Gene finding in prokaryotes
Reading frames

- A protein is coded by groups of three nucleotides (codons):
  - ...ACGTACGTACGTACGT...
  - ...ACG-TAC-GTA-CGT-ACG-T...
  - ...TYVRT...

- There are two other ways in which this sequence can be decomposed into codons:
  - ...A-CGT-ACG-TAC-GTA-CGT...
  - ...AC-GTA-CGT-ACG-TAC-GT...

- These are the three *reading frames*

- The complementary strand has three additional reading frames
Coding for a protein

- Three non-overlapping nucleotides (codon) code for an amino acid
- Each amino acid has more than one codon that codes for it
- The code is almost universal across organisms
- Codons which code for the same amino acid are similar
- Six reading frames
Coding for a protein

• Every gene starts with the codon ATG. This specifies the reading frame and the start of translation site.
• The protein sequence continues until a stop codon (UGA, UAA, UAG) is encountered.
• DNA: [TAC CGC GGC TAT TAC TGC CAG GAA GGA ACT]r
• RNA: AUG GCG CCG AUA AUG ACG GUC CUU CCU UGA
• Protein: Met Ala Pro Ile Met Thr Val Leu Pro Stop
Open Reading Frames (ORFs)

- An open reading frame is a sequence whose length is a multiple of 3, starts with the start codon, and ends with a stop codon, with no stop codon in the middle.
- How do we determine if an ORF is a protein coding gene? Suppose we see a long run of non-stop codons after a start codon, then it has a low probability of arising by chance.
A model\(^1\) for DNA sequences

- Need a probabilistic model assigning a probability to a DNA sequence.
- Simplest model: nucleotides are independent and identically distributed.

Probability of a sequence \(s=s_1s_2...s_n\):

\[
P(s) = P(s_1)P(s_2)...P(s_n)
\]

- Distribution characterized by the numbers \((p_A, p_C, p_G, p_T)\) that satisfy: \(p_A+p_C+p_G+p_T=1\) (multinomial distribution)

---

\(^1\)“All models are wrong. Some are useful” G.E.P. Box
A method for gene finding

- Under our model, assuming that the four nucleotides have equal frequencies:
  \[ P(\text{run of } k \text{ non-stop codons}) = \left(\frac{61}{64}\right)^k \]

- Choose \( k \) such that \( P(\text{run of } k \text{ non-stop codons}) \) is smaller than the rate of false positives we are willing to accept. At the 5\% level we get \( k=62 \).
Information we ignored

- Coding regions have different nucleotide usage
- Different statistics of di-nucleotides and 3-mers in coding/non-coding regions
- Promoter region contains sequence signals for binding of TFs and RNA polymerase
- Need better models!
- But works surprisingly well in prokaryotes.
- Need more detailed models for eukaryotes (introns/exons)
Hidden Markov Models
Example: The Dishonest Casino

**Game:**
1. You bet $1
2. You roll
3. Casino player rolls
4. Highest number wins $2

The casino has two dice:

**Fair die**
\[ P(1) = P(2) = P(3) = P(5) = P(6) = \frac{1}{6} \]

**Loaded die**
\[ P(1) = P(2) = P(3) = P(5) = \frac{1}{10} \]
\[ P(6) = \frac{1}{2} \]

Casino player switches between fair and loaded die (not too often, and not for too long)
The dishonest casino model

FAIR

P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

LOADED

P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2
Question #1 – Evaluation

GIVEN:

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

QUESTION:

How likely is this sequence, given our model of how the casino works?

This is the **EVALUATION** problem in HMMs

Prob = 1.3 x 10^{-35}
Question # 2 – Decoding

GIVEN:

A sequence of rolls by the casino player

FAIR | LOADED | FAIR

QUESTION:

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

This is the **DECODING** question in HMMs
Question # 3 – Learning

GIVEN:

A sequence of rolls by the casino player

12455264621461461361366616646616366163616515615115146123562344

QUESTION:

How does the casino player work: How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?

This is the LEARNING question in HMMs
The dishonest casino model

FAIR

P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

LOADED

P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2
Definition of a hidden Markov model

- **Alphabet** \( \Sigma = \{ b_1, b_2, \ldots, b_M \} \)
- **Set of states** \( Q = \{ 1, \ldots, K \} \) \((K = |Q|)\)
- **Transition probabilities** between any two states
  \[ a_{ij} = \text{transition probability from state } i \text{ to state } j \]
  \[ a_{i1} + \ldots + a_{iK} = 1, \text{ for all states } i \]
- **Initial probabilities** \( a_{0i} \)
  \[ a_{01} + \ldots + a_{0K} = 1 \]
- **Emission probabilities** within each state
  \[ e_k(b) = P( x_i = b \mid \pi_i = k) \]
  \[ e_k(b_1) + \ldots + e_k(b_M) = 1 \]
Hidden states and observed sequence

At time step $t$,

$\pi_t$ denotes the *(hidden)* state in the Markov chain

$x_t$ denotes the symbol emitted in state $\pi_t$

A *path* of length $N$ is: $\pi_1, \pi_2, \ldots, \pi_N$

An observed *sequence* of length $N$ is: $x_1, x_2, \ldots, x_N$
A parse of a sequence

Given a sequence \( x = x_1 \ldots x_N \),

A **parse** of \( x \) is a sequence of states \( \pi = \pi_1, \ldots, \pi_N \)
Likelihood of a parse

Given a sequence \( x = x_1 \ldots x_N \) and a parse \( \pi = \pi_1, \ldots, \pi_N \),

How likely is the parse (given our HMM)?

\[
P(x, \pi) = P(x_1, \ldots, x_N, \pi_1, \ldots, \pi_N) \\
= P(x_N, \pi_N | x_1 \ldots x_{N-1}, \pi_1, \ldots, \pi_{N-1}) P(x_1 \ldots x_{N-1}, \pi_1, \ldots, \pi_{N-1}) \\
= P(x_N, \pi_N | \pi_{N-1}) P(x_1 \ldots x_{N-1}, \pi_1, \ldots, \pi_{N-1}) \\
= \ldots \\
= P(x_N, \pi_N | \pi_{N-1}) P(x_{N-1}, \pi_{N-1} | \pi_{N-2}) \ldots P(x_2, \pi_2 | \pi_1) P(x_1, \pi_1) \\
= P(x_N | \pi_N) P(\pi_N | \pi_{N-1}) \ldots P(x_2 | \pi_2) P(\pi_2 | \pi_1) P(x_1 | \pi_1) P(\pi_1) \\
= a_{0\pi_N} a_{\pi_1\pi_2} \ldots a_{\pi_{N-1}\pi_N} e_{\pi_1}(x_1) \ldots e_{\pi_N}(x_N) \\
= \prod_{i=1}^{N} a_{\pi_{i-1}\pi_i} e_{\pi_i}(x_i) \]
Example: the dishonest casino

What is the probability of a sequence of rolls

\( x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4 \)

and the parse

\( \pi = \text{Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?} \)

(say initial probs \( a_{0,\text{Fair}} = \frac{1}{2}, a_{0,\text{Loaded}} = \frac{1}{2} \))

\[
\frac{1}{2} \times P(1 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) P(2 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) \cdots P(4 \mid \text{Fair}) = \\
\frac{1}{2} \times (\frac{1}{6})^{10} \times (0.95)^9 = 5.2 \times 10^{-9}
\]
**Example: the dishonest casino**

So, the likelihood the die is fair in all this run is $5.2 \times 10^{-9}$

What about

$\pi = \text{Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?}$

$\frac{1}{2} \times P(1 \mid \text{Loaded}) \times P(\text{Loaded} \mid \text{Loaded}) \times \ldots \times P(4 \mid \text{Loaded}) =$

$\frac{1}{2} \times (1/10)^8 \times (1/2)^2 \times (0.9)^9 = 4.8 \times 10^{-10}$

Therefore, it is more likely that the die is fair all the way, than loaded all the way
Example: the dishonest casino

Let the sequence of rolls be:

\[ x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6 \]

And let’s consider \( \pi = F, F, \ldots, F \)

\[
P(x, \pi) = \frac{1}{2} \times (1/6)^{10} \times (0.95)^9 = 5.2 \times 10^{-9}
\]
(same as before)

And for \( \pi = L, L, \ldots, L: \)

\[
P(x, \pi) = \frac{1}{2} \times (1/10)^4 \times (1/2)^6 \times (0.9)^9 = 3.02 \times 10^{-7}
\]

So, the observed sequence is \( \sim 100 \) times more likely if a loaded die is used
What we know

Given a sequence \( x = x_1, \ldots, x_N \) and a parse \( \pi = \pi_1, \ldots, \pi_N \), we know how to compute

\[ P(x, \pi) \]
What we would like to know

1. Evaluation
   GIVEN HMM \( M \), and a sequence \( x \),
   FIND \( \text{Prob}[ x | M ] \)

2. Decoding
   GIVEN HMM \( M \), and a sequence \( x \),
   FIND the sequence \( \pi \) of states that maximizes \( \text{P}[ x, \pi | M ] \)

3. Learning
   GIVEN HMM \( M \), with unspecified transition/emission probs.,
   and a sequence \( x \),
   FIND parameters \( \theta = (e_i(\cdot), a_{ij}) \) that maximize \( \text{P}[ x | \theta ] \)
Problem 2: Decoding

Find the best parse of a sequence
Decoding

GIVEN $x = x_1 x_2, \ldots, x_N$

We want to find $\pi = \pi_1, \ldots, \pi_N$, such that $P[ x, \pi ]$ is maximized

$$\pi^* = \text{argmax}_\pi P[ x, \pi ]$$

Maximize $a_{0\pi_1} e_{\pi_1}(x_1) a_{\pi_1\pi_2} \ldots a_{\pi_{N-1}\pi_N} e_{\pi_N}(x_N)$

We can use dynamic programming!

Let $V_k(i) = \max_{\{\pi_1, \ldots, \pi_{i-1}\}} P[ x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$  

= Probability of maximum probability path ending at state $\pi_i = k$
Decoding – main idea

**Inductive assumption:** Given

\[ V_k(i) = \max_{\{\pi_1, \ldots, \pi_{i-1}\}} P[x_1\ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k] \]

What is \( V_r(i+1) \)?

\[ V_r(i+1) = \max_{\{\pi_1, \ldots, \pi_i\}} P[ x_1\ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = r ] \]

\[ = \max_{\{\pi_1, \ldots, \pi_i\}} P(x_{i+1}, \pi_{i+1} = r \mid x_1\ldots x_i, \pi_1, \ldots, \pi_i) P[x_1\ldots x_{i+1}, \pi_1, \ldots, \pi_i] \]

\[ = \max_{\{\pi_1, \ldots, \pi_i\}} P(x_{i+1}, \pi_{i+1} = r \mid \pi_i) P[x_1\ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i] \]

\[ = \max_k \left[ P(x_{i+1}, \pi_{i+1} = r \mid \pi_i = k) \max_{\{\pi_1, \ldots, \pi_{i-1}\}} P[x_1\ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k] \right] \]

\[ = \max_k e_r(x_{i+1}) a_{kr} V_k(i) \]

\[ = e_r(x_{i+1}) \max_k a_{kr} V_k(i) \]
The Viterbi Algorithm

Input: $x = x_1, \ldots, x_N$

Initialization: 
$V_0(0) = 1$  
(0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

Iteration: 
$V_j(i) = e_j(x_i) \times \max_k a_{kj} \ V_k(i-1)$

$P_{tr}(i) = \arg\max_k a_{kj} \ V_k(i-1)$

Termination: 
$P(x, \pi^*) = \max_k V_k(N)$

Traceback: 
$\pi_N^* = \arg\max_k V_k(N)$

$\pi_{i-1}^* = P_{tr_{\pi_i}}(i)$
The Viterbi Algorithm

Input: \( x = x_1, \ldots, x_N \)

**Initialization:**

\[ V_0(0) = 1 \]  
(0 is the imaginary first position)

\[ V_k(0) = 0, \text{ for all } k > 0 \]

**Iteration:**

\[ V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1) \]
\[ \text{Ptr}_j(i) = \arg\max_k a_{kj} V_k(i-1) \]

**Termination:**

\[ P(x, \pi^*) = \max_k a_{k0} V_k(N) \text{ (with an end state)} \]

**Traceback:**

\[ \pi_N^* = \arg\max_k V_k(N) \]
\[ \pi_{i-1}^* = \text{Ptr}_{\pi_i}(i) \]
The Viterbi Algorithm

Similar to "aligning" a set of states to a sequence
The Viterbi Algorithm

Similar to "aligning" a set of states to a sequence

Time:

\[ O(K^2N) \]

Space:

\[ O(KN) \]
Viterbi Algorithm – a practical detail

Underflows are a significant problem

\[ P [ x_1, \ldots, x_i, \pi_1, \ldots, \pi_i ] = a_{0\pi_1} a_{\pi_1\pi_2} \ldots a_{\pi_i} e_{\pi_1}(x_1) \ldots e_{\pi_i}(x_i) \]

These numbers become extremely small – underflow

**Solution:** Take the logs of all values

\[ V_r(i) = \log e_k(x_i) + \max_k [ V_k(i-1) + \log a_{kr} ] \]
Example

Let $x$ be a sequence with a portion that has 6’s with probability $1/6$, followed by a portion with 6’s with fraction $1/2$

$$x = 123456123456...12345 \ 6626364656...1626364656$$

Easy to convince yourself that the optimal parse is:

```
FFF..........................F  LLL..........................L
```
Example

Observed Sequence: \( x = 1,2,1,6,6 \) \quad P(x) = 0.0002195337

Best 8 paths:

- LLLLL 0.0001018
- FFFFF 0.0000524
- FFFLL 0.0000248
- FFLLL 0.0000149
- FLLLL 0.0000089
- FFFFL 0.0000083
- LLLLLF 0.0000018
- LFFFFF 0.0000017
Problem 1: Evaluation

Finding the probability a sequence is generated by the model
Generating a sequence by the model

Given a HMM, we can generate a sequence of length $n$ as follows:

1. Start at state $\pi_1$ according to probability $a_{0\pi_1}$
2. Emit letter $x_1$ according to probability $e_{\pi_1}(x_1)$
3. Go to state $\pi_2$ according to probability $a_{\pi_1\pi_2}$
4. ... until emitting $x_n$
The Forward Algorithm

want to calculate

\[ P(x) = \text{probability of } x, \text{ given the HMM } (X = x_1, ..., x_N) \]

Sum over all possible ways of generating \( x \):

\[ P(x) = \sum_{\text{all paths } \pi} P(x, \pi) \]

To avoid summing over an exponential number of paths \( \pi \), define

\[ f_k(i) = P(x_1, ..., x_i, \pi_i = k) \] (the forward probability)
The Forward Algorithm – derivation

the forward probability:

\[ f_k(i) = P(x_1...x_i, \pi_i = k) \]

\[ = \sum_{\pi_1...\pi_{i-1}} P(x_1...x_{i-1}, \pi_1,..., \pi_{i-1}, \pi_i = k) e_k(x_i) \]

\[ = \sum_r \sum_{\pi_1...\pi_{i-2}} P(x_1...x_{i-1}, \pi_1,..., \pi_{i-2}, \pi_{i-1} = r) a_{rk} e_k(x_i) \]

\[ = \sum_r P(x_1...x_{i-1}, \pi_{i-1} = r) a_{rk} e_k(x_i) \]

\[ = e_k(x_i) \sum_r f_r(i-1) a_{rk} \]
The Forward Algorithm

A dynamic programming algorithm:

**Initialization:**
\[
f_0(0) = 1; \\
f_k(0) = 0, \text{ for all } k > 0
\]

**Iteration:**
\[
f_k(i) = e_k(x_i) \sum_r f_r(i-1) a_{rk}
\]

**Termination:**
\[
P(x) = \sum_k f_k(N)
\]
The Forward Algorithm

If our model has an “end” state:

Initialization:
\[ f_0(0) = 1 \; ; \]
\[ f_k(0) = 0, \text{ for all } k > 0 \]

Iteration:
\[ f_k(i) = e_k(x_i) \sum_r f_r(i-1) a_{rk} \]

Termination:
\[ P(x) = \sum_k f_k(N) a_{k0} \]

Where, \( a_{k0} \) is the probability that the terminating state is \( k \) (usually = \( a_{0k} \))
Relation between Forward and Viterbi

**VITERBI**

**Initialization:**
- \( V_0(0) = 1 \)
- \( V_k(0) = 0, \text{ for all } k > 0 \)

**Iteration:**
- \( V_j(i) = e_j(x_i) \max_k V_k(i-1) a_{kj} \)

**Termination:**
- \( P(x, \pi^*) = \max_k V_k(N) \)

**FORWARD**

**Initialization:**
- \( f_0(0) = 1 \)
- \( f_k(0) = 0, \text{ for all } k > 0 \)

**Iteration:**
- \( f_l(i) = e_l(x_i) \sum_k f_k(i-1) a_{kl} \)

**Termination:**
- \( P(x) = \sum_k f_k(N) \)
The most likely state

Given a sequence $x$, what is the most likely state that emitted $x_i$?

In other words, we want to compute $P(\pi_i = k \mid x)$

Example: the dishonest casino

Say $x = 12341623162616364616234161221341$

Most likely path: $\pi = FF\ldots F$

However: marked letters more likely to be L
Motivation for the Backward Algorithm

We want to compute \( P(\pi_i = k \mid x) \),

We start by computing

\[
P(\pi_i = k, x) = P(x_1\ldots x_i, \pi_i = k, x_{i+1}\ldots x_N)
= P(x_1\ldots x_i, \pi_i = k) \cdot P(x_{i+1}\ldots x_N \mid x_1\ldots x_i, \pi_i = k)
= P(x_1\ldots x_i, \pi_i = k) \cdot P(x_{i+1}\ldots x_N \mid \pi_i = k)
\]

Forward, \( f_k(i) \) \hspace{1cm} Backward, \( b_k(i) \)

Then, \( P(\pi_i = k \mid x) = P(\pi_i = k, x) / P(x) = f_k(i) b_k(i) / P(x) \)
The Backward Algorithm – derivation

Define the backward probability:

\[ b_k(i) = P(x_{i+1}...x_N \mid \pi_i = k) \]

\[ = \sum_{\pi_{i+1}...\pi_N} P(x_{i+1}, x_{i+2}, ..., x_N, \pi_{i+1}, ..., \pi_N \mid \pi_i = k) \]

\[ = \sum_r \sum_{\pi_{i+2}...\pi_N} P(x_{i+1}, x_{i+2}, ..., x_N, \pi_{i+1} = r, \pi_{i+2}, ..., \pi_N \mid \pi_i = k) \]

\[ = \sum_r e_r(x_{i+1}) a_{kr} \sum_{\pi_{i+2}...\pi_N} P(x_{i+2}, ..., x_N, \pi_{i+2}, ..., \pi_N \mid \pi_{i+1} = r) \]

\[ = \sum_r e_r(x_{i+1}) a_{kr} b_r(i+1) \]
The Backward Algorithm

A dynamic programming algorithm for $b_k(i)$:

**Initialization:**

$$b_k(N) = 1, \text{ for all } k$$

**Iteration:**

$$b_k(i) = \sum_r e_r(x_{i+1}) a_{kr} b_r(i+1)$$

**Termination:**

$$P(x) = \sum_r a_{0r} e_r(x_1) b_r(1)$$
The Backward Algorithm

In case of an “end” state:

**Initialization:**

\[ b_k(N) = a_{k0}, \text{ for all } k \]

**Iteration:**

\[ b_k(i) = \sum_r e_r(x_{i+1}) a_{kr} b_r(i+1) \]

**Termination:**

\[ P(x) = \sum_r a_{0r} e_r(x_1) b_r(1) \]
Example

Observed Sequence: $x = 1,2,1,6,6$ \hspace{1cm} P(x) = 0.0002195337

**Conditional Probabilities given x**

<table>
<thead>
<tr>
<th>Position</th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5029927</td>
<td>0.4970073</td>
</tr>
<tr>
<td>2</td>
<td>0.4752192</td>
<td>0.5247808</td>
</tr>
<tr>
<td>3</td>
<td>0.4116375</td>
<td>0.5883625</td>
</tr>
<tr>
<td>4</td>
<td>0.2945388</td>
<td>0.7054612</td>
</tr>
<tr>
<td>5</td>
<td>0.2665376</td>
<td>0.7334624</td>
</tr>
</tbody>
</table>
Computational Complexity

What is the running time, and space required for Forward, and Backward algorithms?

Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows
- Rescaling at each position by multiplying by a constant
Scaling

- Define:
  \[ \tilde{f}_r(i) = \frac{f_r(i)}{\prod_{j=1}^{i} s_j} \quad \tilde{b}_r(i) = \frac{b_r(i)}{\prod_{j=i+1}^{N} s_j} \]

- Recursion:
  \[ \tilde{f}_r(i + 1) = \frac{1}{s_{i+1}} e_r(x_{i+1}) \sum_k \tilde{f}_k(i) a_{kr} \]

- Choosing scaling factors such that
  \[ \sum_r \tilde{f}_r(i) = 1 \]

Lead to:

\[ s_{i+1} = \sum_r e_r(x_{i+1}) \sum_k \tilde{f}_k(i) a_{kr} \]

Scaling for the backward probabilities:

\[ \tilde{b}_k(i) = \frac{1}{s_i} \sum_r e_r(x_{i+1}) a_{kr} \tilde{b}_r(i + 1) \]
Posterior Decoding

We can now calculate

\[ P(\pi_i = k \mid x) = \frac{f_k(i) \ b_k(i)}{P(x)} \]

we can now ask

What is the most likely state at position \( i \) of sequence \( x \) ?

Using **Posterior Decoding** we can now define:

\[ \hat{\pi}_i = \arg\max_k P(\pi_i = k \mid x) \]
Posterior Decoding

• For each state,

  • Posterior Decoding gives us a curve of probability of state for each position, given the sequence $x$

  • That is sometimes more informative than Viterbi path $\pi^*$

• Posterior Decoding may give an invalid sequence of states

• Why?
Viterbi vs. posterior decoding

- A class takes a multiple choice test.
- How does the lazy professor construct the answer key?
- Viterbi approach: use the answers of the best student
- Posterior decoding: majority vote
Viterbi, Forward, Backward

**VITERBI**

Initialization:

\[ V_0(0) = 1 \]
\[ V_k(0) = 0, \text{ for all } k > 0 \]

Iteration:

\[ V_r(i) = e_r(x_i) \max_k V_k(i-1) a_{kr} \]

Termination:

\[ P(x, \pi^*) = \max_k V_k(N) \]

**FORWARD**

Initialization:

\[ f_0(0) = 1 \]
\[ f_k(0) = 0, \text{ for all } k > 0 \]

Iteration:

\[ f_r(i) = e_r(x_i) \sum_k f_k(i-1) a_{kr} \]

Termination:

\[ P(x) = \sum_k f_k(N) a_{k0} \]

**BACKWARD**

Initialization:

\[ b_k(N) = a_{k0}, \text{ for all } k \]

Iteration:

\[ b_r(i) = \sum_k e_k(x_i+1) a_{rk} b_k(i+1) \]

Termination:

\[ P(x) = \sum_k a_{0k} e_k(x_1) b_k(1) \]
A modeling Example

CpG islands in DNA sequences
Methylation & Silencing

- Methylation
- Addition of CH₃ in C-nucleotides
- Silences genes in region

- CG (denoted CpG) often mutates to TG, when methylated

- Methylation is inherited during cell division
CpG Islands

CpG nucleotides in the genome are frequently methylated

\[ C \rightarrow \text{methyl-C} \rightarrow T \]

Methylation often suppressed around genes, promoters

\[ \rightarrow \text{CpG islands} \]
CpG Islands

- In CpG islands:
  - CG is more frequent
  - Other dinucleotides (AA, AG, AT...) have different frequencies

**Problem:** Detect CpG islands
A model of CpG Islands – Architecture
A model of CpG Islands – Transitions

How do we estimate parameters of the model?

**Emission probabilities:** 1/0

1. Transition probabilities within CpG islands
   Established from known **CpG islands**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.180</td>
<td>.274</td>
<td>.426</td>
<td>.120</td>
</tr>
<tr>
<td>C</td>
<td>.171</td>
<td>.368</td>
<td>.274</td>
<td>.188</td>
</tr>
<tr>
<td>G</td>
<td>.161</td>
<td>.339</td>
<td>.375</td>
<td>.125</td>
</tr>
<tr>
<td>T</td>
<td>.079</td>
<td>.355</td>
<td>.384</td>
<td>.182</td>
</tr>
</tbody>
</table>

2. Transition probabilities within non-CpG islands
   Established from **non-CpG islands**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.300</td>
<td>.205</td>
<td>.285</td>
<td>.210</td>
</tr>
<tr>
<td>C</td>
<td>.233</td>
<td>.298</td>
<td>.078</td>
<td>.302</td>
</tr>
<tr>
<td>G</td>
<td>.248</td>
<td>.246</td>
<td>.298</td>
<td>.208</td>
</tr>
<tr>
<td>T</td>
<td>.177</td>
<td>.239</td>
<td>.292</td>
<td>.292</td>
</tr>
</tbody>
</table>
A model of CpG Islands – Transitions

- What about transitions between (+) and (-) states?
- Their probabilities affect
  - Avg. length of CpG island
  - Avg. separation between two CpG islands

Length distribution of X region:

\[ P[L_X = 1] = 1-p \]
\[ P[L_X = 2] = p(1-p) \]
\[ \ldots \]
\[ P[L_X = k] = p^{k-1}(1-p) \]

\[ E[L_X] = \frac{1}{1-p} \]
Geometric distribution, with mean \( \frac{1}{1-p} \)
A model of CpG Islands – Transitions

No reason to favor exiting/entering (+) and (-) regions at a particular nucleotide

- Estimate average length $L_{\text{CPG}}$ of a CpG island:
  \[ L_{\text{CPG}} = \frac{1}{1-p} \Rightarrow p = 1 - \frac{1}{L_{\text{CPG}}} \]

- For each pair of (+) states: $a_{kr} \leftarrow p \times a_{kr}^+$
- For each (+) state $k$, (-) state $r$:
  \[ a_{kr} = (1-p)(a_{0r^-}) \]
- Do the same for (-) states

A problem with this model:
CpG islands don’t have a geometric length distribution

This is a defect of HMMs – a price we pay for ease of analysis & efficient computation
Using the model

Given a DNA sequence $x$,

The Viterbi algorithm predicts locations of CpG islands

Given a nucleotide $x_i$, (say $x_i = A$)

The Viterbi parse tells whether $x_i$ is in a CpG island in the most likely parse

Using the Forward/Backward algorithms we can calculate

$$P(x_i \text{ is in CpG island}) = P(\pi_i = A^+ \mid x)$$

Posterior Decoding can assign locally optimal predictions of CpG islands

$$\hat{\pi}_i = \arg\max_k P(\pi_i = k \mid x)$$
Posterior decoding

Results of applying posterior decoding to a part of human chromosome 22
Posterior decoding
Viterbi decoding

![Viterbi decoding diagram](image-url)
Sliding window (size = 100)
Sliding widow (size = 200)
Sliding window (size = 300)
Sliding window (size = 400)
Sliding window (size = 600)
Sliding window (size = 1000)
Modeling CpG islands with silent states
What if a new genome comes?

- Suppose we just sequenced the porcupine genome.
- We know CpG islands play the same role in this genome.
- However, we have no known CpG islands for porcupines.
- We suspect the frequency and characteristics of CpG islands are quite different in porcupines.

How do we adjust the parameters in our model?
Two learning scenarios

- **Estimation when the “right answer” is known**
  
  **Examples:**
  
  **GIVEN:** a genomic region $x = x_1...x_N$ where we have good (experimental) annotations of the CpG islands
  
  **GIVEN:** the casino player allows us to observe him one evening as he changes the dice and produces 10,000 rolls

- **Estimation when the “right answer” is unknown**
  
  **Examples:**
  
  **GIVEN:** the porcupine genome; we don’t know how frequent are the CpG islands there, neither do we know their composition
  
  **GIVEN:** 10,000 rolls of the casino player, but we don’t see when he changes dice

**GOAL:** Update the parameters $\theta$ of the model to maximize $P(x|\theta)$
When the right answer is known

Given $x = x_1 ... x_N$

for which $\pi = \pi_1 ... \pi_N$ is known,

Define:

$A_{kr} = \text{# of times } k \rightarrow r \text{ transition occurs in } \pi$

$E_k(b) = \text{# of times state } k \text{ in } \pi \text{ emits } b \text{ in } x$

The maximum likelihood estimates of the parameters are:

$$a_{kr} = \frac{A_{kr}}{\sum_i A_{ki}}$$

$$e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}$$
When the right answer is known

**Intuition:** When we know the underlying states, the best estimate is the average frequency of transitions & emissions that occur in the training data.

**Drawback:**
Given little data, there may be **overfitting**: $P(x|\theta)$ is maximized, but $\theta$ is unreasonable
0 probabilities – VERY BAD

**Example:**
Suppose we observe:
\[ x = 2 \ 1 \ 5 \ 6 \ 1 \ 2 \ 3 \ 6 \ 2 \ 3 \ 5 \ 3 \ 4 \ 2 \ 1 \ 2 \ 1 \ 6 \ 3 \ 6 \]
\[ \pi = \text{F \ F \ F \ F \ F \ F \ F \ F \ F \ F \ F \ F \ L \ L \ L \ L \ L} \]
Then:
\[ a_{FF} = \frac{14}{15} \quad a_{FL} = \frac{1}{15} \quad a_{LL} = 1 \quad a_{LF} = 0 \]
\[ e_F(4) = 0; \ e_L(4) = 0 \]
Pseudocounts

Solution for small training sets:

Add pseudocounts

\[ A_{kr} = \# \text{ times } k \rightarrow r \text{ state transition occurs} + t_{kr} \]
\[ E_k(b) = \# \text{ times state } k \text{ emits symbol } b \text{ in } x + t_k(b) \]

\( t_{kr}, t_k(b) \) are pseudocounts representing our prior belief

Larger pseudocounts \( \Rightarrow \) Strong prior belief

Small pseudocounts \( (\varepsilon < 1) \): just to avoid 0 probabilities
Pseudocounts

\[ a_{kr} = \frac{A_{kr} + t_{kr}}{\sum_i A_{ki} + t_{kr}} \]

\[ e_k(b) = \frac{E_k(b) + t_k(b)}{\sum_c E_k(c) + t_k(b)} \]
**Pseudocounts**

**Example:** dishonest casino

We will observe player for one day, 600 rolls

Reasonable pseudocounts:

\[
\begin{align*}
t_{OF} &= t_{OL} = t_{FO} = t_{LO} = 1; \\
t_{FL} &= t_{LF} = t_{FF} = t_{LL} = 1; \\
t_F(1) &= t_F(2) = \ldots = t_F(6) = 20 \text{ (strong belief fair is fair)} \\
t_L(1) &= t_L(2) = \ldots = t_L(6) = 5 \text{ (wait and see for loaded)}
\end{align*}
\]

Above numbers arbitrary – assigning priors is an art
When the right answer is unknown
We don’t know the actual $A_{kr}$, $E_k(b)$

Idea:

- Initialize the model
- Compute $A_{kr}$, $E_k(b)$
- Update the parameters of the model, based on $A_{kr}$, $E_k(b)$
- Repeat until convergence

Two algorithms: Baum-Welch, Viterbi training
Gene finding with HMMs
E. Coli Gene HMM (A Simple Version)
Example: genes in prokaryotes

- EasyGene: a prokaryotic gene-finder (Larsen TS, Krogh A)

- Codons are modeled with 3 looped triplets of states (this converts the length distribution to negative binomial)
Using the gene finder

• Apply the gene finder to both strands

• Training using annotated genes
A simple eukaryotic gene finder