Hidden Markov Models

Outline

- CG-islands
- The “Fair Bet Casino”
- Hidden Markov Model
- Decoding Algorithm
- Forward-Backward Algorithm
- Profile HMMs
- HMM Parameter Estimation
- Viterbi training
- Baum-Welch algorithm

CG-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 *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

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: **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 $x = x_1x_2x_3\ldots x_n$ of coin tosses made by two possible coins (***F*** or ***B***).

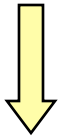
- **Output:** A sequence $\pi = \pi_1\ \pi_2\ \pi_3\ldots\ \pi_n$, with each π_i being either *F* or *B* indicating that x_i is the result of tossing the Fair or Biased coin respectively.

Problem…

Fair Bet Casino Problem

Any observed outcome of coin tosses could have been generated by any sequence of states!

Need to incorporate a way to grade different sequences differently.



Decoding Problem

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

- Suppose first that 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(*x*|fair coin)=P($x_1 \ldots x_n$|fair coin)

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

- P(*x*|biased coin)= P($x_1 \ldots x_n$|biased coin) =

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

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

- P(*x*|fair coin) = P(*x*|biased coin)
- $1/2^n = 3^k/4^n$
- $2^n = 3^k$
- $n = k \log_2 3$
- when $k = n / \log_2 3$ ($k \sim 0.67n$)

Log-odds Ratio

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

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

$$= \Sigma^{k}_{i=1} \log_2(p^{+}(x_i) / p^{-}(x_i))$$

$$= n - k \log_2 3$$

Computing Log-odds Ratio in Sliding Windows

$$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 \ldots x_n$$

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?

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.

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 (cont'd)

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

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

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

E = $(e_k(b))$: a |Q| x |Σ| matrix of probability of emitting symbol *b* while being in state *k.*

$e_F(0) = ½$ $e_F(1) = ½$

$e_B(0) = ¼$ $e_B(1) = ¾$

HMM for Fair Bet Casino

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

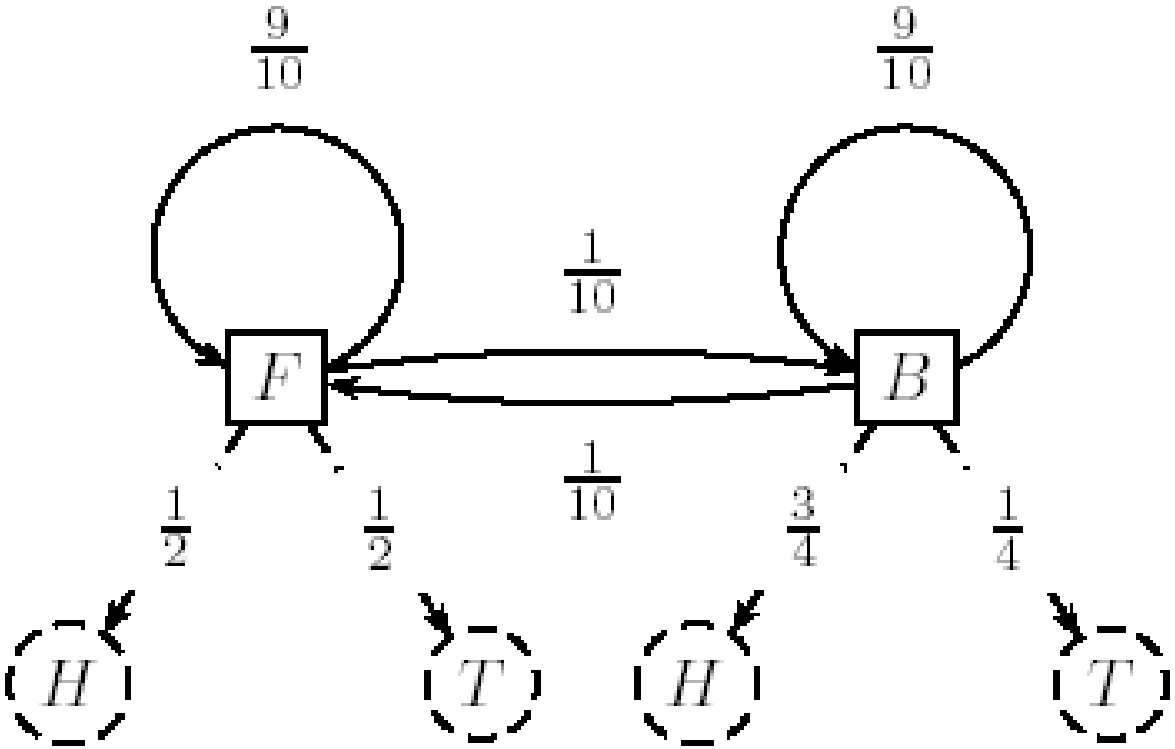
  Σ = {0, 1} (0 for ***T***ails and 1 ***H***eads)

  Q = {*F,B*} – *F* for Fair & *B* for Biased coin.
- Transition Probabilities *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	$e_F(0) = ½$	$e_F(1) = ½$
Biased	$e_B(0) = ¼$	$e_B(1) = ¾$

HMM for Fair Bet Casino (cont'd)



HMM model for the *Fair Bet Casino* Problem

Hidden Paths

- A *path* $\pi = \pi_1 \ldots \pi_n$ in the HMM is defined as a sequence of states.
- Consider path π = FFFBBBBBFFF and sequence 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
$P(x_i \mid \pi_i)$		½	½	½	¾	¾	¾	¼	¾	½	½	½
$P(\pi_{i-1} \to \pi_i)$		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 π_{i-1} to state π_i

P(*x*|π) Calculation

- P(*x*|π): Probability that sequence *x* was generated by the path *π:*

$$P(x|\pi) = P(\pi_0 \rightarrow \pi_1) \cdot \prod_{i=1}^{n} P(x_i|\pi_i) \cdot P(\pi_i \rightarrow \pi_{i+1})$$

$$= a_{\pi_0, \pi_1} \cdot \prod_{i=1}^{n} e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_{i+1}}$$

P(*x*|π) Calculation

- P(*x*|π): Probability that sequence *x* was generated by the path *π:*

$$P(x|\pi) = P(\pi_0 \rightarrow \pi_1) \cdot \prod_{i=1}^{n} P(x_i|\pi_i) \cdot P(\pi_i \rightarrow \pi_{i+1})$$

$$= a_{\pi_0, \pi_1} \cdot \prod_{i=1}^{n} e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_{i+1}}$$

$$= \prod_{i=1}^{n+1} e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_i}$$

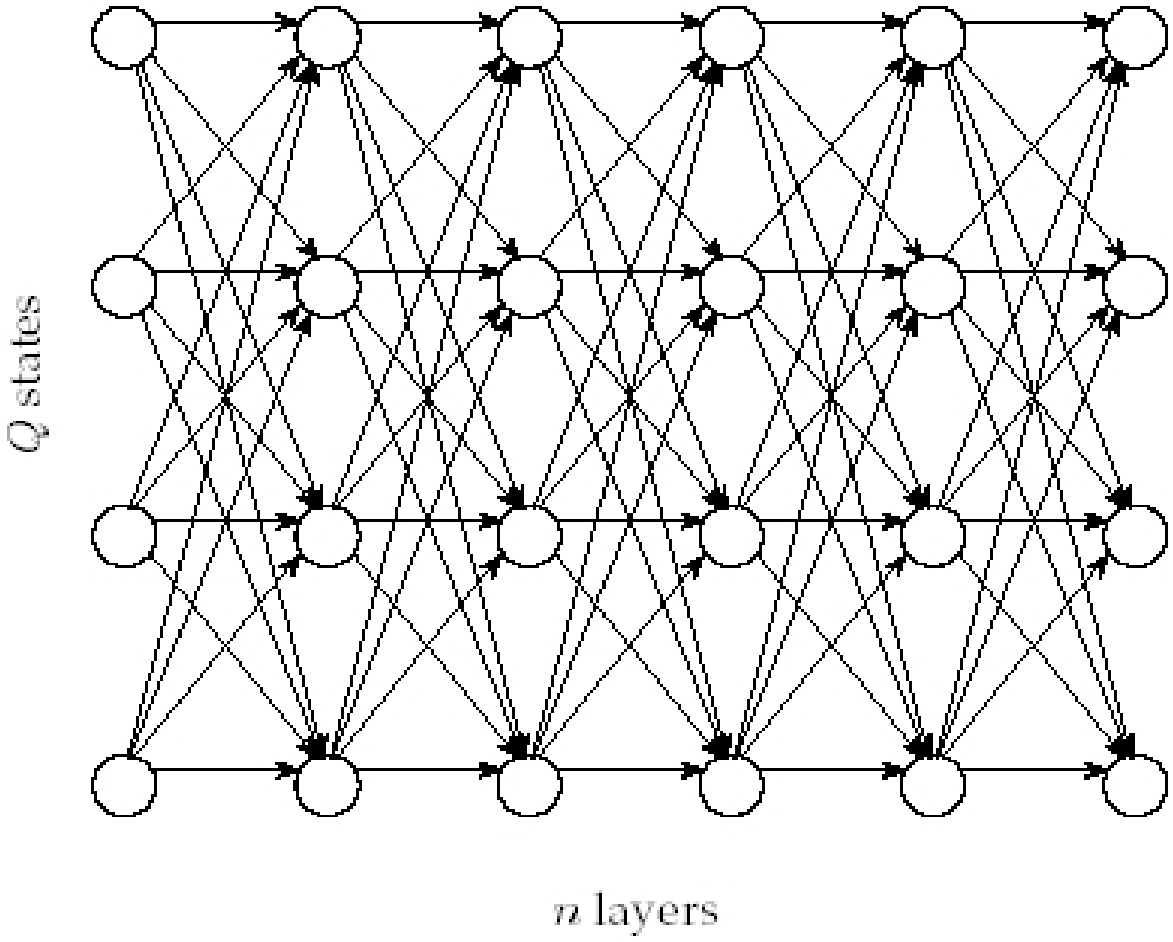
Decoding Problem

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

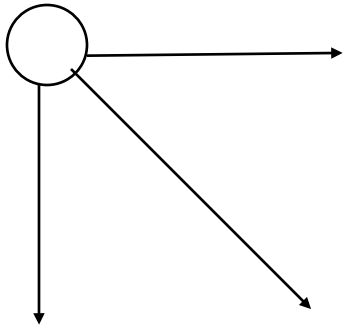
Building Manhattan for Decoding Problem

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

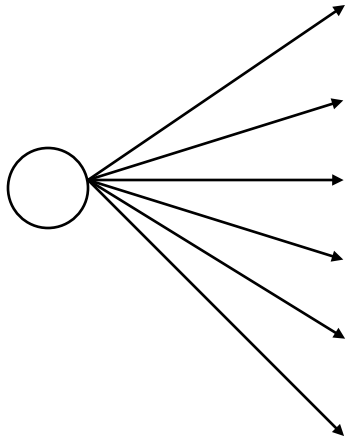
Edit Graph for Decoding Problem



Decoding Problem vs. Alignment Problem



Valid directions in the *alignment problem.*



Valid directions in the *decoding problem.*

Decoding Problem as Finding a Longest Path in a DAG

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

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

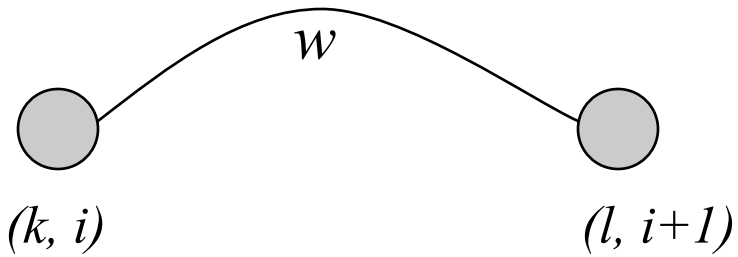
Decoding Problem (cont'd)

- Every path in the graph has the probability $P(x|\pi)$.

- The Viterbi algorithm finds the path that maximizes $P(x|\pi)$ among all possible paths.

- The Viterbi algorithm runs in $\boldsymbol{O(n|Q|^2)}$ time.

Decoding Problem: weights of edges

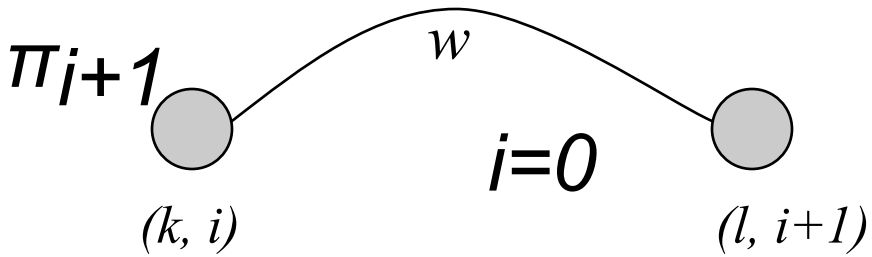


The weight ***w*** is given by:

???

Decoding Problem: weights of edges

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

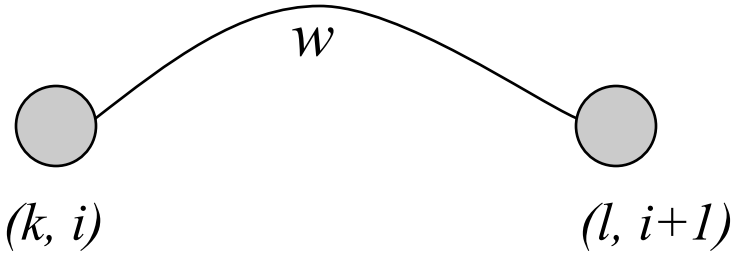


The weight ***w*** is given by:

??

Decoding Problem: weights of edges

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

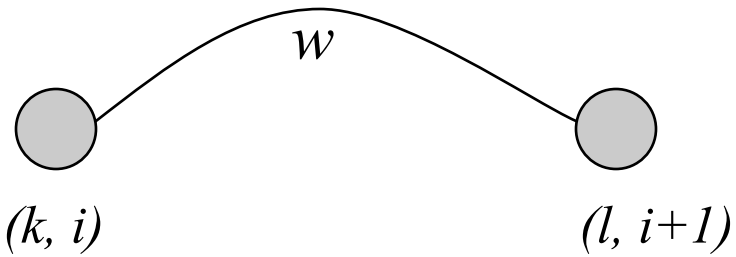


The weight ***w*** is given by:

?

Decoding Problem: weights of edges

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



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

Decoding Problem and Dynamic Programming

$s_{l,i+1} = \max_{k \in Q} \{s_{k,i} \cdot$ weight of edge between (k,i) 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 for $k \neq begin$.
- Let π^* be the optimal path. Then,

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

Viterbi Algorithm

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

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

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 $x_1 \ldots x_i$ and reaching the state $\pi = k$.
- The recurrence for the forward algorithm:

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

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 $P(\pi_i = k|x)$.

Backward Algorithm (cont'd)

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

$$P(\pi_i = k|x) = \frac{P(x, \pi_i = k)}{P(x)} = \frac{f_k(i) \cdot b_k(i)}{P(x)}$$

P(x) is the sum of $P(x, \pi_i = k)$ over all k

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.
- 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.

Profiles and HMMs

- HMMs can also be used for aligning a sequence against a profile representing protein family.
- A $20 \cdot n$ profile P corresponds to n sequentially linked *match* states $M_1,\ldots,M_n$ in the profile HMM of P.

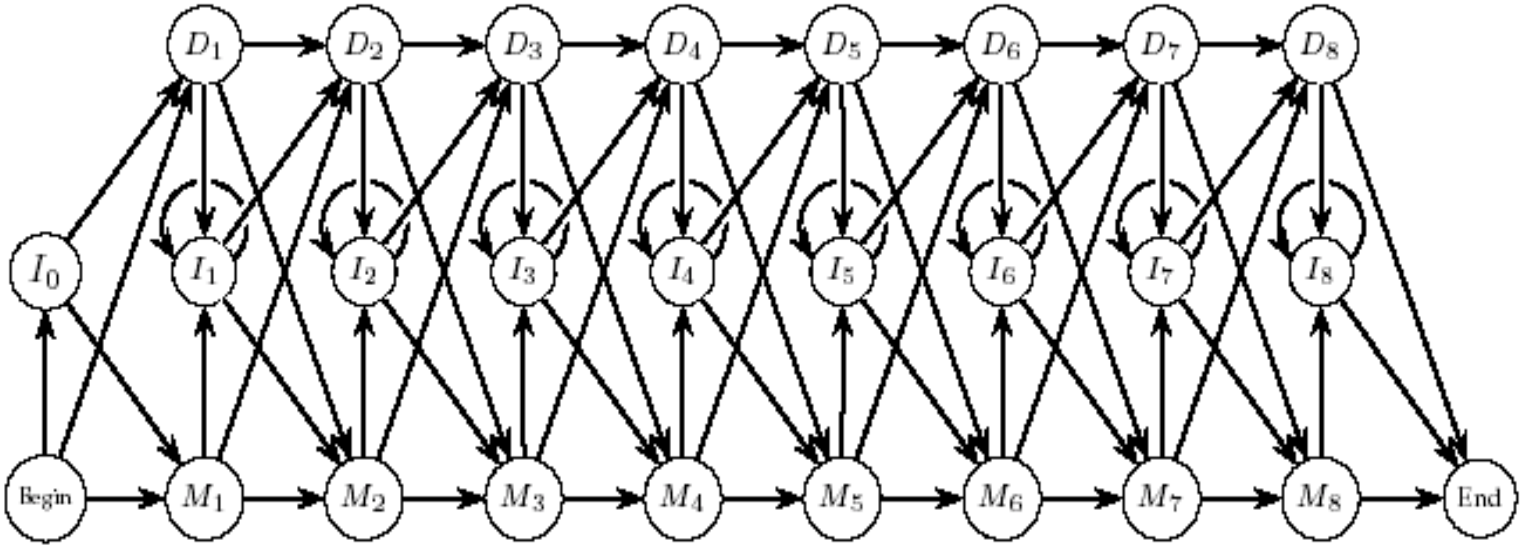
Multiple Alignments and Protein Family Classification

- Multiple alignment of a protein family shows variations in conservation along the length of a protein
- Example: after aligning many globin proteins, the biologists recognized that the helices region in globins are more conserved than others.

What are Profile HMMs ?

- A Profile HMM is a probabilistic representation of a multiple alignment.
- A given multiple alignment (of a protein family) is used to build a profile HMM.
- This model then may be used to find and score less obvious potential matches of new protein sequences.

Profile HMM



A profile HMM

Building a profile HMM

- Multiple alignment is used to construct the HMM model.
- Assign each column to a *Match* state in HMM. Add I*nsertion* and *Deletion* state.
- Estimate the emission probabilities according to amino acid counts in column. Different positions in the protein will have different emission probabilities.
- Estimate the transition probabilities between *Match, Deletion* and *Insertion* states
- The HMM model gets trained to derive the optimal parameters.

States of Profile HMM

- Match states $M_1 \ldots M_n$ (plus *begin/end* states)
- Insertion states $I_0 I_1 \ldots I_n$
- Deletion states $D_1 \ldots D_n$

Transition Probabilities in Profile HMM

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

Emission Probabilities in Profile HMM

- Probabilty of emitting a symbol *a* at an insertion state I_j:

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

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

Profile HMM Alignment

- Define $v^M_j(i)$ as the logarithmic likelihood score of the best path for matching $x_1..x_i$ to profile HMM ending with x_i emitted by the state M_j.

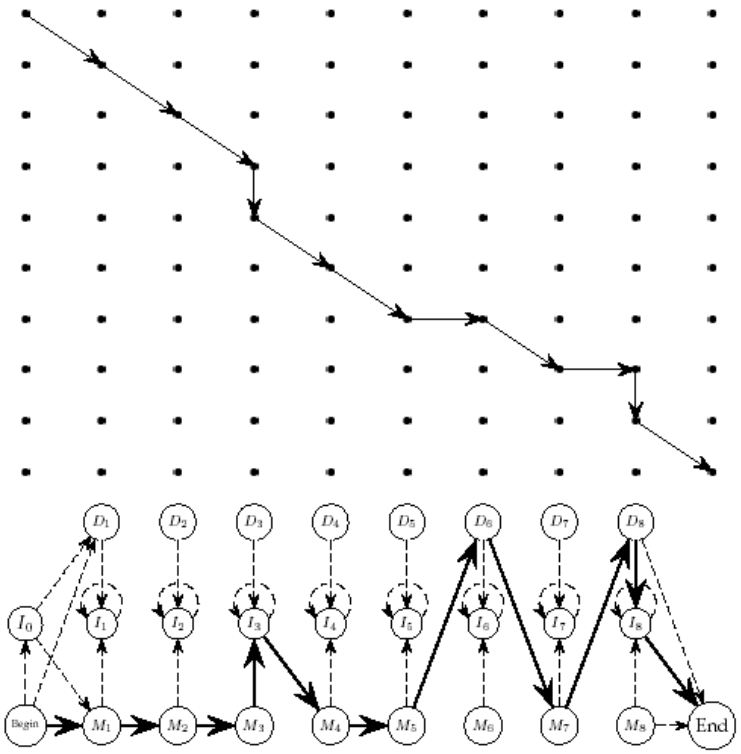
- $v^I_j(i)$ and $v^D_j(i)$ are defined similarly.

Profile HMM Alignment: Dynamic Programming

$$v^{M}_{j}(i) = \log\left(e_{M_j}(x_i)/p(x_i)\right) + \max \begin{cases} v^{M}_{j-1}(i-1) + \log(a_{M_{j-1},M_j}) \\ v^{I}_{j-1}(i-1) + \log(a_{I_{j-1},M_j}) \\ v^{D}_{j-1}(i-1) + \log(a_{D_{j-1},M_j}) \end{cases}$$

$$v^{I}_{j}(i) = \log\left(e_{I_j}(x_i)/p(x_i)\right) + \max \begin{cases} v^{M}_{j}(i-1) + \log(a_{M_j, I_j}) \\ v^{I}_{j}(i-1) + \log(a_{I_j, I_j}) \\ v^{D}_{j}(i-1) + \log(a_{D_j, I_j}) \end{cases}$$

Paths in Edit Graph and Profile HMM



A path through an edit graph and the corresponding path through 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

Application of Profile HMM to Modeling Globin Proteins

- Globins represent a large collection of protein sequences
- 400 globin sequences were randomly selected from all globins and used to construct a multiple alignment.
- Multiple alignment was used to assign an initial HMM
- This model then get trained repeatedly with model lengths chosen randomly between 145 to 170, to get an HMM model optimized probabilities.

How Good is the Globin HMM?

- 625 remaining globin sequences in the database were aligned to the constructed HMM resulting in a multiple alignment. This multiple alignment agrees extremely well with the structurally derived alignment.
- 25,044 proteins, were randomly chosen from the database and compared against the globin HMM.
- This experiment resulted in an excellent separation between globin and non-globin families.

PFAM

- Pfam decribes ***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.
  - HMM profiles and full alignments can be updated with newly found amino acid sequences.

PFAM Uses

- Pfam HMMs span entire domains that include both well-conserved motifs and less-conserved regions with insertions and deletions.
- It results in modeling complete domains that facilitates better sequence annotation and leeds to a more sensitive detection.

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. It's very hard to estimate the probabilities.

HMM Parameter Estimation Problem

- Given
  - HMM with states and alphabet (emission characters)
  - Independent training sequences $x^1, \ldots x^m$
- Find HMM parameters Θ (that is, a_{kl}, $e_k(b)$) that maximize

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

the joint probability of the training sequences.

Maximize the likelihood

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

The training sequences are assumed independent, therefore

$$P(x^1, \ldots, x^m \mid \Theta) = \Pi_i P(x^i \mid \Theta)$$

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

Two situations

Known paths for training sequences

- CpG islands marked on training sequences
- One evening the casino dealer allows us to see when he changes dice

Unknown paths

- CpG islands are not marked
- Do not see when the casino dealer changes dice

Known paths

A_{kl} = # of times each $k \rightarrow l$ is taken 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:

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

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

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

  $E_k(b)$ = # of emissions of *b* 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 path for training sequence *x*

- Start with some guess for initial parameters and compute π^* the most probable path for *x* using initia parameters.
- Iterate until no change in π^* :

4. Determine A_{kl} and $E_k(b)$ as before
5. Compute new parameters a_{kl} and $e_k(b)$ using the same formulas as before
6. Compute new π^* for *x* and the current parameters

Viterbi training analysis

- The algorithm converges precisely

  There are finitely many possible paths.

  New parameters are uniquely determined by the current π^*.

  There may be several paths for x with the same probability, hence must compare the new π^* with all previous paths having highest probability.

- Does not maximize the likelihood $\Pi_x P(x \mid \Theta)$ but the contribution to the likelihood of the most probable path $\Pi_x P$

  $(x \mid \Theta, \pi^*)$

- In general performs less well than Baum-Welch

Unknown paths: Baum-Welch

Idea:

2. Guess initial values for parameters.

   art and experience, not science

4. Estimate new (better) values for parameters.

   how ?

6. 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 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 $x^1, \ldots, x^m$ consider a discrete probability space with elementary events

$\epsilon_{k,l}$ = "$k \rightarrow l$ is taken in $x^1, \ldots, x^m$"

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

$$Y_{x,i}(\epsilon_{k,l}) = \begin{cases} 1, \text{ if } x_i = k \text{ and } x_{i+1} = l \\ 0, \text{ otherwise} \end{cases}$$

Define $Y = \Sigma_x \Sigma_i Y_{x,i}$ random var that counts # of times the event $\epsilon_{k,l}$ happens in $x^1, \ldots, x^m$.

The meaning of A_{kl}

Let A_{kl} be the expectation of Y

$$E(Y) = \Sigma_x \Sigma_i E(Y_{x,i}) = \Sigma_x \Sigma_i P(Y_{x,i} = 1) =$$
$$\Sigma_x \Sigma_i P(\{\epsilon_{k,l} \mid \pi_i = k \text{ and } \pi_{i+1} = l\}) =$$
$$\Sigma_x \Sigma_i P(\pi_i = k, \pi_{i+1} = l \mid x)$$

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

Probabilistic setting for $E_k(b)$

Given $x^1, \ldots, x^m$ consider a discrete probability space with elementary events

$\epsilon_{k,b}$ = "b is emitted in state k in $x^1, \ldots, x^m$"

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

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

Define $Y = \Sigma_x \Sigma_i Y_{x,i}$ random var that counts # of times the event $\epsilon_{k,b}$ happens in $x^1,\ldots,x^m$.

The meaning of $E_k(b)$

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

$$E(Y) = \Sigma_x \Sigma_i E(Y_{x,i}) = \Sigma_x \Sigma_i P(Y_{x,i} = 1) =$$

$$\Sigma_x \Sigma_i P(\{\epsilon_{k,b} \mid x_i = b \text{ and } \pi_i = k\})$$

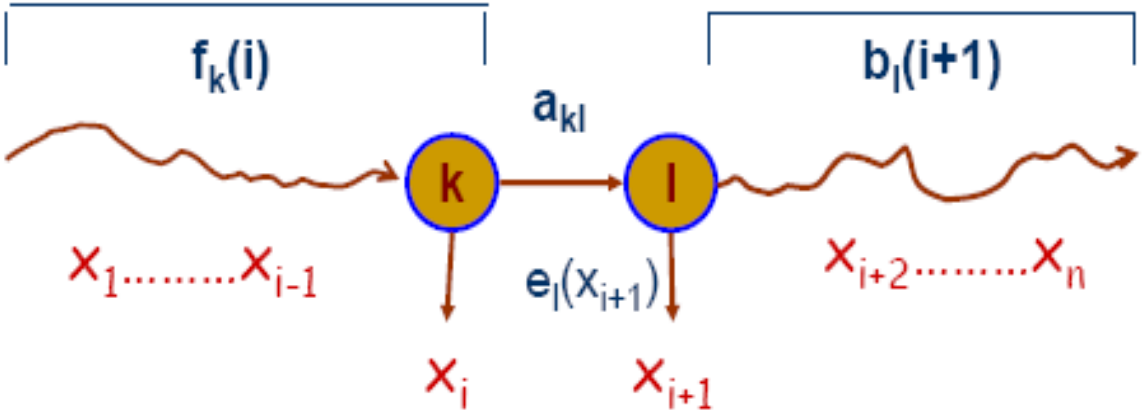
$$\sum_x \sum_{\{i|x_i=b\}} P(\{\epsilon_{k,b} \mid x_i = b, \pi_i = k\}) = \sum_x \sum_{\{i|x_i=b\}} P(\pi_i = k \mid x)$$

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

Computing new parameters

Consider $x = x_1 \ldots x_n$ training sequence

Concentrate on positions *i* and *i+1*



Use the forward-backward values:

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

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

Compute A_{kl} (1)

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

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

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

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

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

$$A_{kl} = \Sigma_x \Sigma_i (b_{li+1} \cdot e_l(x_{i+1}) \cdot a_{kl} \cdot f_{ki}) / P(x)$$

Compute A_{kl} (2)

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

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

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

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

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

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

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

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

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

Compute $E_k(b)$

Prob x_i of x is emitted in state k

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

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

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

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

Expected # times b is emitted in state k

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

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

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

  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_{kl}, $E_k(b)$
4. Calculate new a_{kl}, $e_k(b)$
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

Speech Recognition

- Create an *HMM* of the words in a language
  - Each word is a hidden state in *Q*.
  - Each of the basic sounds in the language is a symbol in Σ.
- **Input:** use speech as the input sequence.
- **Goal:** find the most probable sequence of states.

Speech Recognition: Building the Model

- Analyze some large source of English sentences, such as a database of newspaper articles, to form probability matrixes.
  - A_{0i}: the chance that word *i* begins a sentence.
  - A_{ij}: the chance that word *j* follows word *i*.

Building the Model (cont'd)

- Analyze English speakers to determine what sounds are emitted with what words.
  - $E_k(b)$: the chance that sound *b* is spoken in word *k*. Allows for alternate pronunciation of words.

Speech Recognition: Using the Model

- Use the same dynamic programming algorithm as before
  - Weave the spoken sounds through the model the same way we wove the rolls of the die through the casino model.
  - π represents the most likely set of words.

Using the Model (cont'd)

- How well does it work?
  - Common words, such as 'the', 'a', 'of' make prediction less accurate, since there are so many words that follow normally.

Improving Speech Recognition

- Initially, we were using a *'bigram,'* a graph connecting every two words.
- Expand that to a *'trigram'*
  - Each state represents two words spoken in succession.
  - Each edge joins those two words (*A B*) to another state representing (*B C*)
  - Requires n^3 vertices and edges, where n is the number of words in the language.
  - Much better, but still limited context.

References

- Slides for CS 262 course at Stanford given by Serafim Batzoglou