define $Y = \sum_x \sum_i Y_{x,i}$ random variable that counts # of times the event $\varepsilon_{k,b}$ happens in x^1, \dots, x^m .

The meaning of $E_k(b)$

Let $E_k(b)$ be the **expectation of Y**

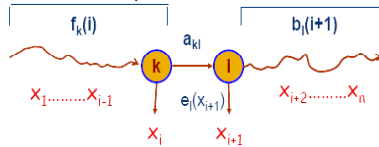
$$\begin{aligned} E(Y) &= \sum_x \sum_i E(Y_{x,i}) = \sum_x \sum_i P(Y_{x,i} = 1) = \\ &= \sum_x \sum_i P(\{\varepsilon_{k,b} \mid x_i = b \text{ and } \pi_i = k\}) = \\ &= \sum_x \sum_{\{i \mid x_i = b\}} P(\{\varepsilon_{k,b} \mid x_i = b, \pi_i = k\}) = \sum_x \sum_{\{i \mid x_i = b\}} P(\pi_i = k \mid x) \end{aligned}$$

Need to compute $P(\pi_i = k \mid x)$

Computing new parameters

Consider $x = x_1 \dots x_n$ training sequence

Concentrate on positions i and $i+1$



Use the forward-backward values:

$$f_{ki} = P(x_1 \dots x_i, \pi_i = k)$$

$$b_{ki} = P(x_{i+1} \dots x_n \mid \pi_i = k)$$

Compute A_{kl} (I)

Prob $k \rightarrow l$ is taken at position i of x

$$P(\pi_i = k, \pi_{i+1} = l \mid x_1 \dots x_n) = P(x, \pi_i = k, \pi_{i+1} = l) / P(x)$$

Compute $P(x)$ using either forward or backward values

We'll show that $P(x, \pi_i = k, \pi_{i+1} = l) = b_{l(i+1)} \cdot e_l(x_{i+1}) \cdot a_{kl} \cdot f_{ki}$

Expected # times $k \rightarrow l$ is used in training sequences

$$A_{kl} = \sum_x \sum_i (b_{l(i+1)} \cdot e_l(x_{i+1}) \cdot a_{kl} \cdot f_{ki}) / P(x)$$

Compute A_{kl} (2)

$$P(x, \pi_i = k, \pi_{i+1} = l) =$$

$$P(x_1 \dots x_p, \pi_i = k, \pi_{i+1} = l, x_{i+1} \dots x_n) =$$

$$P(\pi_{i+1} = l, x_{i+1} \dots x_n \mid x_1 \dots x_p, \pi_i = k) \cdot P(x_1 \dots x_p, \pi_i = k) =$$

$$P(\pi_{i+1} = l, x_{i+1} \dots x_n \mid \pi_i = k) \cdot f_{ki} =$$

$$P(x_{i+1} \dots x_n \mid \pi_i = k, \pi_{i+1} = l) \cdot P(\pi_{i+1} = l \mid \pi_i = k) \cdot f_{ki} =$$

$$P(x_{i+1} \dots x_n \mid \pi_{i+1} = l) \cdot a_{kl} \cdot f_{ki} =$$

$$P(x_{i+2} \dots x_n \mid x_{i+1}, \pi_{i+1} = l) \cdot P(x_{i+1} \mid \pi_{i+1} = l) \cdot a_{kl} \cdot f_{ki} =$$

$$P(x_{i+2} \dots x_n \mid \pi_{i+1} = l) \cdot e_l(x_{i+1}) \cdot a_{kl} \cdot f_{ki} =$$

$$b_{l(i+1)} \cdot e_l(x_{i+1}) \cdot a_{kl} \cdot f_{ki}$$

Compute $E_k(b)$

Prob x_i of x is emitted in state k

$$P(\pi_i = k \mid x_1 \dots x_n) = P(\pi_i = k, x_1 \dots x_n) / P(x)$$

$$P(\pi_i = k, x_1 \dots x_n) = P(x_1 \dots x_p, \pi_i = k, x_{i+1} \dots x_n) =$$

$$P(x_{i+1} \dots x_n \mid x_1 \dots x_p, \pi_i = k) \cdot P(x_1 \dots x_p, \pi_i = k) =$$

$$P(x_{i+1} \dots x_n \mid \pi_i = k) \cdot f_{ki} = b_{ki} \cdot f_{ki}$$

Expected # times b is emitted in state k

$$E_k(b) = \sum_x \sum_{i: x_i=b} (f_{ki} \cdot b_{ki}) / P(x)$$

Finally, new parameters

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

$$e_k(b) = E_k(b) / \sum_{b'} E_k(b')$$

Can add pseudocounts as before.

Stopping criteria

Cannot actually reach maximum (optimization of continuous functions)

Therefore need stopping criteria

• Compute the log likelihood of the model for current Θ

$$\sum_x \log P(x | \Theta)$$

Compare with previous log likelihood

Stop if small difference

• Stop after a certain number of iterations

The Baum-Welch algorithm

Initialization:

Pick the best-guess for model parameters
(or arbitrary)

Iteration:

1. Forward for each x
2. Backward for each x
3. Calculate $A_{k,b} E_k(b)$
4. Calculate new $a_{k,b} e_k(b)$
5. Calculate new log-likelihood

Until log-likelihood does not change much

Baum-Welch analysis

- Log-likelihood is increased by iterations
- Baum-Welch is a particular case of the EM (expectation maximization) algorithm
- Convergence to local maximum. Choice of initial parameters determines local maximum to which the algorithm converges