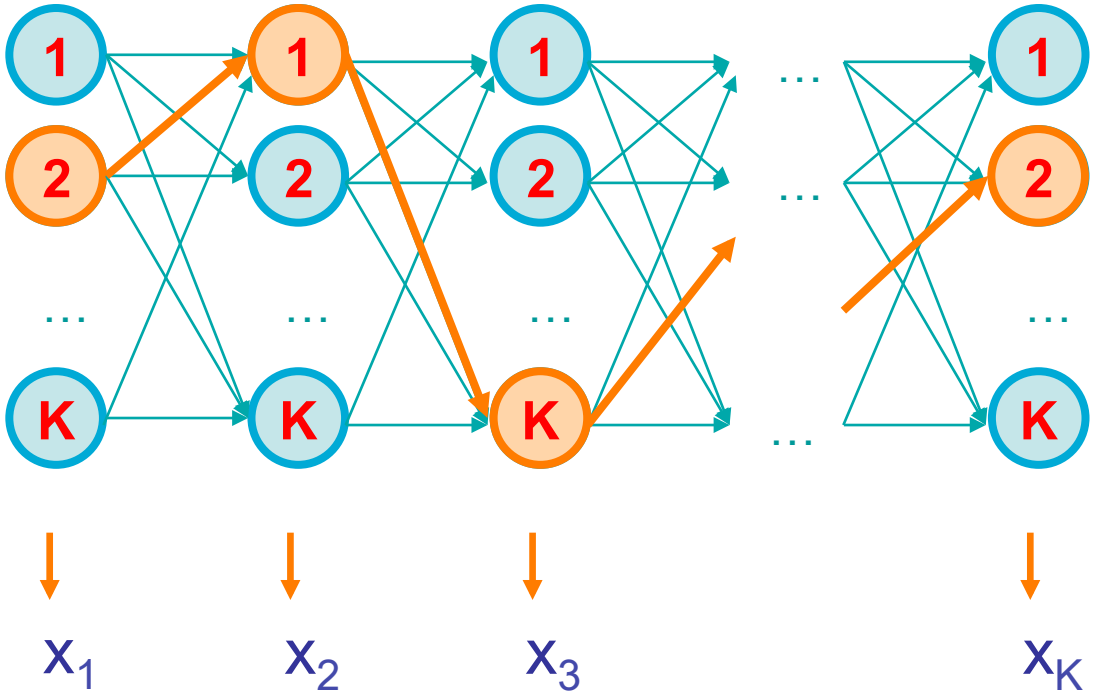
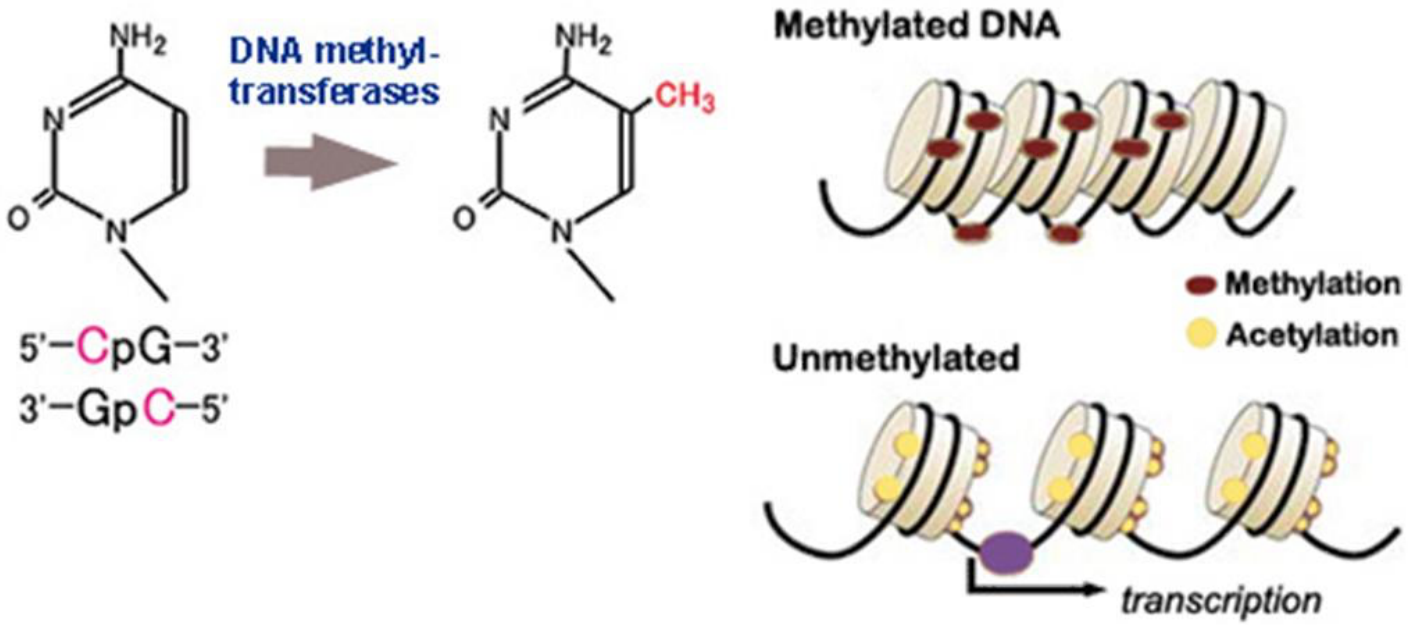
Hidden Markov Models



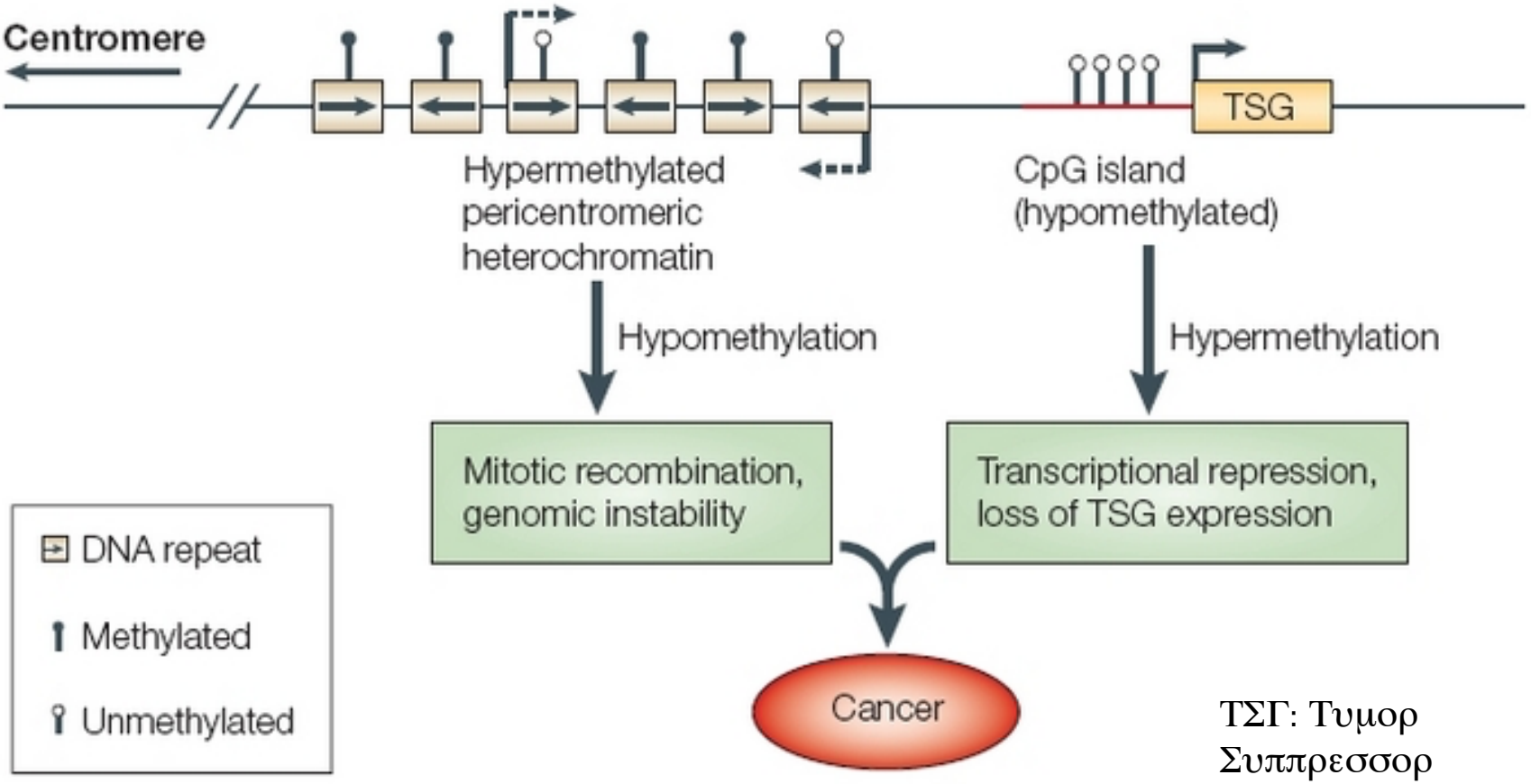
Motivating Example

- CpG sites (-cytosine-phosphate-guanine-)
  - C in CpG can be methylated -> 5-methylcytosine
  - 70%-80% of CpG cytonsines are methylated



http://helicase.pbworks.com/w/page/17605615/DNA%20Methylation

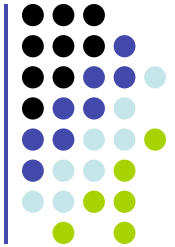
Motivating Example



ΤΣΓ: Τυμορ Συππρεσσορ Γενε

http://www.nature.com/scitable/topicpage/the-role-of-methylation-in-gene-expression-1070

Motivating Example



CpG islands: regions with a high frequency of CpG sites

near gene-promoter

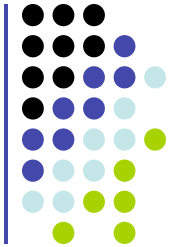
CATTCCGCCTTCTCTCCCGAGGTGGCGCGTGGGA
GGTGTTTTGCTCGGGTTCTGTAAGAATAGGCCAGG
CAGCTTCCCGCGGGATGCGCTCATCCCCTCTCGG
GGTTCCGCTCCCACCGCGCCGCGTTCGGCCGGTT
CCGCCTGCGAGATGTTTTCCGACGGACAATGATTC
CACTCTCGGCGCCTCCCATGTTGATCCCAGCTCCT
CTGCGGGCGTCAGGACCCCTGGGCCCCGCCCCG
CTCCACTCAGTCAATCTTTTGTCCCCGTATAAGGCG
GATTATCGGGGTGGCTGGGGGCGGCTGATTCCGA
CGAATGCCCTTGGGGGTCACCCGGGAGGGAACTC
CGGGCTCCGGCTTTGGCCAGCCCGCACCCCTGGT
TGAGCCGGCCCGAGGGCCACCAGGGGGCGCTCG
ATGTTCCTGCAGCCCCCCGCAGCAGCCCCACTCC
CCGGCTCACCCTACGATTGGCTGGCCGCCCCGAG
CTCTGTGCTGTGATTGGTCACAGCCCGTGTCCGTC
GCGGGCGCCGGGCGGATACGAGGTGACGCGCA
GAGGCCCAGCTCGGGCGGTGTCCCGCGCCGGC
GACTGCGGGCGGAGTTTCGCGAGGGCCGAAGCG
GGGCAGTGTGACGGCAGCGGTCCTGGGAGGCGC
CCGCGCGCGTCGGAGCAGCTCCCCGTCCTCCGCA
GCCGTCACCGCCGGCCGTCGCCGCGCCCTGGCC
TCCCGCACTCGCGCACTCCTGTCCGCCGCCCACC
GCCCACCTCCCACCTCGATGCGGTGCCGGCTGC
TGCGTGATGGGGCTGCGGAGCGGCGCCCTGCGG
CTCGCGGCGGCCGCTGCTCGCGCTGAGGTGCGT
CGGTGCCCGGCCCCCCGCGCCCCGCGCGCCGCC
GGCTCCTGTTGACCCGGTCCGCCCGTCGGTCTGC
AGCGCGGCTGAGGTAAGGCGGCGGGCTGGCCG
CGGTTGGCGCCGCGGTCGCGGGGTTGGGGAGGG
GGCCGCTTCCGCGGGAGGAGCGGCCGGCCGG
GGTCCGGGCGGGGTCTGAGGGGA

normal example of the genome

CTCTTAGTTTTGGGTGCATTTGTCTGGTCTTCCAAA
CTAGATTGAAAGCTCTGAAAAAAAAAACTATCTTGT
GTTTCTATCTGTTGAGCTCATAGTAGGTATCCAGGA
AGTAGTAGGGTTGACTGCATTGATTTGGGACTACAC
TGGGAGTTTTCTTCGCCATCTCCCTTTAGTTTTCCT
TTTTTTCTTTCTTTCTTTTCTTTTTTTTCTTTTTTTTT
TTGAGATGTCGTCTTGCTCAGTCCCCCAGGCTGGA
GTGCAGTGGTGCGATCTTGGCTCACTGTAGCCTCC
ACCTCCCAGGTTCAAGCAATTCTACTGCCTTAGCCT
CCCGAGTAGCTGGGATTACAAGCACCCGCCACCAT
TCCTGGCTAATTTTTTTTTTTGTATTTTTAGTTGAGA
CAGGGTTTCACCATGTTGGTGATGCTGGTCTCAGA
CTCCTGGGGCCTAGCGATCCCCCTGCCTCAGCCT
CCCAGAGTGTTAGGATTACAGGCATGAGCCACTGT
ACCCGGCCTCTCTCCAGTTTCCAGTTGGAATCCAA
GGGAAGTAAGTTTAAGATAAAGTTACGATTTTGAAAT
CTTTGGATTCAGAAGAATTTGTCACCTTTAACACCT
AGAGTTGAACGTTCATACCTGGAGAGCCTTAACATT
AAGCCCTAGCCAGCCTCCAGCAAGTGGACATTGGT
CAGGTTTGGCAGGATTCGTCCCCTGAAGTGGACT
GAGAGCCACACCCTGGCCTGTCACCATACCCATCC
CCTATCCTTAGTGAAGCAAAACTCCTTTGTTCCCTT
CTCCTTCTCCTAGTGACAGGAAATATTGTGATCCTA
AAGAATGAAAATAGCTTGTCACCTCGTGGCCTCAG
GCCTCTTGACTTCAGGCGGTTCTGTTTAATCAAGT
GACATCTTCCCGAGGCTCCCTGAATGTGGCAGATG
AAAGAGACTAGTTCAACCCTGACCTGAGGGGAAAG
CCTTTGTGAAGGGTCAGGAG

https://commons.wikimedia.org/wiki/File:Cpg_islands.svg

The three main questions on HMMs



1. **Decoding**

   GIVEN a HMM M, and a sequence x,
   FIND the sequence π of states that maximizes P[x, π | M]

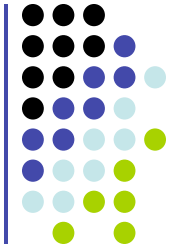
2. **Evaluation**

   GIVEN a HMM M, and a sequence x,
   FIND Prob[x | M]

3. **Learning**

   GIVEN a HMM M, with unspecified transition/emission probs.,
   and a sequence x,

   FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize P[x | θ]



Problem 1: Decoding

Find the most likely parse of a sequence

Decoding - Review

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

Find $\pi = \pi_1, \ldots\ldots, \pi_N,$
to maximize $P[x, \pi]$

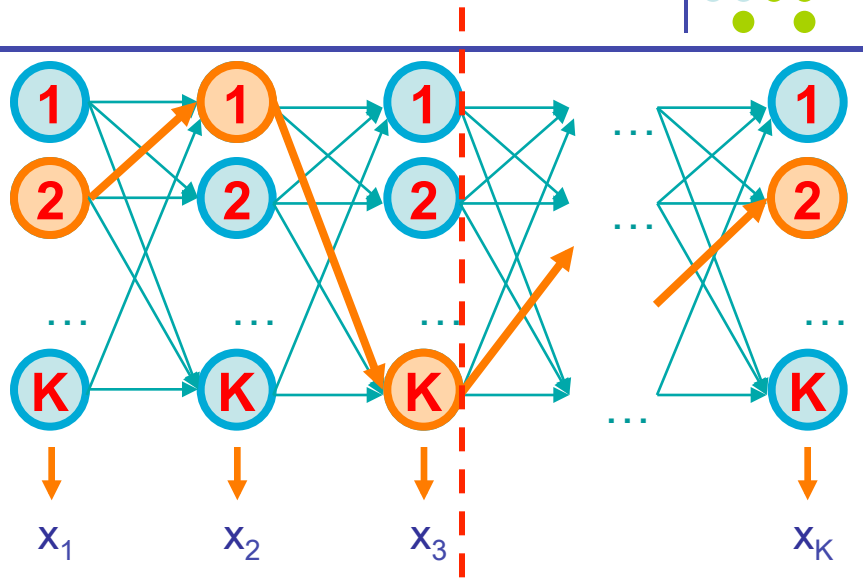
$\pi^* = \text{argmax}_\pi P[x, \pi]$

Maximizes $a_{0\pi 1}\, e_{\pi 1}(x_1)\, a_{\pi 1 \pi 2} \ldots\ldots a_{\pi N-1 \pi N}\, e_{\pi N}(x_N)$

Dynamic Programming!

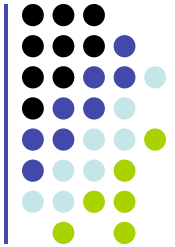
$V_k(i) = \max_{\{\pi 1 \ldots \pi i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

= Prob. of most likely sequence of states ending at state $\pi_i = k$



Given that we end up in state k at step i, maximize product to the left and right

Decoding – Review



Inductive assumption: Given that for all states k,
and for a fixed position i,

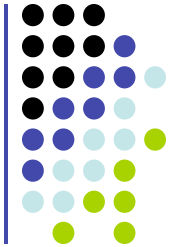
$$\mathbf{V_k(i) = \max_{\{\pi 1 \ldots \pi i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]}$$

What is $V_l(i+1)$?

From definition,

$$V_l(i+1) = \max_{\{\pi 1 \ldots \pi i\}} P[\, x_1 \ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l \,]$$
$$= \max_{\{\pi 1 \ldots \pi i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_1 \ldots x_i, \pi_1, \ldots, \pi_i)\, P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i]$$
$$= \max_{\{\pi 1 \ldots \pi i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i)\, P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i]$$
$$= \max_k \left[P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k)\, \mathbf{\max_{\{\pi 1 \ldots \pi i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]} \right]$$
$$= \max_k \left[\, P(x_{i+1} \mid \pi_{i+1} = l)\, P(\pi_{i+1} = l \mid \pi_i = k)\, \mathbf{V_k(i)} \,\right]$$
$$= e_l(x_{i+1}) \max_k a_{kl}\, \mathbf{V_k(i)}$$

The Viterbi Algorithm - Review



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

Initialization:

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

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

Iteration:

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

$Ptr_j(i) = \text{argmax}_k a_{kj} V_k(i-1)$

Termination:

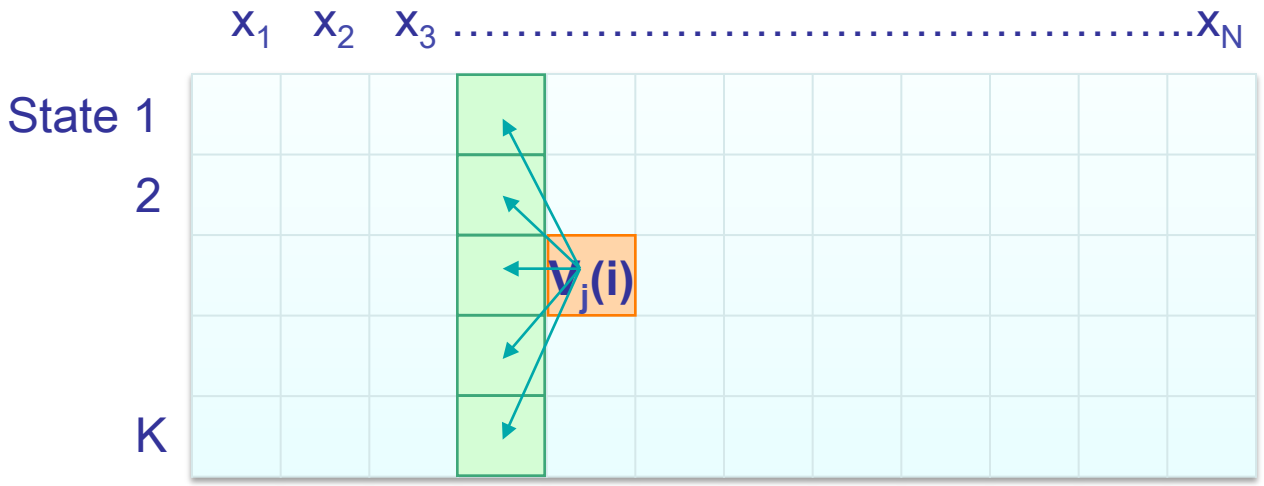
$P(x, \pi^*) = \max_k V_k(N)$

Traceback:

$\pi_N^* = \text{argmax}_k V_k(N)$

$\pi_{i-1}^* = Ptr_{\pi i}(i)$

The Viterbi Algorithm - Review



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

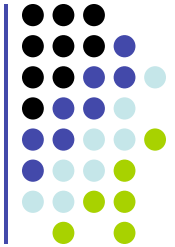
Time:

$O(K^2N)$

Space:

$O(KN)$

Viterbi Algorithm – Review



Underflows are a significant problem

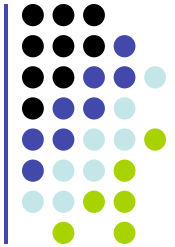
$$P[x_1,\ldots., x_i, \pi_1, \ldots, \pi_i] = a_{0\pi1} a_{\pi1\pi2}\ldots\ldots a_{\pi i} e_{\pi1}(x_1)\ldots\ldots e_{\pi i}(x_i)$$

These numbers become extremely small – underflow

Solution: Take the logs of all values

$$V_l(i) = \log e_k(x_i) + \max_k [V_k(i-1) + \log a_{kl}]$$

Example - Review



Let x be a long sequence with a portion of ~ 1/6 6's,
followed by a portion of ~ ½ 6's…

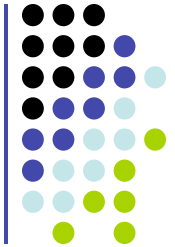
x = 123456123456…12345 6626364656…1626364656

Then, it is not hard to show that optimal parse is (exercise):

FFF……...F LLL……...L

6 characters "123456" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$
parsed as L, contribute $.95^6 \times (1/2)^1 \times (1/10)^5 = 0.4 \times 10^{-5}$

"162636" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$
parsed as L, contribute $.95^6 \times (1/2)^3 \times (1/10)^3 = 9.0 \times 10^{-5}$



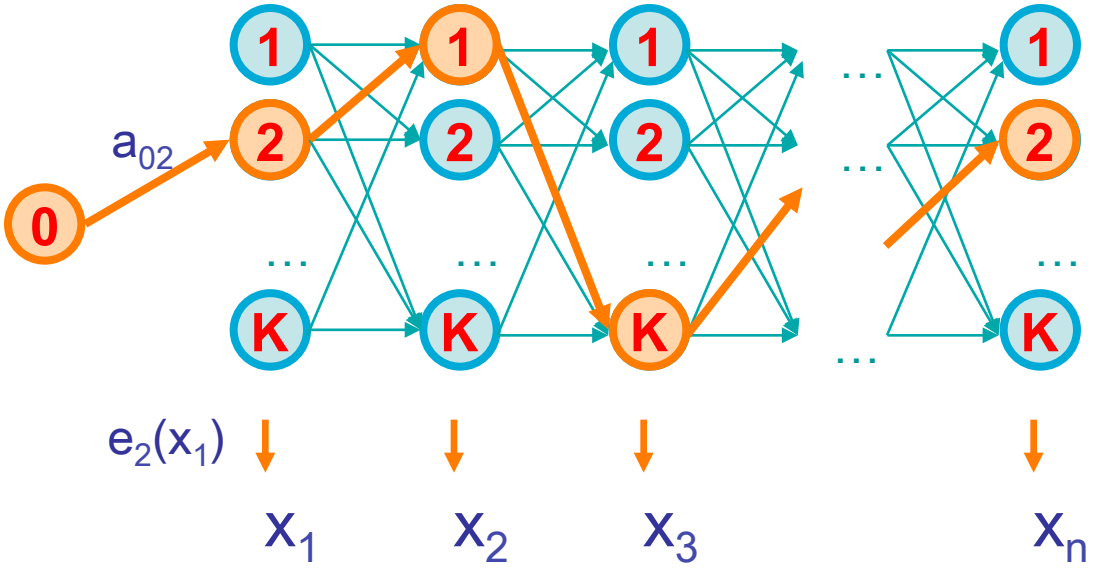
Problem 2: Evaluation

Find the likelihood a sequence is generated by the model

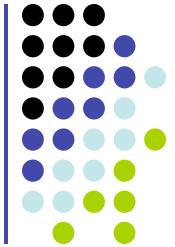
Generating a sequence by the model

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

1. Start at state π_1 according to prob $a_{0\pi 1}$
2. Emit letter x_1 according to prob $e_{\pi 1}(x_1)$
3. Go to state π_2 according to prob $a_{\pi 1 \pi 2}$
4. … until emitting x_n



A couple of questions



Given a sequence x,

- What is the probability that
- Given a position i, what is th

Example: the dishonest ca

P(box: FFFFFFFFFFF) =
$(1/6)^{11} * 0.95^{12}$ =
$2.76^{-9} * 0.54$ =
1.49^{-9}

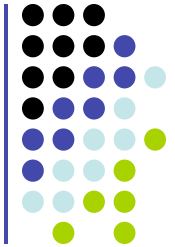
P(box: LLLLLLLLLL) =
$[(1/2)^6 * (1/10)^5] * 0.95^{10} * 0.05^2$ =
$1.56*10^{-7} * 1.5^{-3}$ =
0.23^{-9}

Say x = 12341…23162616364616234112…21341

F F

Most likely path: π = FF……F
(too "unlikely" to transition F → L → F)
However: marked letters more likely to be L than unmarked letters

Evaluation

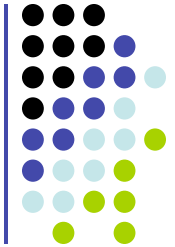


We will develop algorithms that allow us to compute:

$P(x)$	Probability of x given the model
$P(x_i \ldots x_j)$	Probability of a substring of x given the model
$P(\pi_i = k \mid x)$	"Posterior" probability that the i^{th} state is k, given x

A more refined measure of which states x may be in

The Forward Algorithm



We want to calculate

P(x) = probability of x, given the HMM

Sum over all possible ways of generating x:

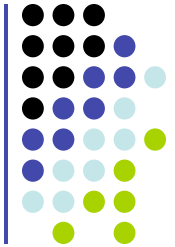
$$P(x) = \Sigma_{\pi} P(x, \pi) = \Sigma_{\pi} P(x \mid \pi) P(\pi)$$

To avoid summing over an exponential number of paths π, define

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k) \quad \text{(the forward probability)}$$

"generate i first characters of x and end up in state k"

The Forward Algorithm – derivation



Define the forward probability:

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k)$$

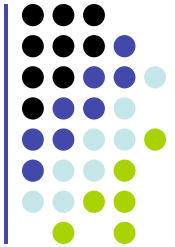
$$= \sum_{\pi 1 \ldots \pi i-1} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, \pi_i = k)\, e_k(x_i)$$

$$= \sum_l \sum_{\pi 1 \ldots \pi i-2} \mathbf{P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-2}, \pi_{i-1} = l)}\, a_{lk}\, e_k(x_i)$$

$$= \sum_l \mathbf{P(x_1 \ldots x_{i-1}, \pi_{i-1} = l)}\, a_{lk}\, e_k(x_i)$$

$$= e_k(x_i) \sum_l \mathbf{f_l(i-1)}\, a_{lk}$$

The Forward Algorithm



We can compute $f_k(i)$ for all k, i, using dynamic programming!

Initialization:

$f_0(0) = 1$

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

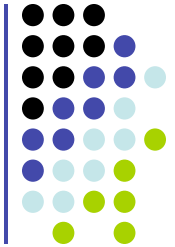
Iteration:

$$f_k(i) = e_k(x_i) \sum_l f_l(i - 1)\, a_{lk}$$

Termination:

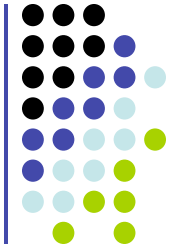
$$P(x) = \sum_k f_k(N)$$

Relation between Forward and Viterbi



VITERBI	FORWARD
Initialization:	**Initialization:**
$V_0(0) = 1$	$f_0(0) = 1$
$V_k(0) = 0$, for all $k > 0$	$f_k(0) = 0$, for all $k > 0$
Iteration:	**Iteration:**
$V_j(i) = e_j(x_i) \max_k V_k(i-1)\, a_{kj}$	$f_l(i) = e_l(x_i) \sum_k f_k(i-1)\, a_{kl}$
Termination:	**Termination:**
$P(x, \pi^*) = \max_k V_k(N)$	$P(x) = \sum_k f_k(N)$

Motivation for the Backward Algorithm



We want to compute

$$P(\pi_i = k \mid x),$$

the probability distribution on the i^{th} position, given x

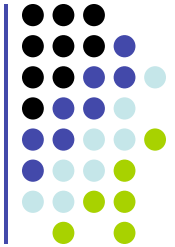
We start by computing

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

Forward, $f_k(i)$ Backward, $b_k(i)$

Then, $P(\pi_i = k \mid x) = P(\pi_i = k, x) / P(x)$

The Backward Algorithm – derivation



Define the backward probability:

$b_k(i) = P(x_{i+1} \ldots x_N \mid \pi_i = k)$ *"starting from i^{th} state = k, generate rest of x"*

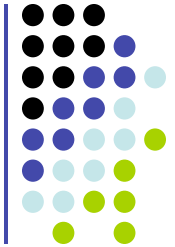
$= \sum_{\pi i+1 \ldots \pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k)$

$= \sum_l \sum_{\pi i+1 \ldots \pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1} = l, \pi_{i+2}, \ldots, \pi_N \mid \pi_i = k)$

$= \sum_l e_l(x_{i+1})\, a_{kl} \sum_{\pi i+1 \ldots \pi N} P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N \mid \pi_{i+1} = l)$

$= \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$

The Backward Algorithm



We can compute $b_k(i)$ for all k, i, using dynamic programming

Initialization:

$b_k(N) = 1$, for all k

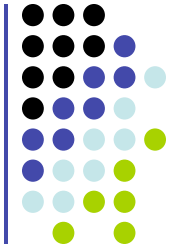
Iteration:

$$b_k(i) = \sum_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$$

Termination:

$$P(x) = \sum_l a_{0l}\, e_l(x_1)\, b_l(1)$$

Computational Complexity



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

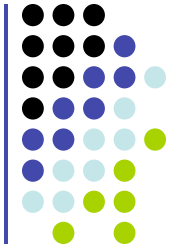
Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs

Forward/Backward: rescaling at each few positions by multiplying by a constant

Posterior Decoding



We can now calculate

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

Then, we can ask

$P(\pi_i = k \mid x) =$

$P(\pi_i = k\ , x)/P(x) =$

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

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

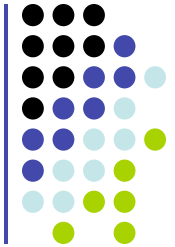
$f_k(i)\, b_k(i) / P(x)$

What is the most likely state at position i of sequence x:

Define $\hat{\pi}$ by Posterior Decoding:

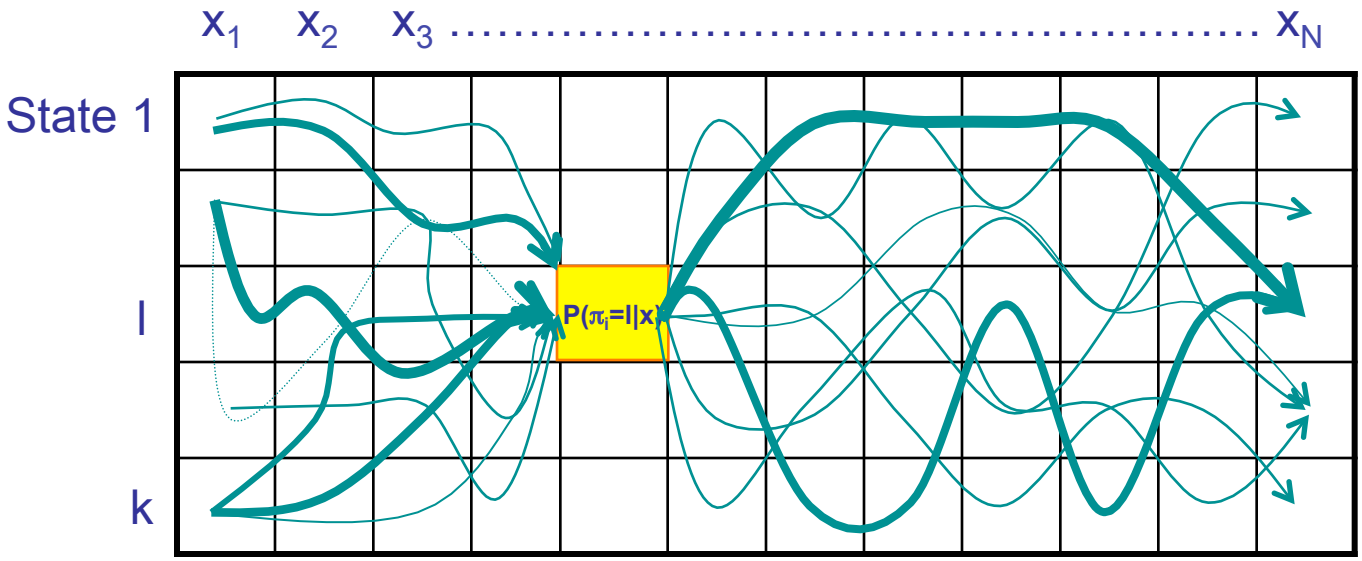
$$\hat{\pi}_i = \text{argmax}_k\, P(\pi_i = k \mid x)$$

Posterior Decoding



- For each state,
  - Posterior Decoding gives us a curve of likelihood of state for each position
  - That is sometimes more informative than Viterbi path π^*
- Posterior Decoding may give an invalid sequence of states (of prob 0)
  - Why?

Posterior Decoding

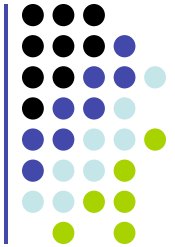


- $P(\pi_i = k \mid x) = \sum_{\pi} P(\pi \mid x)\, \mathbf{1}(\pi_i = k)$

$= \sum_{\{\pi:\pi[i] = k\}} P(\pi \mid x)$

$\mathbf{1}(\psi) = 1$, if ψ is true
0, otherwise

Viterbi, Forward, Backward



VITERBI

Initialization:

$V_0(0) = 1$

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

Iteration:

$V_l(i) = e_l(x_i)\ \max_k V_k(i\text{-}1)\ a_{kl}$

Termination:

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

FORWARD

Initialization:

$f_0(0) = 1$

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

Iteration:

$f_l(i) = e_l(x_i) \sum_k f_k(i\text{-}1)\ a_{kl}$

Termination:

$P(x) = \sum_k f_k(N)$

BACKWARD

Initialization:

$b_k(N) = 1$, for all k

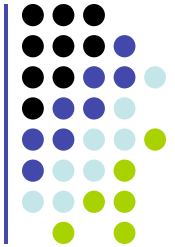
Iteration:

$b_l(i) = \sum_k e_l(x_i{+}1)\ a_{kl}\ b_k(i{+}1)$

Termination:

$P(x) = \sum_k a_{0k}\ e_k(x_1)\ b_k(1)$

Viterbi, Forward, Backward



VITERBI

Initialization:

$V_0(0) = 1$

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

Iteration:

$V_l(i) = e_l(x_i)\ \max_k V_k(i\text{-}1)\ a_{kl}$

Termination:

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

FORWARD

Initialization:

$f_0(0) = 1$

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

Iteration:

$f_l(i) = e_l(x_i) \sum_k f_k(i\text{-}1)\ a_{kl}$

Termination:

$P(x) = \sum_k f_k(N)$

BACKWARD

Initialization:

$b_k(N) = 1$, for all k

Iteration:

$b_l(i) = \sum_k e_l(x_i{+}1)\ a_{kl}\ b_k(i{+}1)$

Termination:

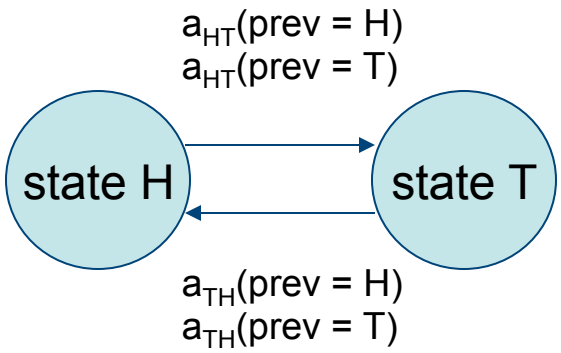
$P(x) = \sum_k a_{0k}\ e_k(x_1)\ b_k(1)$

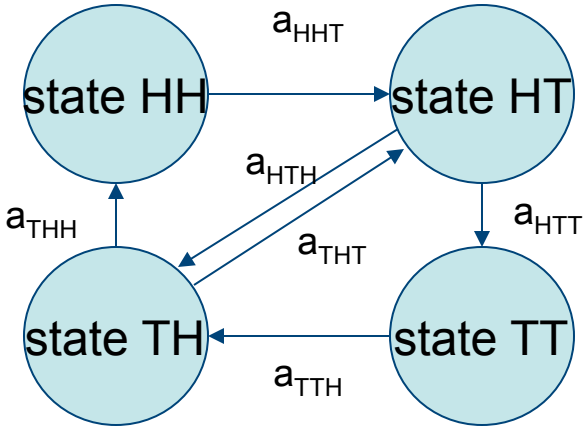
Variants of HMMs

Higher-order HMMs

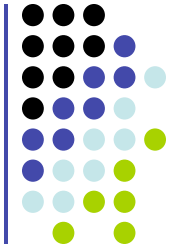
- How do we model “memory” larger than one time point?

- $P(\pi_{i+1} = l \mid \pi_i = k)$ a_{kl}
- $P(\pi_{i+1} = l \mid \pi_i = k, \pi_{i-1} = j)$ a_{jkl}
- …
- A second order HMM with K states is equivalent to a first order HMM with K^2 states



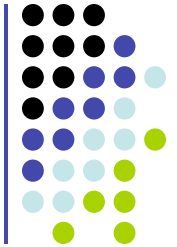


Similar Algorithms to 1st Order



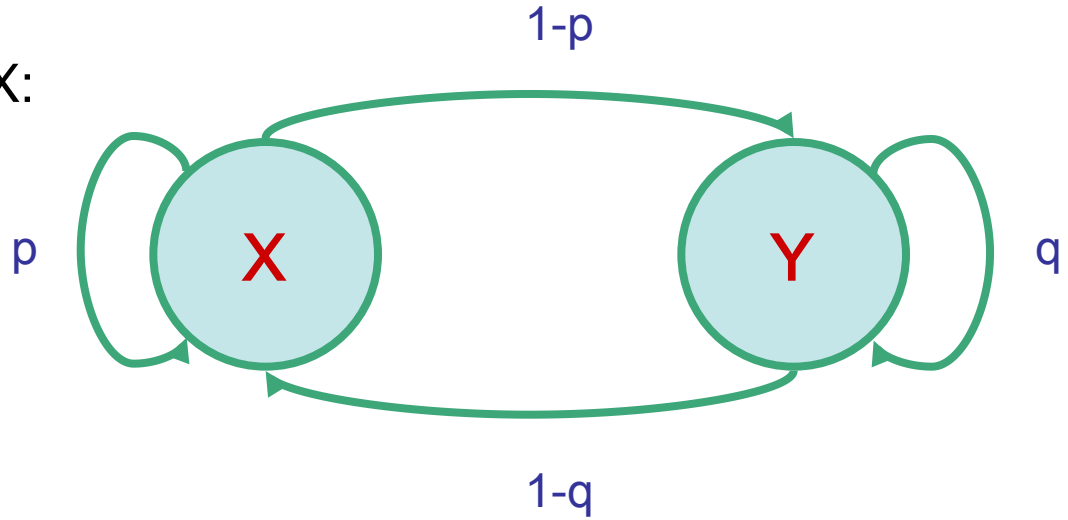
- $P(\pi_{i+1} = l \mid \pi_i = k, \pi_{i-1} = j)$
  - $V_{lk}(i) = \max_j\{ V_{kj}(i - 1) + \ldots \}$
  - Time? Space?

Modeling the Duration of States



Length distribution of region X:

$E[l_X] = 1/(1-p)$

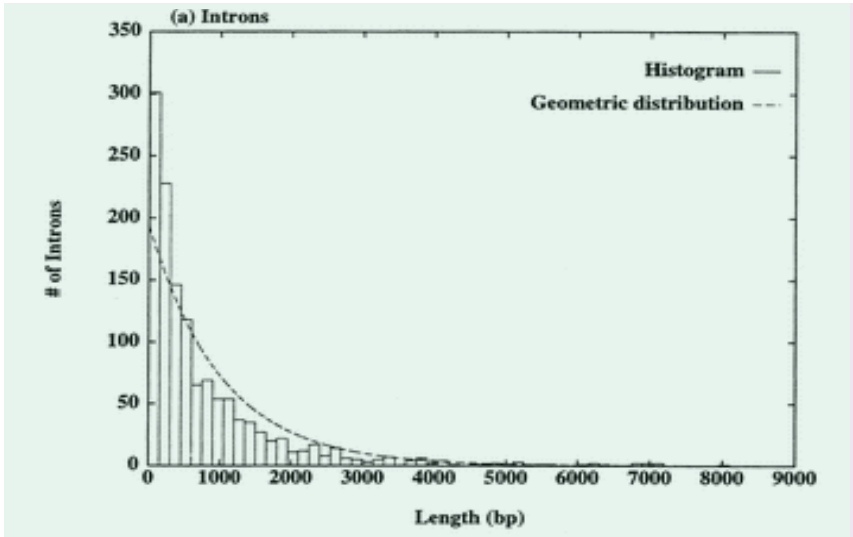


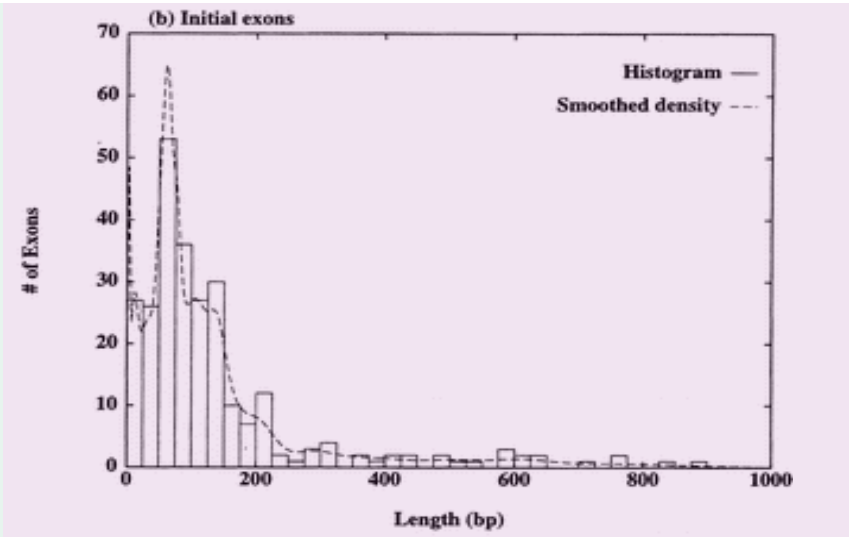
- Geometric distribution, with mean 1/(1-p)

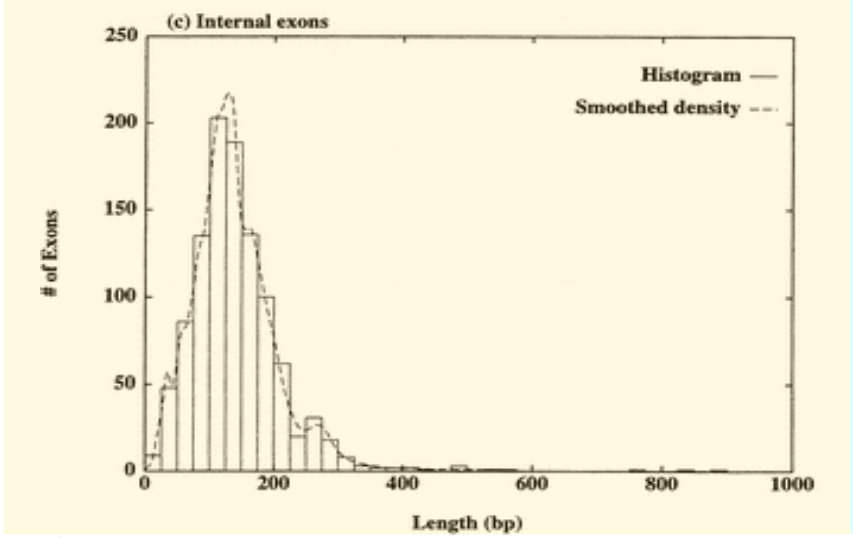
This is a significant disadvantage of HMMs

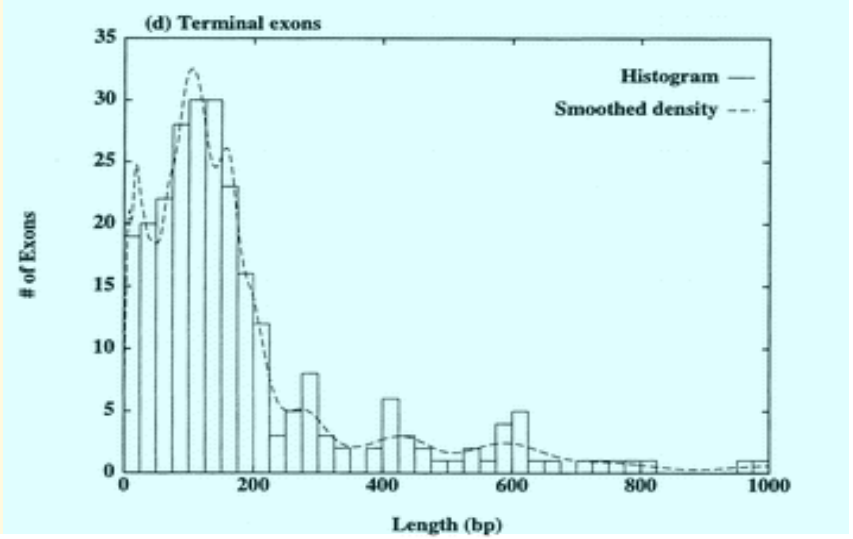
Several solutions exist for modeling different length distributions

Example: exon lengths in genes

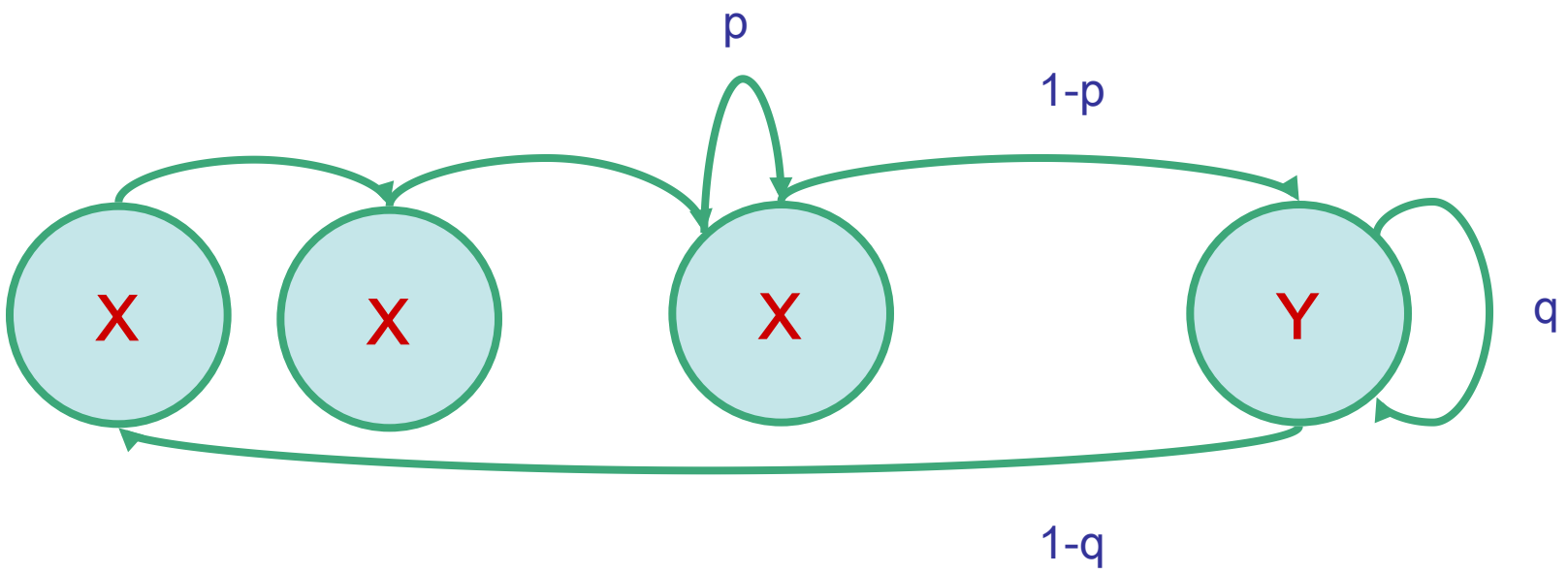








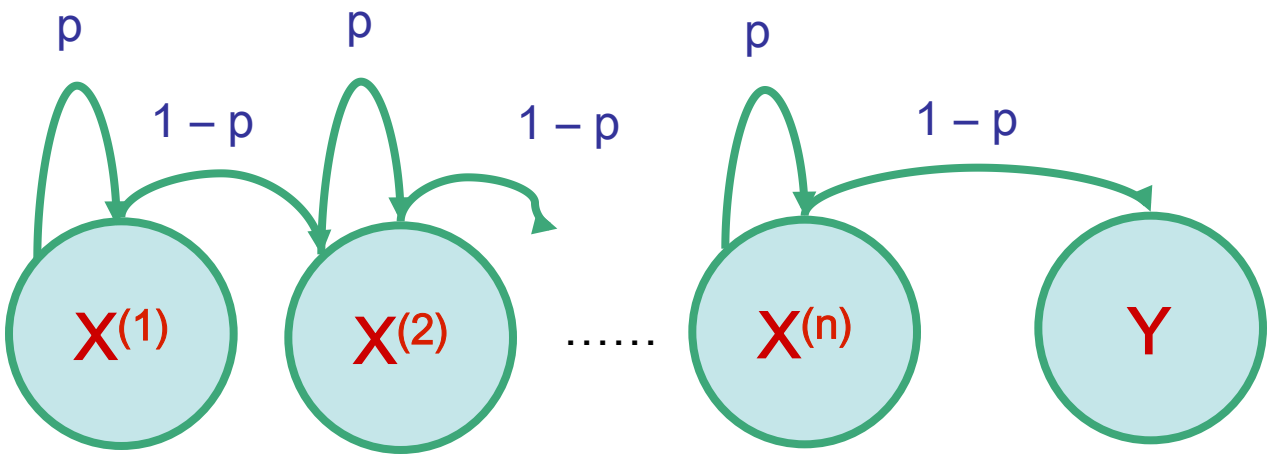
Solution 1: Chain several states



Disadvantage: Still very inflexible

l_X = C + geometric with mean 1/(1-p)

Solution 2: Negative binomial distribution

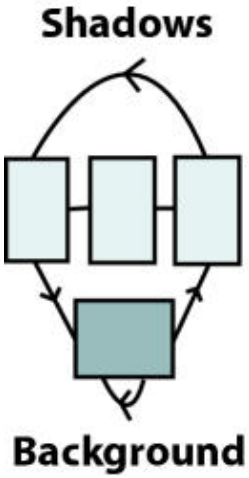


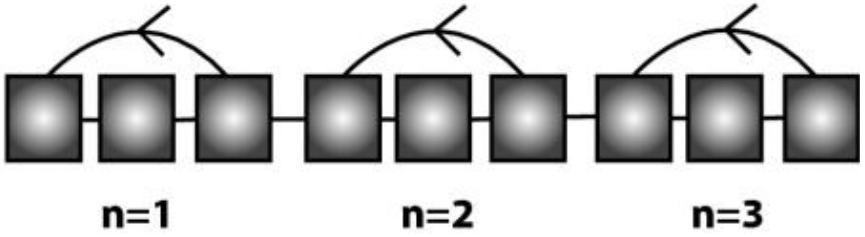
Duration in **X**: **m** turns, where

- During first **m – 1** turns, exactly **n – 1** arrows to next state are followed
- During **m**th turn, an arrow to next state is followed

$$P(I_X = m) = \begin{bmatrix} m-1 \\ n-1 \end{bmatrix} (1-p)^{n-1+1} p^{(m-1)-(n-1)} = \begin{bmatrix} m-1 \\ n-1 \end{bmatrix} (1-p)^n p^{m-n}$$

Example: genes in prokaryotes

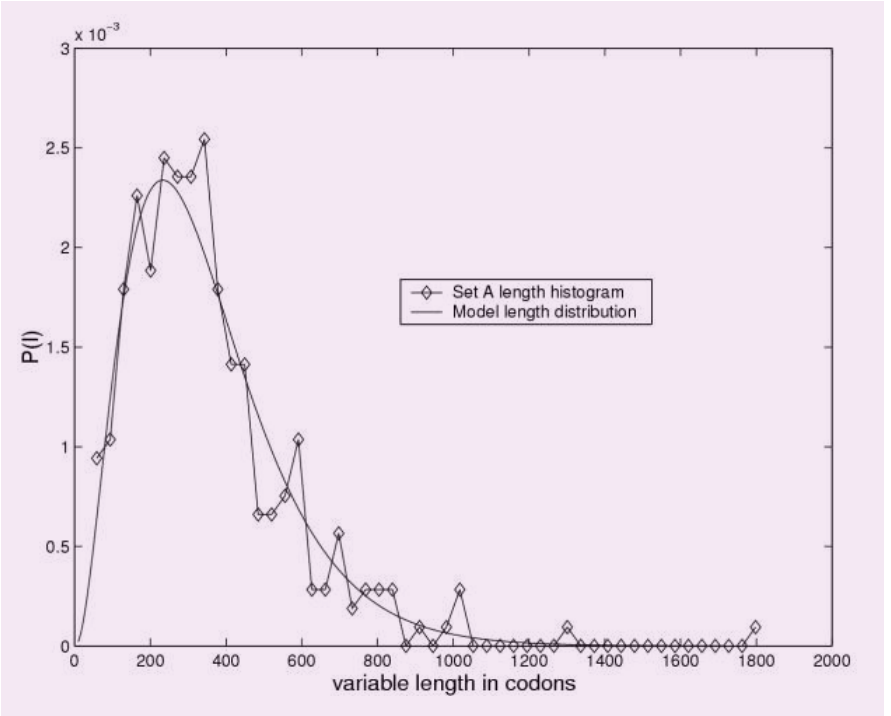




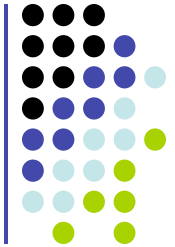
- EasyGene: Prokaryotic gene-finder

  Larsen TS, Krogh A

- Negative binomial with n = 3

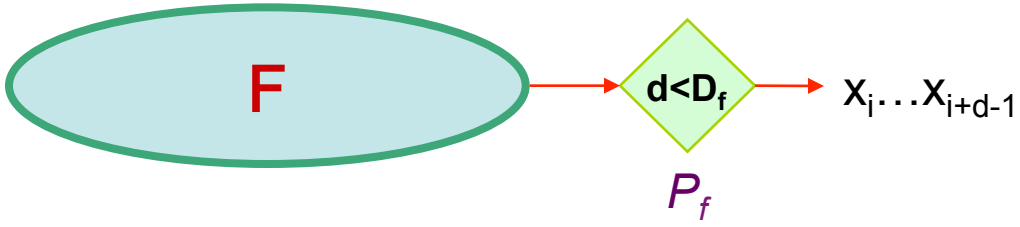


Solution 3: Duration modeling



Upon entering a state:

1. Choose duration d, according to probability distribution
2. Generate d letters according to emission probs
3. Take a transition to next state according to transition probs



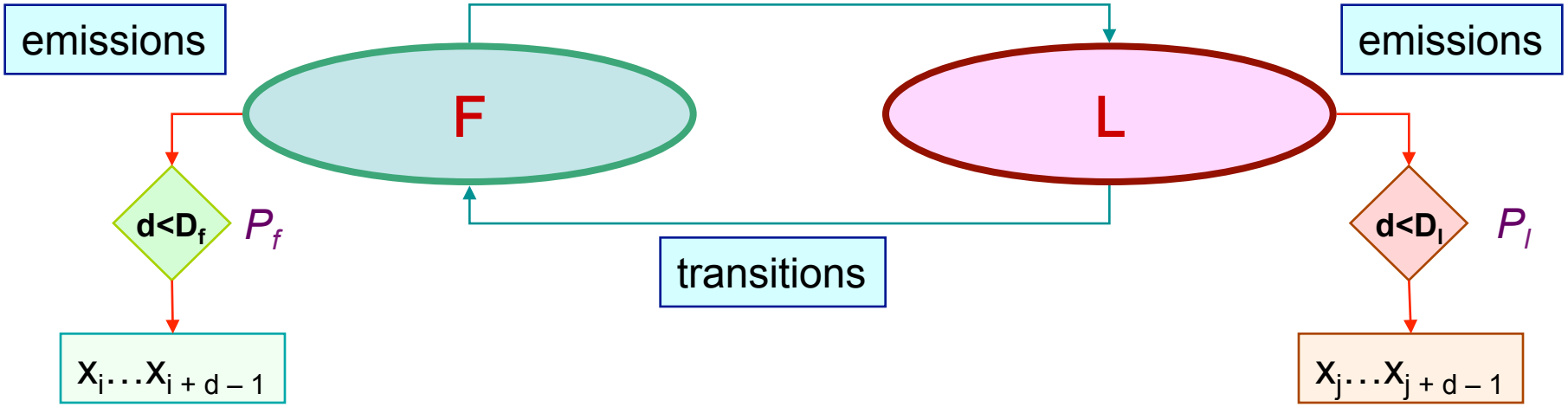
Disadvantage: Increase in complexity of Viterbi:

Time: O(D)
Space: O(1)

Warning, Rabiner's tutorial claims $O(D^2)$ & O(D) increases

where D = maximum duration of state

Viterbi with duration modeling



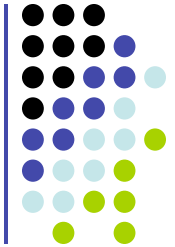
Recall original iteration:

$$V_l(i) = \mathbf{max}_k \; V_k(i-1)\, a_{kl} \times e_l(x_i)$$

Precompute cumulative values

New iteration:

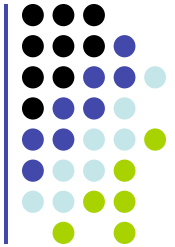
$$V_l(i) = \mathbf{max}_k \; \mathbf{max}_{d=1\ldots Dl} \; V_k(i-d) \times \mathbf{P_l(d)} \times a_{kl} \times \prod_{j=i-d+1\ldots i} e_l(x_j)$$



Learning

Re-estimate the parameters of the model based on training data

Two learning scenarios



1. Estimation when the "right answer" is known

   Examples:

   GIVEN: a genomic region $x = x_1 \ldots x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands

   GIVEN: the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

2. Estimation when the "right answer" is unknown

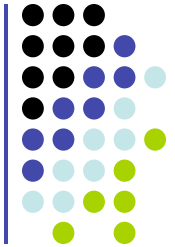
   Examples:

   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

QUESTION: Update the parameters θ of the model to maximize $P(x|\theta)$

1. When the states are known



Given $x = x_1 \ldots x_N$
for which the true $\pi = \pi_1 \ldots \pi_N$ is known,

Define:

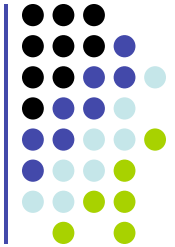
A_{kl} = # times k→l transition occurs in π
$E_k(b)$ = # times state k in π emits b in x

We can show that the maximum likelihood parameters θ (maximize $P(x|\theta)$) are:

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

$$e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}$$

1. When the states are known



Intuition: When we know the underlying states,
Best estimate is the normalized frequency of
transitions & emissions that occur in the training data

Drawback:

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

Example:

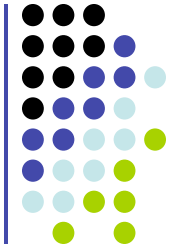
Given 10 casino rolls, we observe

x = 2, 1, 5, 6, 1, 2, 3, 6, 2, 3
π = F, F, F, F, F, F, F, F, F, F

Then:

$a_{FF} = 1$; $a_{FL} = 0$
$e_F(1) = e_F(3) = .2$;
$e_F(2) = .3$; $e_F(4) = 0$; $e_F(5) = e_F(6) = .1$

Pseudocounts



Solution for small training sets:

Add pseudocounts

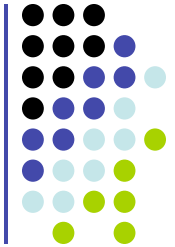
A_{kl}	= # times k→l transition occurs in π	+ r_{kl}
$E_k(b)$	= # times state k in π emits b in x	+ $r_k(b)$

r_{kl}, $r_k(b)$ are pseudocounts representing our prior belief

Larger pseudocounts ⇒ Strong priof belief

Small pseudocounts (ε < 1): just to avoid 0 probabilities

Pseudocounts



Example: dishonest casino

We will observe player for one day, 600 rolls

Reasonable pseudocounts:

$r_{0F} = r_{0L} = r_{F0} = r_{L0} = 1;$

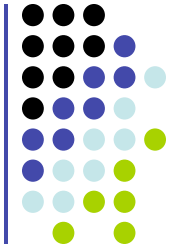
$r_{FL} = r_{LF} = r_{FF} = r_{LL} = 1;$

$r_F(1) = r_F(2) = \ldots = r_F(6) = 20$ (strong belief fair is fair)

$r_L(1) = r_L(2) = \ldots = r_L(6) = 5$ (wait and see for loaded)

Above #s are arbitrary – assigning priors is an art

2. When the states are hidden

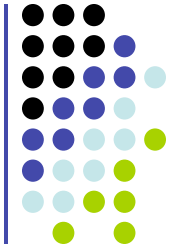


We don't know the true A_{kl}, $E_k(b)$

Idea:

- We estimate our "best guess" on what A_{kl}, $E_k(b)$ are
  - Or, we start with random / uniform values
- We update the parameters of the model, based on our guess
- We repeat

2. When the states are hidden



Starting with our best guess of a model M, parameters θ:

Given $x = x_1 \ldots x_N$

for which the true $\pi = \pi_1 \ldots \pi_N$ is unknown,

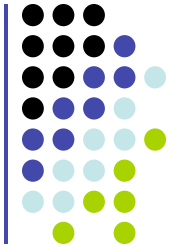
We can get to a provably more likely parameter set θ

i.e., θ that increases the probability $P(x \mid \theta)$

Principle: EXPECTATION MAXIMIZATION

1. Estimate A_{kl}, $E_k(b)$ in the training data
2. Update θ according to A_{kl}, $E_k(b)$
3. Repeat 1 & 2, until convergence

Estimating new parameters



To estimate A_{kl}: (assume "$| \theta_{CURRENT}$", in all formulas below)

At each position i of sequence x, find probability transition k→l is used:

$$P(\pi_i = k, \pi_{i+1} = l \mid x) = [1/P(x)] \times P(\pi_i = k, \pi_{i+1} = l, x_1 \ldots x_N) = Q/P(x)$$

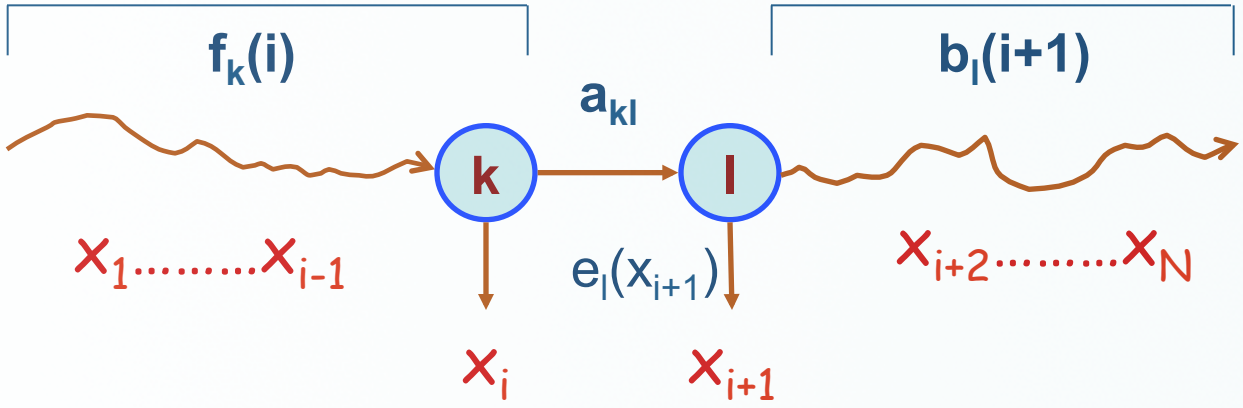
where
$$Q = P(x_1 \ldots x_i, \pi_i = k, \pi_{i+1} = l, x_{i+1} \ldots x_N) =$$
$$= P(\pi_{i+1} = l, x_{i+1} \ldots x_N \mid \pi_i = k)\, P(x_1 \ldots x_i, \pi_i = k) =$$
$$= P(\pi_{i+1} = l, x_{i+1} x_{i+2} \ldots x_N \mid \pi_i = k)\, f_k(i) =$$
$$= P(x_{i+2} \ldots x_N \mid \pi_{i+1} = l)\, P(x_{i+1} \mid \pi_{i+1} = l)\, P(\pi_{i+1} = l \mid \pi_i = k)\, f_k(i) =$$
$$= b_l(i+1)\, e_l(x_{i+1})\, a_{kl}\, f_k(i)$$

So:
$$\mathbf{P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \frac{f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1)}{P(x \mid \theta_{CURRENT})}}$$

Estimating new parameters

- So, A_{kl} is the E[# times transition k→l, given current θ]

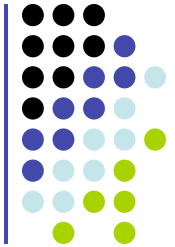
$$A_{kl} = \sum_i P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \sum_i \frac{f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1)}{P(x \mid \theta)}$$



- Similarly,

$$E_k(b) = [1/P(x \mid \theta)] \sum_{\{i \mid x^i = b\}} f_k(i)\, b_k(i)$$

The Baum-Welch Algorithm



Initialization:

Pick the best-guess for model parameters

(or arbitrary)

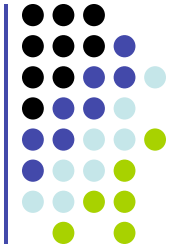
Iteration:

1. Forward
2. Backward
3. Calculate A_{kl}, $E_k(b)$, given $\theta_{CURRENT}$
4. Calculate new model parameters θ_{NEW} : a_{kl}, $e_k(b)$
5. Calculate new log-likelihood $P(x \mid \theta_{NEW})$

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until $P(x \mid \theta)$ does not change much

The Baum-Welch Algorithm



Time Complexity:

iterations $\times$ $O(K^2N)$

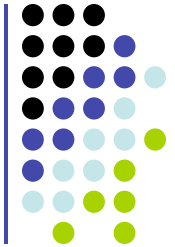
- Guaranteed to increase the log likelihood $P(x \mid \theta)$

- Not guaranteed to find globally best parameters

  Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overtraining

Alternative: Viterbi Training



Initialization: Same

Iteration:

1. Perform Viterbi, to find π^*
2. Calculate A_{kl}, $E_k(b)$ according to π^* + pseudocounts
3. Calculate the new parameters a_{kl}, $e_k(b)$

Until convergence

Notes:

- Not guaranteed to increase $P(x \mid \theta)$
- Guaranteed to increase $P(x \mid \theta, \pi^*)$
- In general, worse performance than Baum-Welch