Hidden Markov Models

Outline

1. CG-Islands
2. The "Fair Bet Casino"
3. Hidden Markov Model
4. Decoding Algorithm
5. Forward-Backward Algorithm
6. Profile HMMs
7. HMM Parameter Estimation
8. Viterbi Training
9. Baum-Welch Algorithm

Section 1: CG-Islands

CG-Islands

- Given 4 nucleotides: probability of any one's occurrence is ~ 1/4.
- Thus, probability of occurrence of a given **dinucleotide** (pair of successive nucleotides 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)

CG-Islands

- CG is the least frequent dinucleotide because the C in CG is easily **methylated** and has the tendency to mutate into *T* afterwards.

- However, 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 biological problem.

Section 2: The Fair Bet Casino

The "Fair Bet Casino"

- The CG-islands problem can be modeled after a problem named **The Fair Bet Casino**.
  - The game is to flip two coins, which results in only two possible outcomes: Head (**H**) or Tail(**T**).
  - The Fair coin (**F**) will give **H** and **T** each with probability ½, which is written P(**H** | **F**) = P(**T** | **F**) = ½.
  - The Biased coin (**B**) will give **H** with probability ¾, which we write as P(**H** | **B**) = ¾, P(**T** | **B**) = ¼.
  - The crooked dealer changes between **F** and **B** coins with probability 10%.
  - How can we tell when the dealer is using **F** and when he is using **B**?

The Fair Bet Casino Problem

- Input: A sequence $x = x_1x_2x_3...x_n$ of coin tosses made by two possible coins (**F** or **B**).

- Output: A sequence $\pi = \pi_1 \pi_2 \pi_3... \pi_n$, with each π_i being either **F** or **B** and 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!
  - **Example**: HHHHHHHHHH could be generated by BBBBBBBBBB, FFFFFFFFFF, FBFBFBFBFB, etc.

- We need to incorporate a way to grade different sequences differently.

- This provides us with the decoding problem.

Simple Case: The Dealer Never Switches Coins

- We assume first that the dealer never changes coins:
  - P(x | **F**): probability of the dealer using **F** and generating the outcome x.
  - P(x | **B**): probability of the dealer using the B coin and generating outcome x.
  - **Example**: Say that in x we observe k heads and $n - k$ tails:

$$P(x \mid \mathrm{F}) = \prod_{i=1}^{n} \left(\frac{1}{2}\right) = \left(\frac{1}{2}\right)^n$$

$$P(x \mid \mathrm{B}) = \left(\frac{3}{4}\right)^k \left(\frac{1}{4}\right)^{n-k} = \frac{3^k}{4^n}$$

When Does P(*x* | **F**) = P(*x* | **B**)?

$$P(x \mid \mathrm{F}) = P(x \mid \mathrm{B})$$

$$\frac{1}{2^n} = \frac{3^k}{4^n}$$

$$2^n = 3^k$$

$$n = \log_2 3^k$$

$$n = k \log_2 3$$

$$\therefore k = \frac{n}{\log_2 3}$$

Log-odds Ratio

- We define the **log-odds ratio** (L) as follows:

$$L = \log_2\left(\frac{P(x \mid \mathrm{F})}{P(x \mid \mathrm{B})}\right)$$

$$= \log_2\left(\frac{1}{2^n}\right) - \log_2\left(\frac{3^k}{4^n}\right)$$

$$= -n - \left[\log_2\left(3^k\right) - \log_2 4^n\right]$$

$$= -n - k\log_2(3) + 2n$$

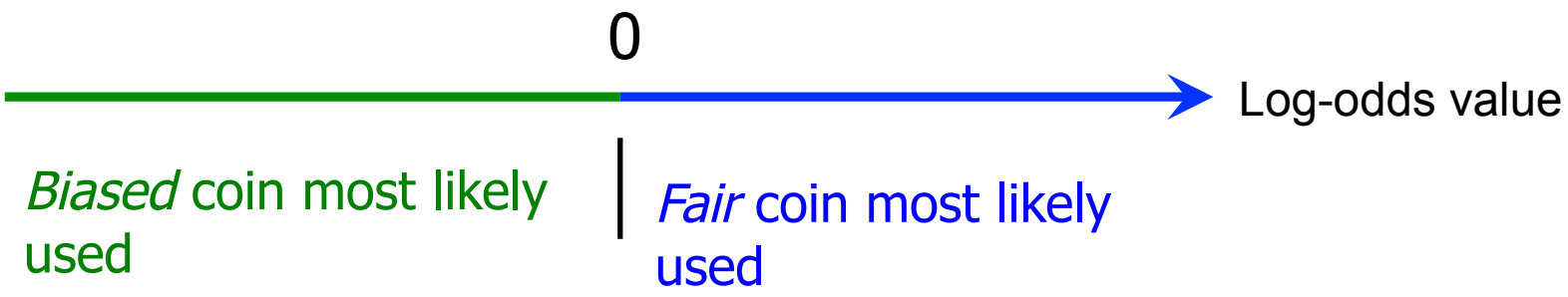
$$= n - k\log_2(3)$$

- From the previous slide, if $L > 0$ we have reason to believe that the coin is fair, and if $L < 0$ we think the coin is biased.

Computing Log-odds Ratio in Sliding Windows

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

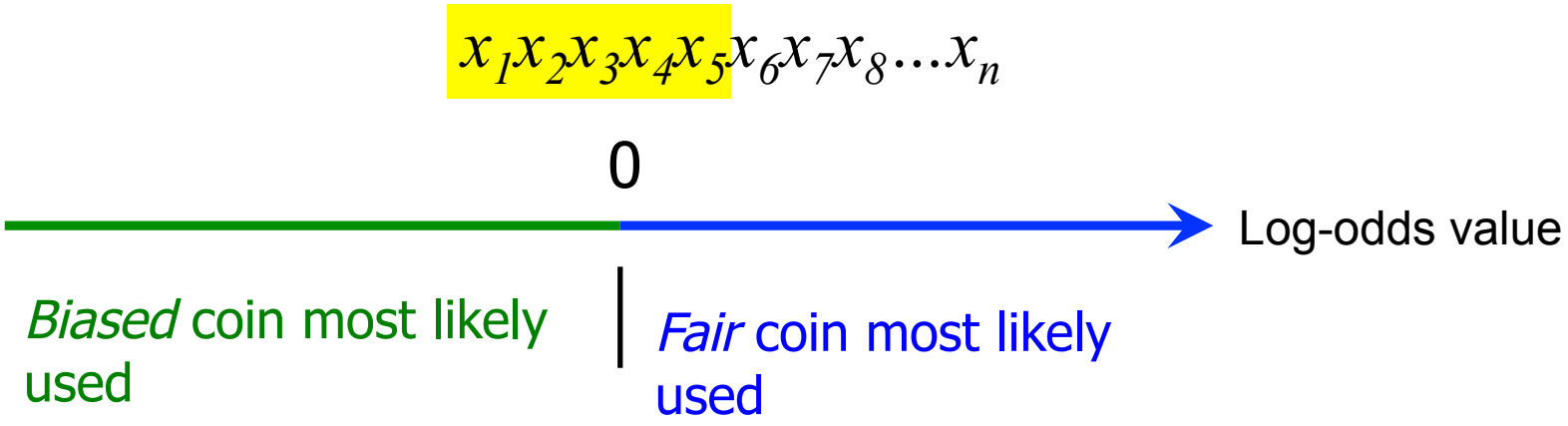
$$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 \ldots x_n$$



- Key Disadvantages:
  - The length of the CG-island is not known in advance.
  - Different windows may classify the same position differently.

Computing Log-odds Ratio in Sliding Windows

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

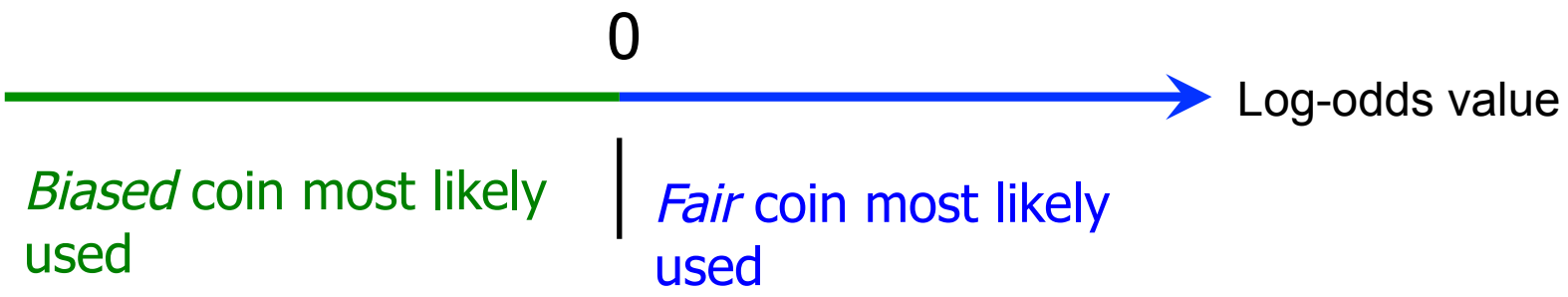


- Key Disadvantages:
  - The length of the CG-island is not known in advance.
  - Different windows may classify the same position differently.

Computing Log-odds Ratio in Sliding Windows

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

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

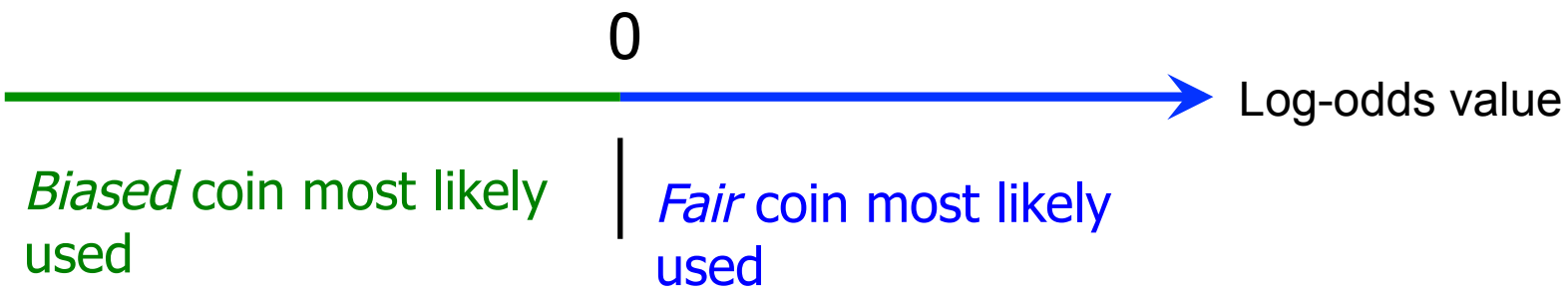


- Key Disadvantages:
  - The length of the CG-island is not known in advance.
  - Different windows may classify the same position differently.

Computing Log-odds Ratio in Sliding Windows

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

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

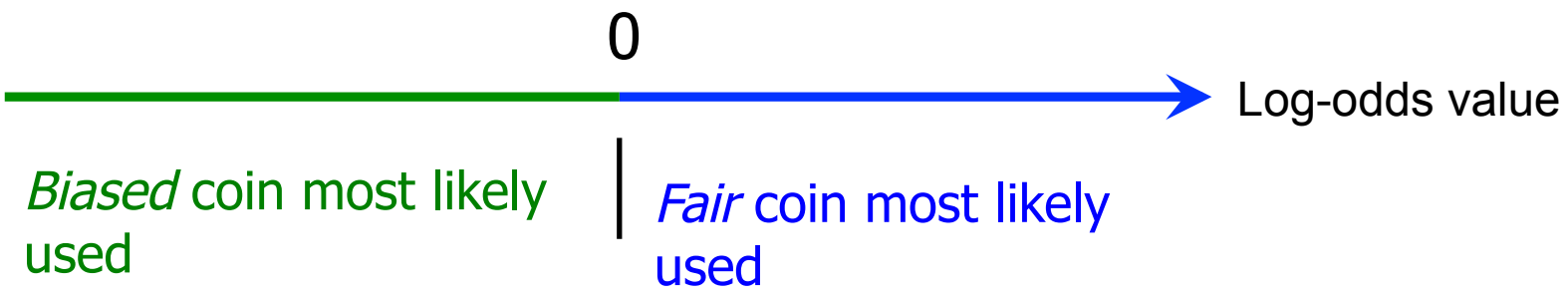


- Key Disadvantages:
  - The length of the CG-island is not known in advance.
  - Different windows may classify the same position differently.

Computing Log-odds Ratio in Sliding Windows

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

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



- Key Disadvantages:
  - The length of the CG-island is not known in advance.
  - Different windows may classify the same position differently.

Section 3: Hidden Markov Models

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 *and* chooses characters according to this probability distribution.

- While in a certain state, the machine makes two decisions:
  1. What state should I move to next?
  2. 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*.

- 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.
- Q: set of hidden states, each emitting symbols from Σ.
- $A = (a_{kl})$: a |Q| x |Q| matrix containing the probabilities of changing from state *k* to state *l*.
- $E = (e_k(b))$: a |Q| x |Σ| matrix of probability of emitting symbol *b* while being in state *k*.

HMM Parameters

- $A = (a_{kl})$: a $|Q|$ x $|Q|$ matrix containing the probabilities of changing from state k to state l.
  - $a_{\mathbf{FF}} = 0.9$ $a_{\mathbf{FB}} = 0.1$
  - $a_{\mathbf{BF}} = 0.1$ $a_{\mathbf{BB}} = 0.9$

- $E = (e_k(b))$: a $|Q|$ x $|\Sigma|$ matrix of probability of emitting symbol b while being in state k.
  - $e_{\mathbf{F}}(0) = ½$ $e_{\mathbf{F}}(1) = ½$
  - $e_{\mathbf{B}}(0) = ¼$ $e_{\mathbf{B}}(1) = ¾$

HMM for the Fair Bet Casino

- The Fair Bet Casino in HMM terms:
  - $\Sigma = \{0, 1\}$ (0 for **T** and 1 for **H**)
  - $Q = \{\mathbf{F}, \mathbf{B}\}$

Transition Probabilities (*A*)

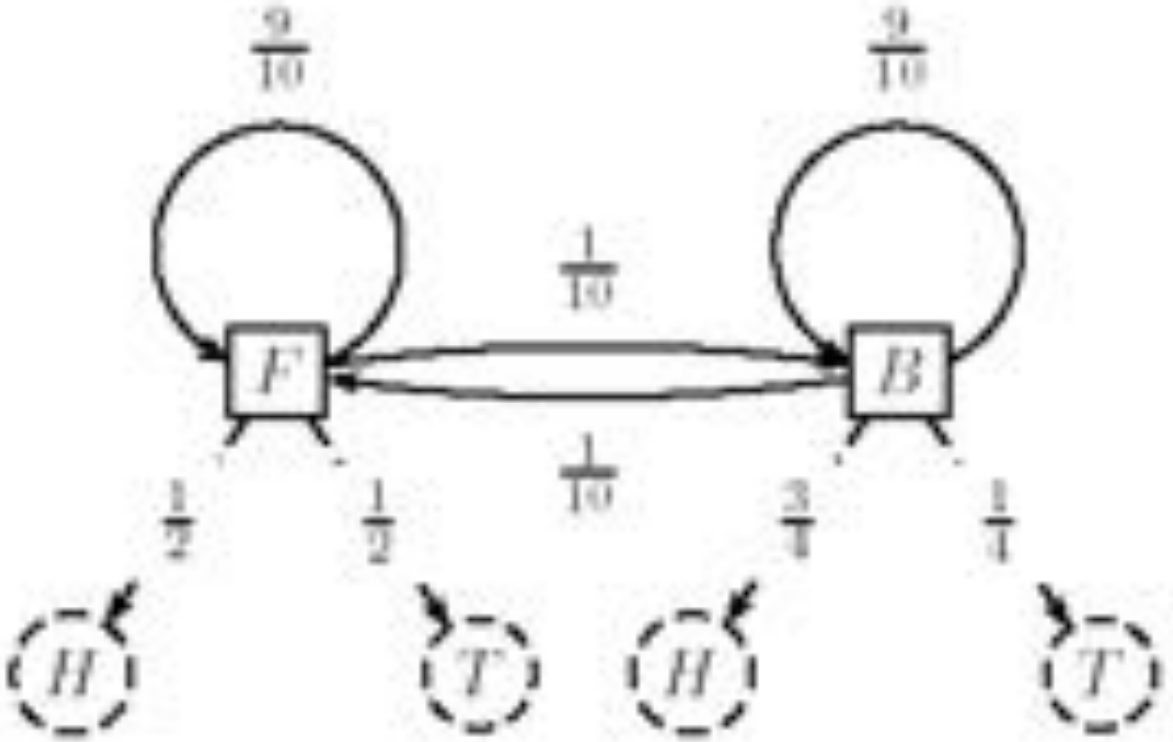
	Fair	Biased
Fair	$a_{FF} = 0.9$	$a_{FB} = 0.1$
Biased	$a_{BF} = 0.1$	$a_{BB} = 0.9$

Emission Probabilities (*E*)

	Tails(0)	Heads(1)
Fair	$e_F(0) = ½$	$e_F(1) = ½$
Biased	$e_B(0) = ¼$	$e_B(1) = ¾$

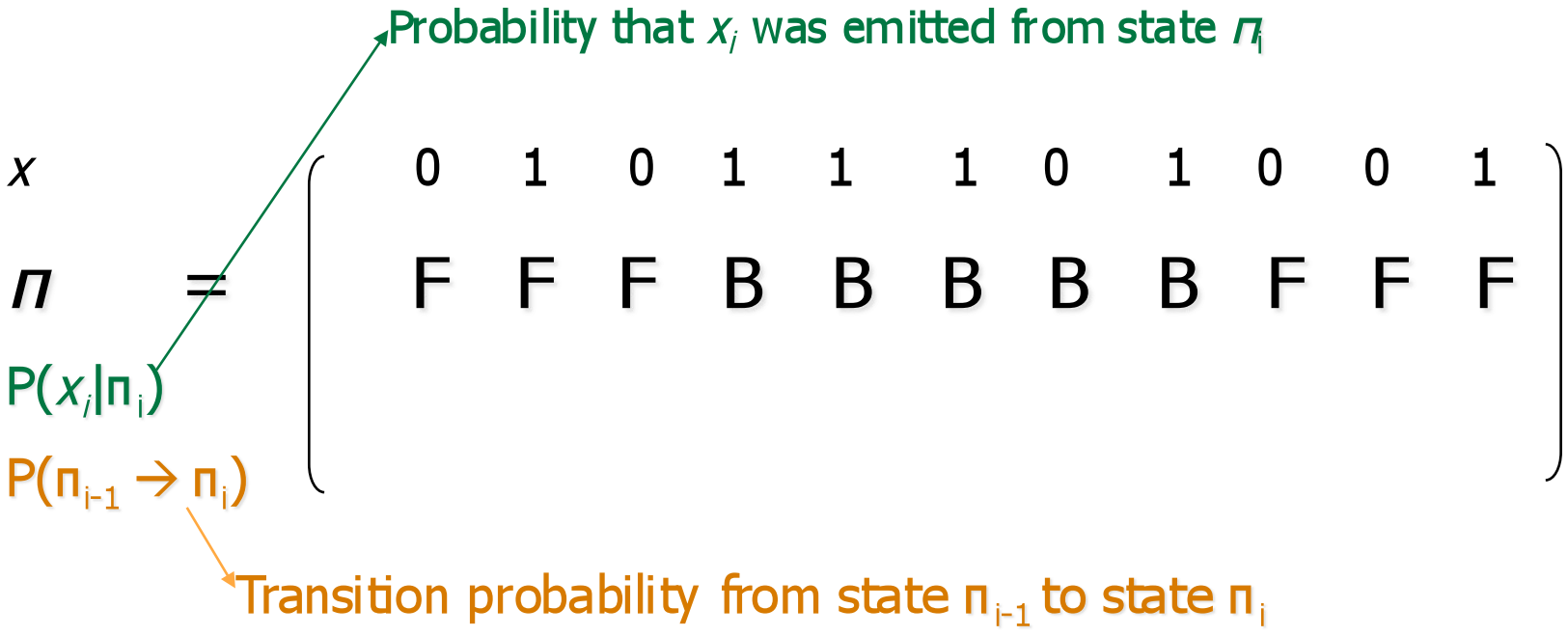
HMM for the Fair Bet Casino

- HMM model for the Fair Bet Casino Problem:



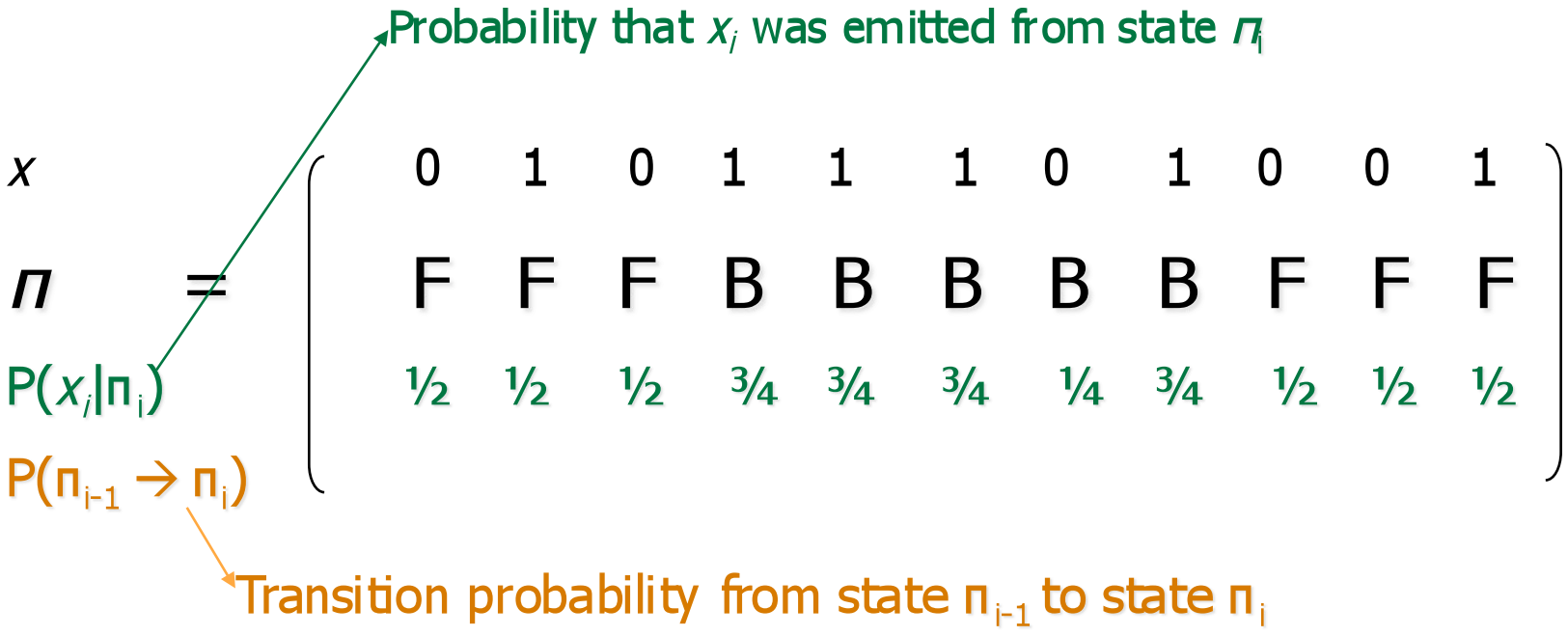
Hidden Paths

- 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



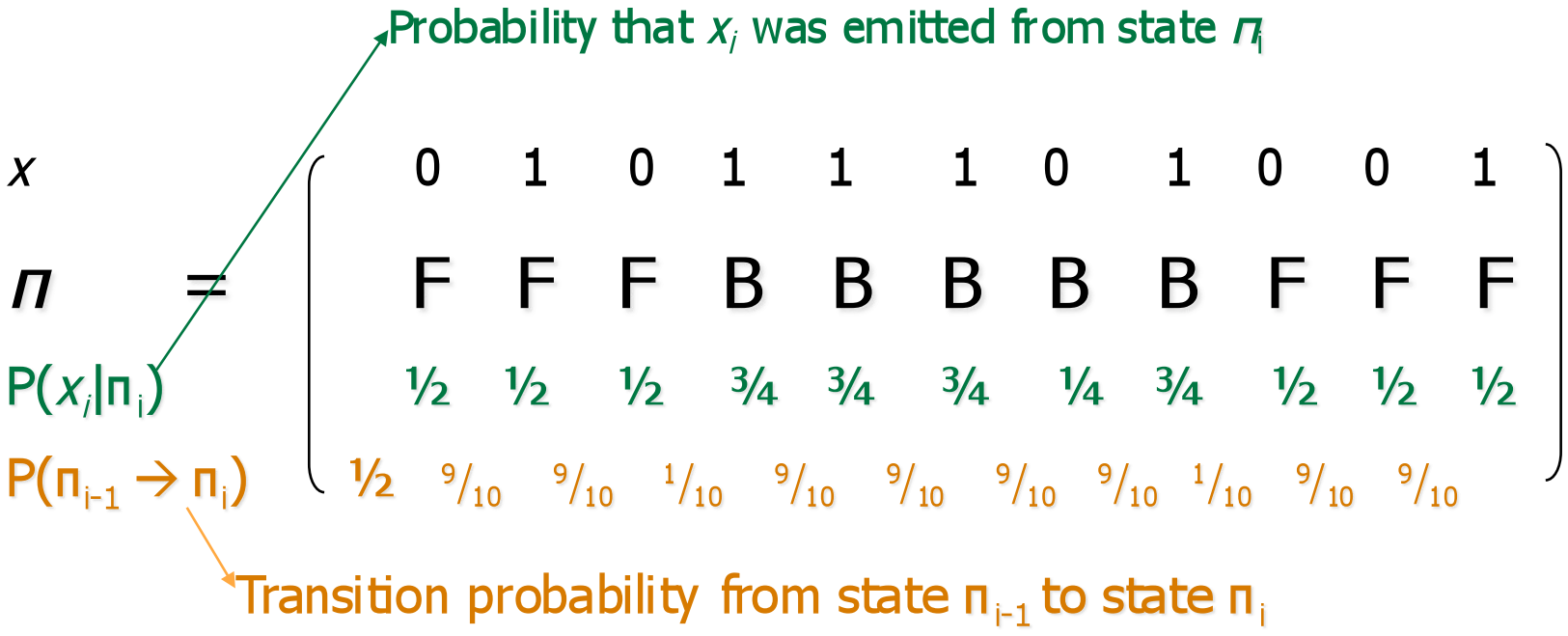
Hidden Paths

- 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



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



P(x | π) Calculation

- P(x | π): Probability that sequence x was generated if we know that we have the path π.

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

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

P(x | π) Calculation

- P(x | π): Probability that sequence x was generated if we know that we have the path π.

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

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

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

Section 4: Decoding Algorithm

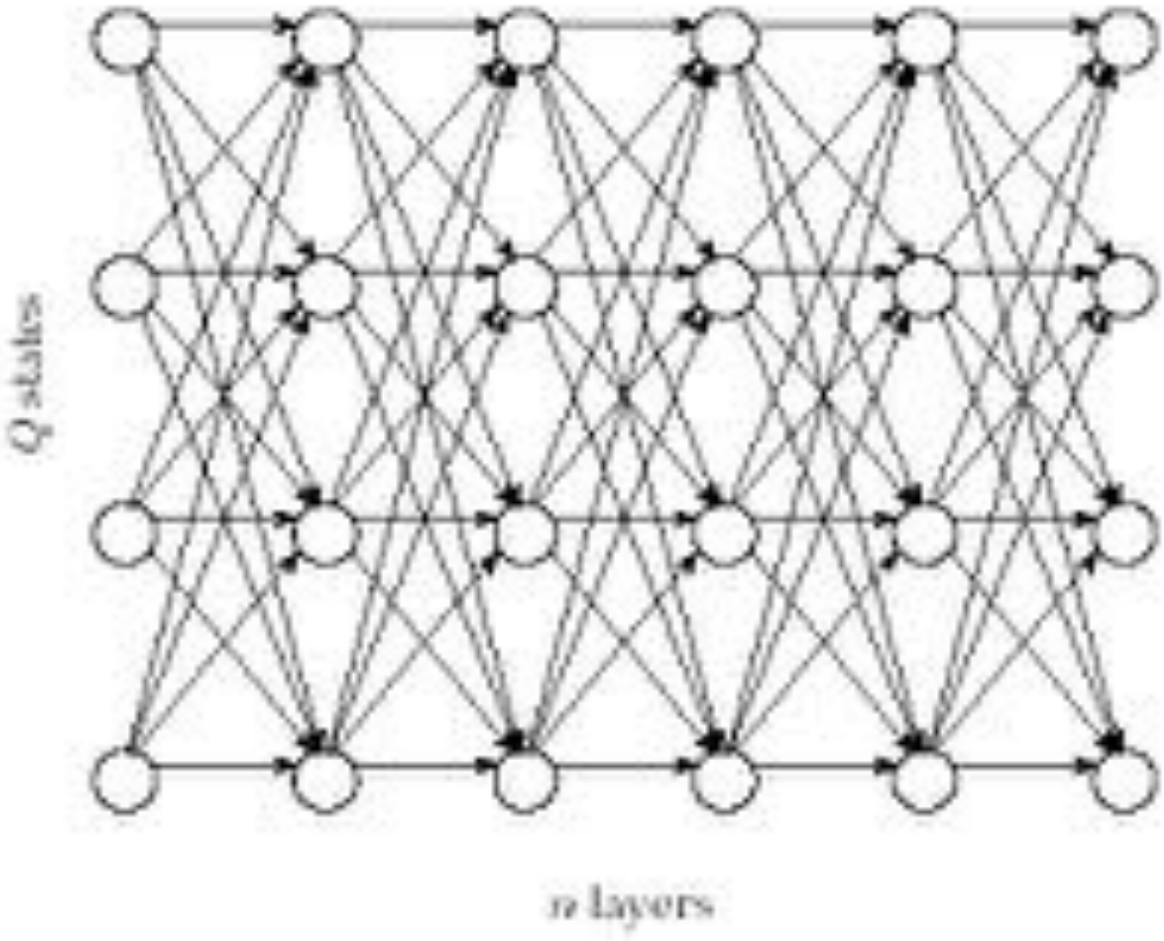
Decoding Problem

- 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 $P(x \mid \pi)$ over all possible paths π.

Building Manhattan for Decoding Problem

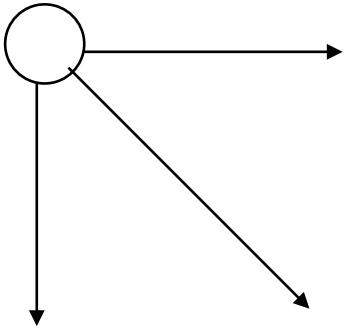
- Andrew Viterbi used the Manhattan edit graph model to solve the Decoding Problem.
- Vertices are composed of n "levels" with $|Q|$ vertices in each level; each vertex represents a different state.
- We connect each vertex in level i to each vertex in level $i + 1$ via a directed edge, giving $|Q|^2(n - 1)$ edges.
- Therefore every choice of $\pi = \pi_1 \ldots \pi_n$ corresponds to a path in the graph.

Edit Graph for Decoding Problem: Example

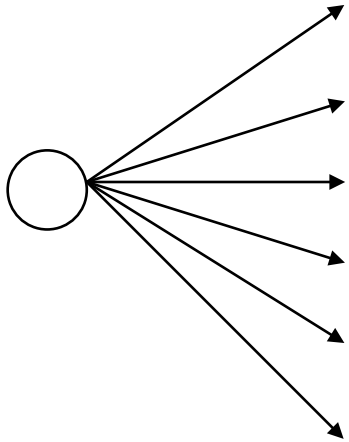


Decoding Problem vs. Alignment Problem

Valid Directions in Alignment



Valid Directions in Decoding

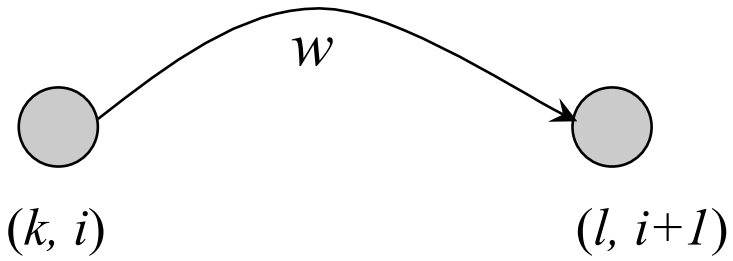


Decoding Problem

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

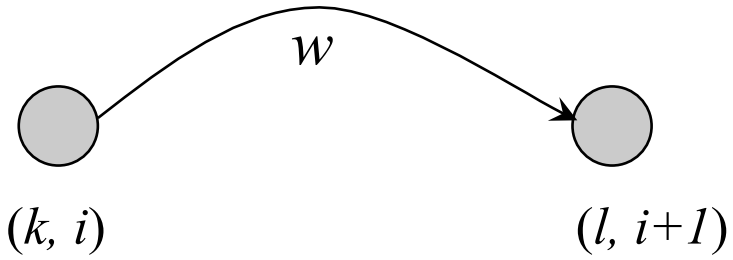
Decoding Problem: Weights of Edges

- The weight w is given by: ?



Decoding Problem: Weights of Edges

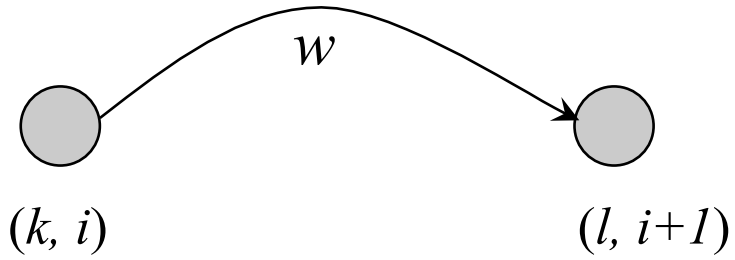
- The weight w is given by: ?



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

Decoding Problem: Weights of Edges

- The weight w is given by: ?

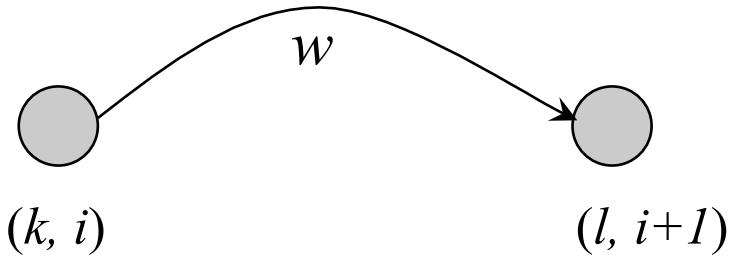


$$P(x \mid \pi) = \prod_{i=0}^{n-1} e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i, \pi_{i+1}}$$

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

Decoding Problem: Weights of Edges

- The weight w is given by: $e_l(x_{i+1}) \cdot a_{k,\,l}$



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

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

Decoding Problem and Dynamic Programming

- $s_{l,\, i+1}$ = max probability of all paths of length $i + 1$ ending in state l (for the first $i + 1$ observations).

- Recursion:

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

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

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

Decoding Problem and Dynamic Programming

- The value of the product can become extremely small, which leads to **overflow**.
  - A computer has only finite storage to store any given number, and if the number is too small it runs out of room.

- To avoid overflow, take the logarithm of the right side instead.

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

Decoding Problem and Dynamic Programming

- Initialization:

$$s_{k,0} = \begin{cases} 1 & \text{if } k = begin \\ 0 & \text{otherwise} \end{cases}$$

- Let π^* be the optimal path. Then,

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

Section 5: Forward-Backward Algorithm

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 Probability

- Define $f_{k,i}$ (**forward probability**) as the probability of emitting the *prefix* $x_1 \ldots x_i$ and reaching the state $\pi = k$.

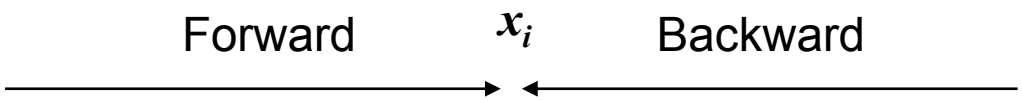
- The recurrence for the forward algorithm:

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

Backward Probability

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

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



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

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

Backward-Forward Probability

- The probability that HMM is in a certain state k at any moment i, given that we observe the output x, is therefore influenced by both the forward and backward probabilities.

- We use the mathematical definition of conditional probability to calculate $P(\pi_i = k \mid x)$:

$$P\left(\pi_i = k \mid x\right) = \frac{P\left(x, \pi_i = k\right)}{P\left(x\right)} = \frac{f_{k,i} \cdot b_{k,i}}{P\left(x\right)}$$

Section 6: Profile HMMs

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 a significance test.
- However, they may have these weak similarities with *many* members of the family, indicating a correlation.
- The goal is to align a sequence to *all* members of the family at once.
- A 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 x *n* profile matrix reflecting the frequencies of nucleotides in every aligned position.
  - **Example**:

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

- Similarly, a protein family can be represented by a 20 x *n* profile representing frequencies of amino acids.

Protein Family Classification

- Multiple alignment of a protein family shows variations in conservation along the length of a protein.

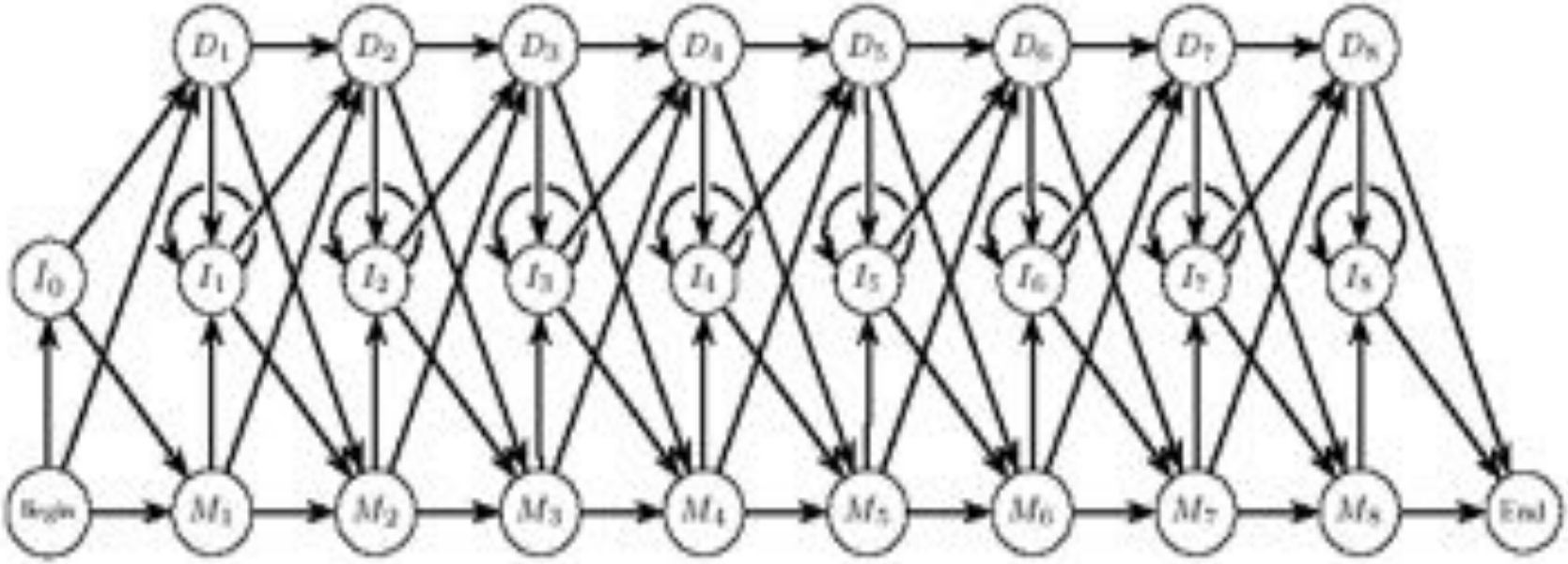
- **Example**: After aligning many globin proteins, biologists recognized that the helices region in globins are more conserved than others.

What Is a Profile HMM?

- 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 has three sets of states:
  - **Match states**: $M_1, \ldots, M_n$ (plus *begin/end* states)
  - **Insertion states**: $I_0, I_1, \ldots, I_n$
  - **Deletion states**: $D_1, \ldots, D_n$



Building a Profile HMM

1. Multiple alignment is used to construct the HMM model.

2. Assign each column to a *Match* state in HMM. Add *Insertion* and *Deletion* state.

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.

Transition Probabilities in a Profile HMM

- **Gap Initiation Penalty**: The cost of beginning a gap, which means that we must have transitions from match state to insertion state and vice versa.
  - Penalty: $\log(a_{MI}) + \log(a_{IM})$

- **Gap Extension Penalty**: The cost of extending a gap, which corresponds to maintaining the insertion state for one period.
  - Penalty: $\log(a_{II})$

Emission Probabilities in a Profile HMM

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

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

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

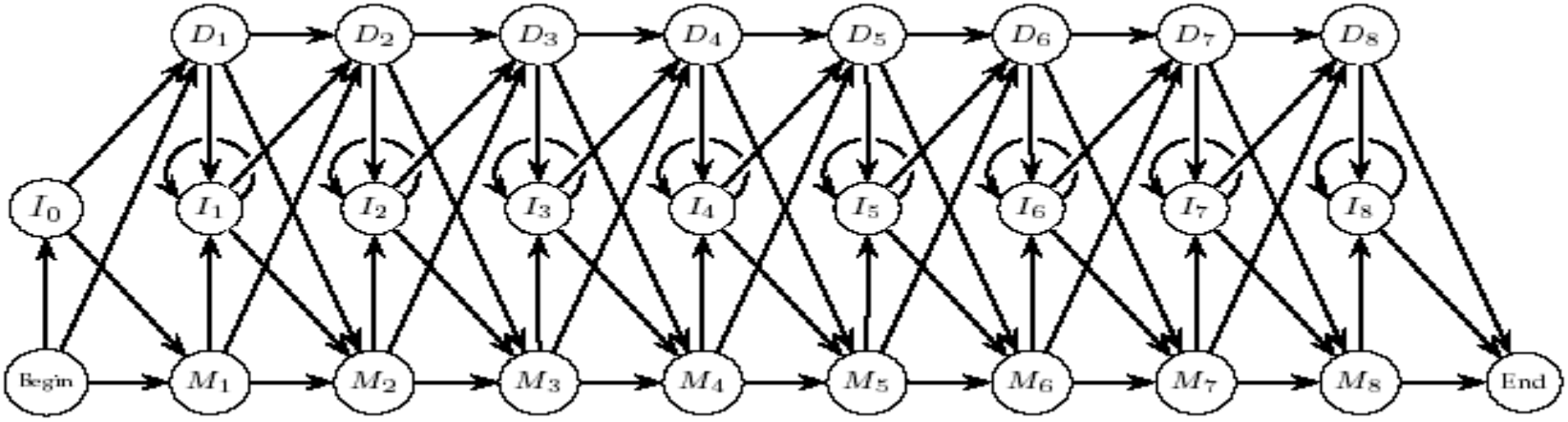
Profile HMM Alignment

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

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

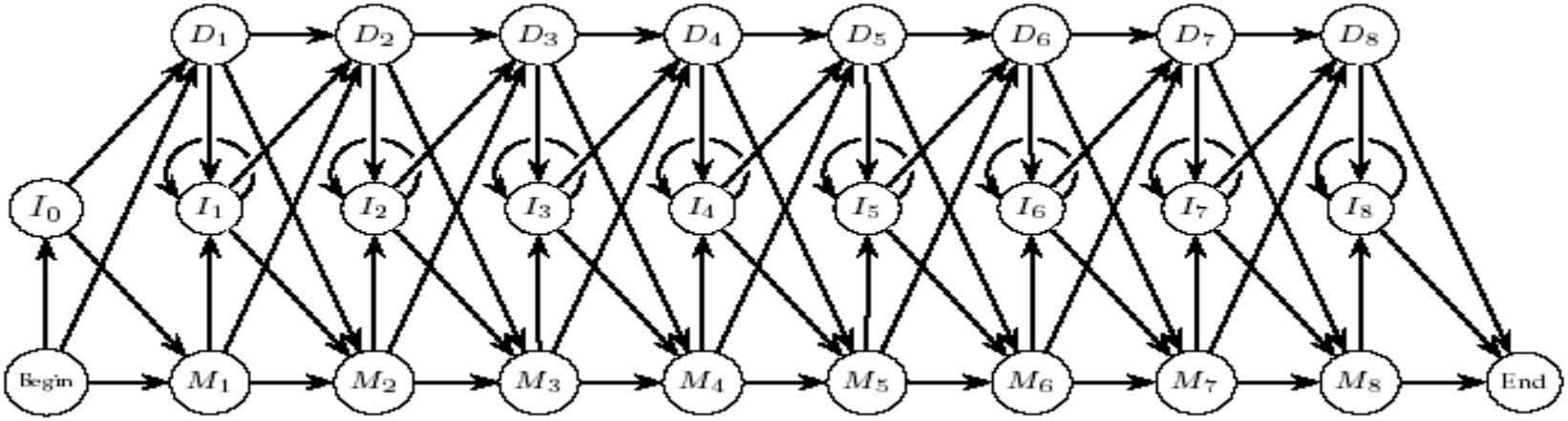
Profile HMM Alignment: Dynamic Programming

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



Profile HMM Alignment: Dynamic Programming

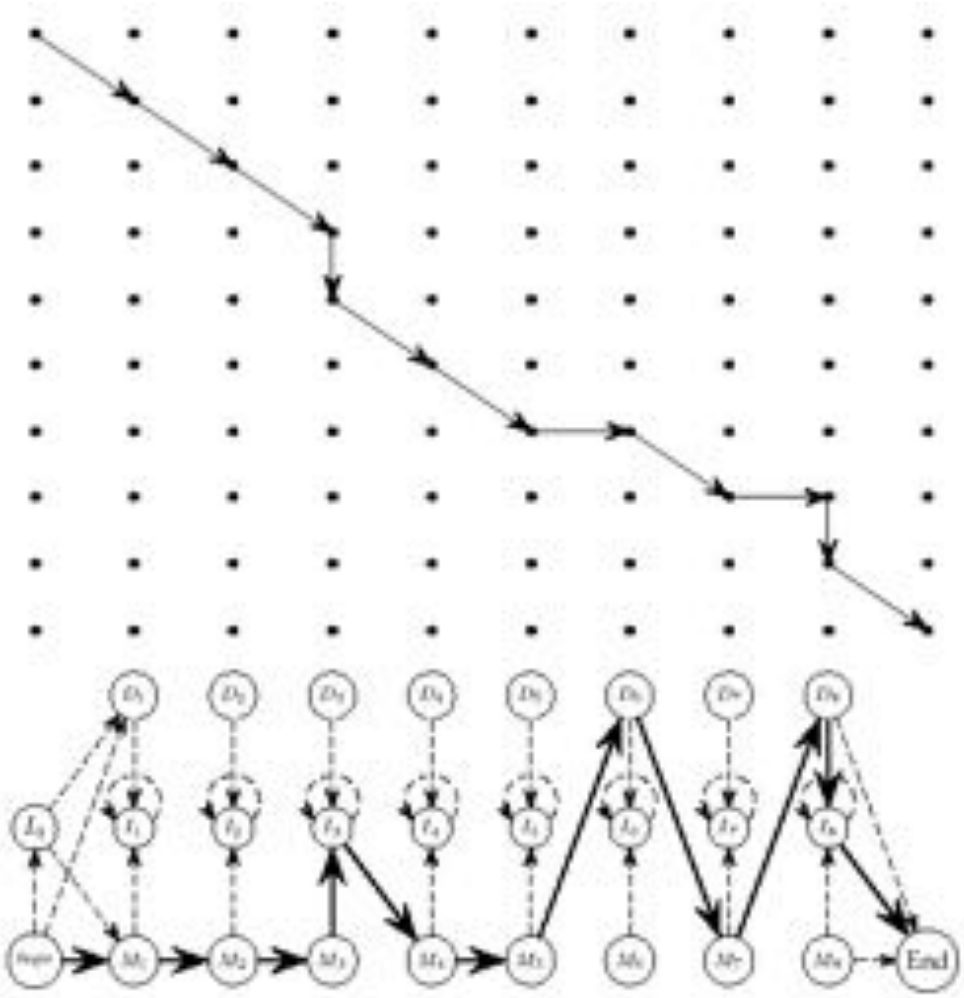
$$v_j^I(i) = \log\left[\frac{e_{I_j}(x_i)}{p(x_i)}\right] + \max\begin{cases} v_j^M(i-1) + \log\left(a_{M_j, I_j}\right) \\ v_j^I(i-1) + \log\left(a_{I_j, I_j}\right) \\ v_j^D(i-1) + \log\left(a_{D_j, I_j}\right) \end{cases}$$



Paths in Edit Graph and Profile HMM

- At right is a path through an edit graph and the corresponding path through a profile HMM.

- Observe:
  - Diagonal→match
  - Vertical→insertion
  - Horizontal→deletion



Making a Collection of HMM for Protein Families

1. Use BLAST to separate a protein database into families of related proteins.

2. Construct a multiple alignment for each protein family.

3. Construct a profile HMM model and optimize the parameters of the model (transition and emission probabilities).

4. Align the target sequence against each HMM to find the best fit between a target sequence and an HMM.

Profile HMMs and 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 HMM.
- 625 remaining globin sequences were aligned to the HMM, resulting in a multiple alignment. This multiple alignment was in a good agreement with the structurally derived alignment.
- Other 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 searches.
  - **Full alignment**: *G*enerated 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 leads to more sensitive detection.

Section 7: HMM Parameter Estimation

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 is very difficult to estimate the probabilities.

HMM Parameter Estimation Problem

- Given: HMM with states and alphabet (emission characters), as well as independent training sequences $x^1, \ldots x^m$.

- Goal: Find HMM parameters Θ (that is, $a_{k,,b}$, $e_k(b)$ that maximize the joint probability of the training sequences, which is given by the following:

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

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,

$$P\left(x^1, \ldots, x^m \mid \Theta\right) = \prod_{i=1}^{m} P\left(x^i \mid \Theta\right)$$

- 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

1. Known paths for training sequences:
   - CpG islands marked on training sequences
   - Casino analogue: One evening the dealer allows us to see when he changes the dice.

2. Unknown paths for training sequences:
   - CpG islands are not marked
   - We 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:

$$a_{k,l} = \frac{A_{k,l}}{\sum_{l'} A_{k,l'}} \qquad e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}$$

Pseudocounts

- Some state k may not appear in *any* of the training sequences. This means $A_{k,\,l} = 0$ for every state l and $a_{k,\,l}$ cannot be computed with the given equation.

- To avoid this overfitting, use predetermined **pseudocounts** r_{kl} and $r_k(b)$ which reflect prior biases about the probability values:
  - $A_{k,\,l}$ = number of transitions $k \rightarrow l + r_{k,\,l}$
  - $E_k(b)$ = number of emissions of b from $k + r_k(b)$

Section 8: Viterbi Training

Unknown Paths Method 1: Viterbi Training

- **Idea**: Use Viterbi decoding to compute the most probable path for training sequence *x*.

- **Method**:
  1. Start with some guess for initial parameters and compute π* = the most probable path for x using initial parameters.
  2. Iterate until no change in π*.
  3. Determine $A_{k,\,l}$ and $E_k(b)$ as before.
  4. Compute new parameters $a_{k,\,l}$ and $e_k(b)$ using the same formulas as before.
  5. 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 we must compare the new π^* with all previous paths having highest probability.
- Does not maximize the likelihood $\Pi_x P(x \mid \Theta)$ but rather 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 (below).

Section 9: Baum-Welch Algorithm

Unknown Paths Method 2: Baum-Welch

- Idea: Guess initial values for parameters.
  - This is art and experience, not science.
- We then estimate new (better) values for parameters.
  - How?
- We repeat until stopping criterion is met.
  - What criterion?

Improved Parameters

- We would need the $A_{k,l}$ and $E_k(b)$ values, but the path is unknown, and we do not want to use a most probable path.

- Therefore for all states k, l, symbols b, and training sequences x:
  - Compute $A_{k,l}$ and $E_k(b)$ as expected values, given the current parameters.

Probabilistic Setting for $A_{k,l}$

- Given our training sequences $x^1, \ldots, x^m$ consider a discrete probability space with elementary events $\varepsilon_{k,l,}$ = "$k \rightarrow l$ 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

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

- Define $Y = \Sigma_x \Sigma_i Y_{x,i}$ as the random variable which counts the number of times the event $\varepsilon_{k,l}$ happens in $x^1, \ldots, x^m$.

The meaning of $A_{k,l}$

- Let A_{kl} be the expectation of Y:

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

- We therefore 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 $\varepsilon_{k,b}$ = "b is emitted in state k in $x^1, \ldots, x^m$."

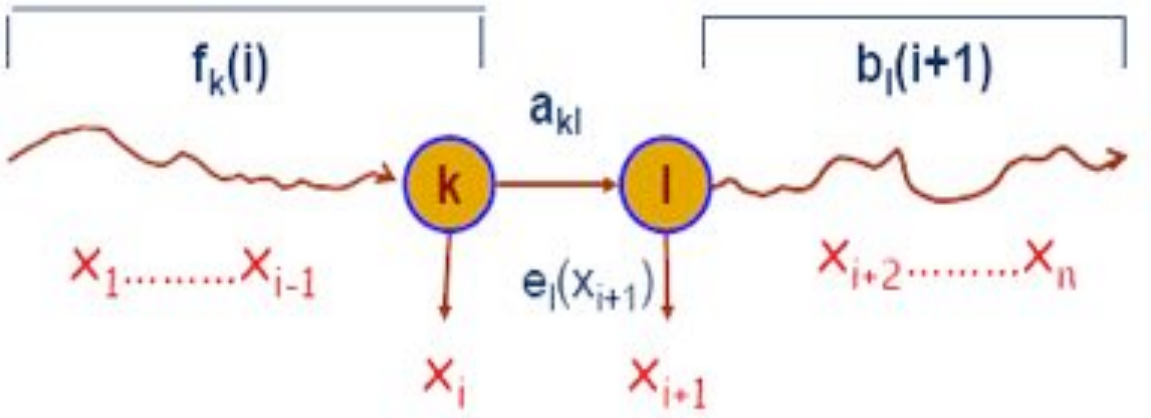
- For each x in $\{x^1, \ldots, x^m\}$ and each position i in x, let $Y_{x,i}$ be a random variable defined by

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

- Define $Y = \Sigma_x \Sigma_i Y_{x,i}$ as the random variable which counts the number of times the event $\varepsilon_{k,b}$ happens in $x^1, \ldots, x^m$.

Computing New Parameters

- Consider a training sequence $x = x^1, \ldots, x^m$.
- Concentrate on positions i and $i + 1$:



- Use the forward-backward values:

$$f_{k,i} = P\left(x_1 \cdots x_i \,\middle|\, \pi_i = k\right)$$

$$b_{k,i} = P\left(x_{i+1} \cdots x_n \,\middle|\, \pi_i = k\right)$$

Compute $A_{k,l}$ (1)

- The probability $k \rightarrow l$ is taken at position i of x:

$$P\left(\pi_i = k, \pi_{i+1} = l \,\middle|\, x_1 \cdots x_n\right) = \frac{P\left(x, \pi_i = k, \pi_{i+1} = l\right)}{P(x)}$$

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

$$P\left(x, \pi_i = k, \pi_{i+1} = l\right) = b_{l,i+1} \cdot e_l\left(x_{i+1}\right) \cdot a_{k,l} \cdot f_{k,i}$$

- Expected number of times $k \rightarrow l$ is used in training sequences:

$$A_{k,l} = \frac{\sum_x \sum_i \left(b_{l,i+1} \cdot e_l\left(x_{i+1}\right) \cdot a_{k,l} \cdot f_{k,i}\right)}{P(x)}$$

Compute $A_{kl}(2)$

$$
\begin{aligned}
P(x, \pi_i = k, \pi_{i+1} = l) &= P(x_1 \cdots x_i, \pi_i = k, \pi_{i+1} = l, x_{i+1} \cdots x_n) \\
&= P(\pi_{i+1} = l, x_{i+1} \cdots x_n \mid x_1 \cdots x_i, \pi_i = k) \cdot P(x_1 \cdots x_i, \pi_i = k) \\
&= P(\pi_{i+1} = l, x_{i+1} \cdots x_n \mid \pi_i = k) \cdot f_{k,i} \\
&= P(x_{i+1} \cdots x_n \mid \pi_i = k, \pi_{i+1} = l) \cdot P(\pi_{i+1} = l \mid \pi_i = k) \cdot f_{k,i} \\
&= P(x_{i+1} \cdots x_n \mid \pi_{i+1} = l) \cdot a_{k,l} \cdot f_{k,i} \\
&= P(x_{i+2} \cdots x_n \mid x_{i+1}, \pi_{i+1} = l) \cdot P(x_{i+1} \mid \pi_{i+1} = l) \cdot a_{k,l} \cdot f_{k,i} \\
&= P(x_{i+2} \cdots x_n \mid \pi_{i+1} = l) \cdot e_l(x_{i+1}) \cdot a_{k,l} \cdot f_{k,i} \\
&= b_{l,i+1} \cdot e_l(x_{i+1}) \cdot a_{k,l} \cdot f_{k,i}
\end{aligned}
$$

Compute $E_k(b)$

- Probability that x_i of x is emitted in state k:

$$P(\pi_i = k \mid x_1 \cdots x_n) = \frac{P(\pi_i = k, x_1 \cdots x_n)}{P(x)}$$

$$\begin{aligned} P(\pi_i = k, x_1 \cdots x_n) &= P(x_1 \cdots x_i, \pi_i = k, x_{i+1} \cdots x_n) \\ &= P(x_{i+1} \cdots x_n \mid x_1 \cdots x_i, \pi_i = k) \cdot P(x_1 \cdots x_i, \pi_i = k) \\ &= P(x_{i+1} \cdots x_n \mid \pi_i = k) \cdot f_{k,i} \\ &= b_{k,i} \cdot f_{k,i} \end{aligned}$$

- Expected number of times b is emitted in state k:

$$E_k(b) = \sum_x \sum_{i:x_i = b} \frac{f_{k,i} \cdot b_{k,i}}{P(x)}$$

Finally, new parameters

- These methods allow us to calculate our new parameters $a_{k,l}$ and $e_k(b)$:

$$a_{k,l} = \frac{A_{k,l}}{\sum_{l'} A_{k,l'}} \qquad e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}$$

- We can then add pseudocounts as before.

Stopping criteria

- We cannot actually reach maximum (property of optimization of continuous functions).

- Therefore we need stopping criteria.

- Compute the **log likelihood** of the model for current Θ :

$$\sum_{x} \log\left[P\left(x \mid \Theta\right)\right]$$

  - Compare with previous log likelihood.
  - Stop if small difference.
  - Stop after a certain number of iterations to avoid infinite loop.

The Baum-Welch Algorithm Summarized

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

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

- Repeat until log-likelihood does not change much.

Baum-Welch Analysis

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

Additional Application: 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: Fragment of speech.

- 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 matrices.
  - A_{0i}: The chance that word i begins a sentence.
  - A_{ij}: The chance that word j follows word i.

- 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 will therefore represent the most likely set of words.

Using the Model

- How well does the model 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,* or 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

- CS 262 course at Stanford given by Serafim Batzoglou