Hidden Markov Models

BMI/CS 576

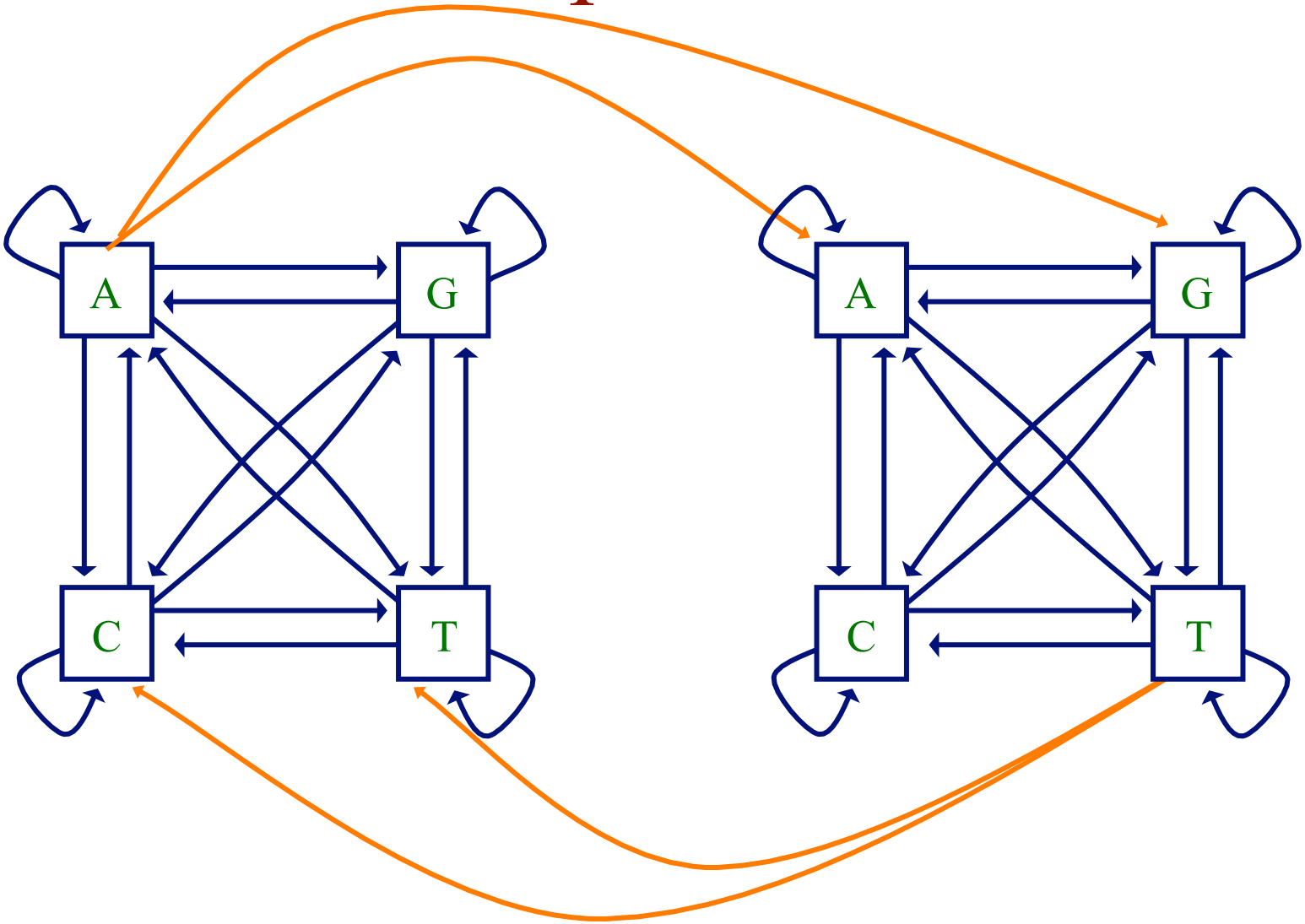
www.biostat.wisc.edu/bmi576.html

Colin Dewey

cdewey@biostat.wisc.edu

Fall 2008

A Simple HMM



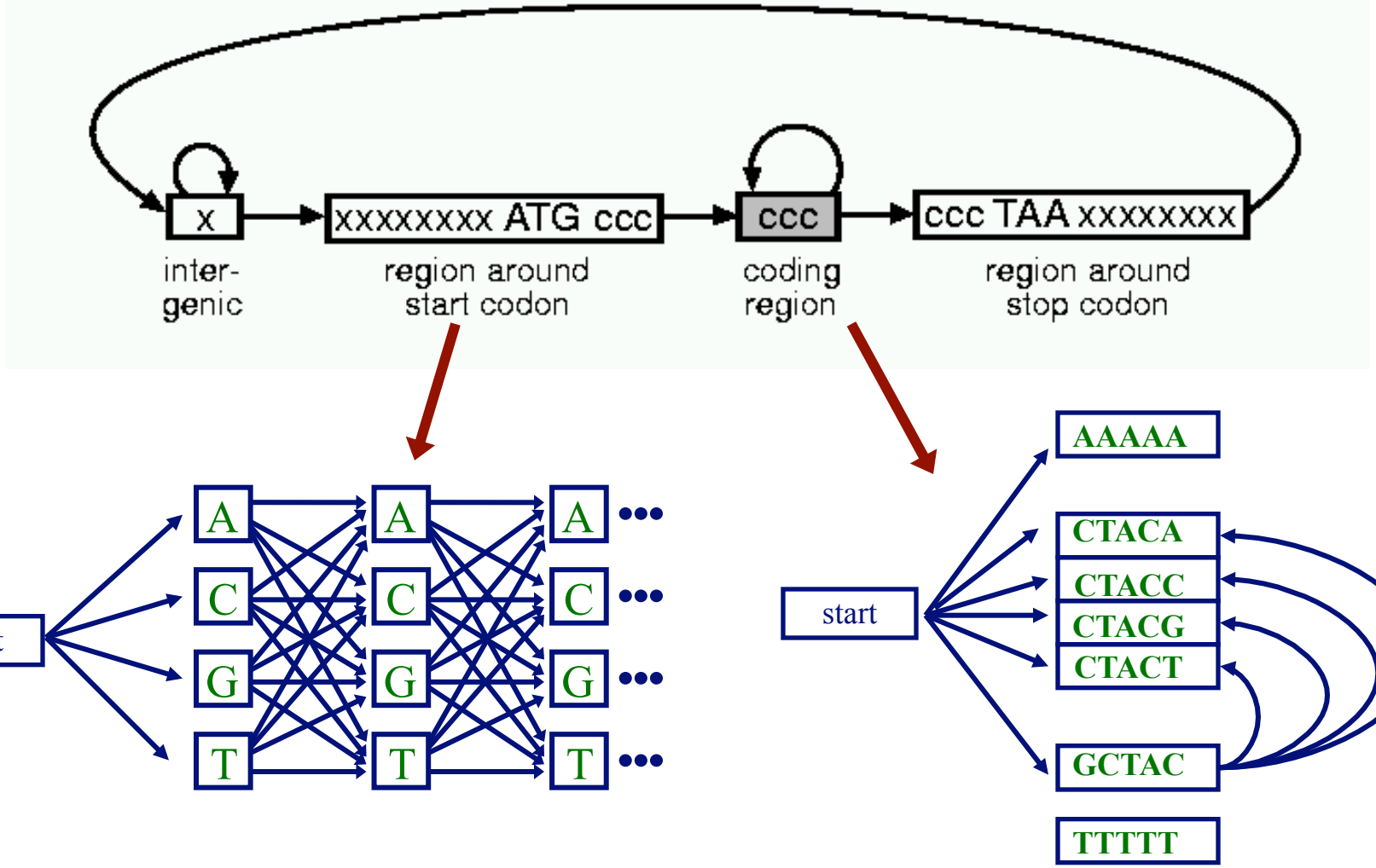
- given say a *T* in our input sequence, which state emitted it?

Hidden State

- we'll distinguish between the *observed* parts of a problem and the *hidden* parts
- in the Markov models we've considered previously, it is clear which state accounts for each part of the observed sequence
- in the model above, there are multiple states that could account for each part of the observed sequence
  - this is the hidden part of the problem

Simple HMM for Gene Finding

Figure from A. Krogh, An Introduction to Hidden Markov Models for Biological Sequences



The Parameters of an HMM

- as in Markov chain models, we have transition probabilities

$$a_{kl} = \Pr(\pi_i = l \mid \pi_{i-1} = k)$$

probability of a transition from state k to l

π represents a path (sequence of states) through the model

- since we've decoupled states and characters, we might also have emission probabilities

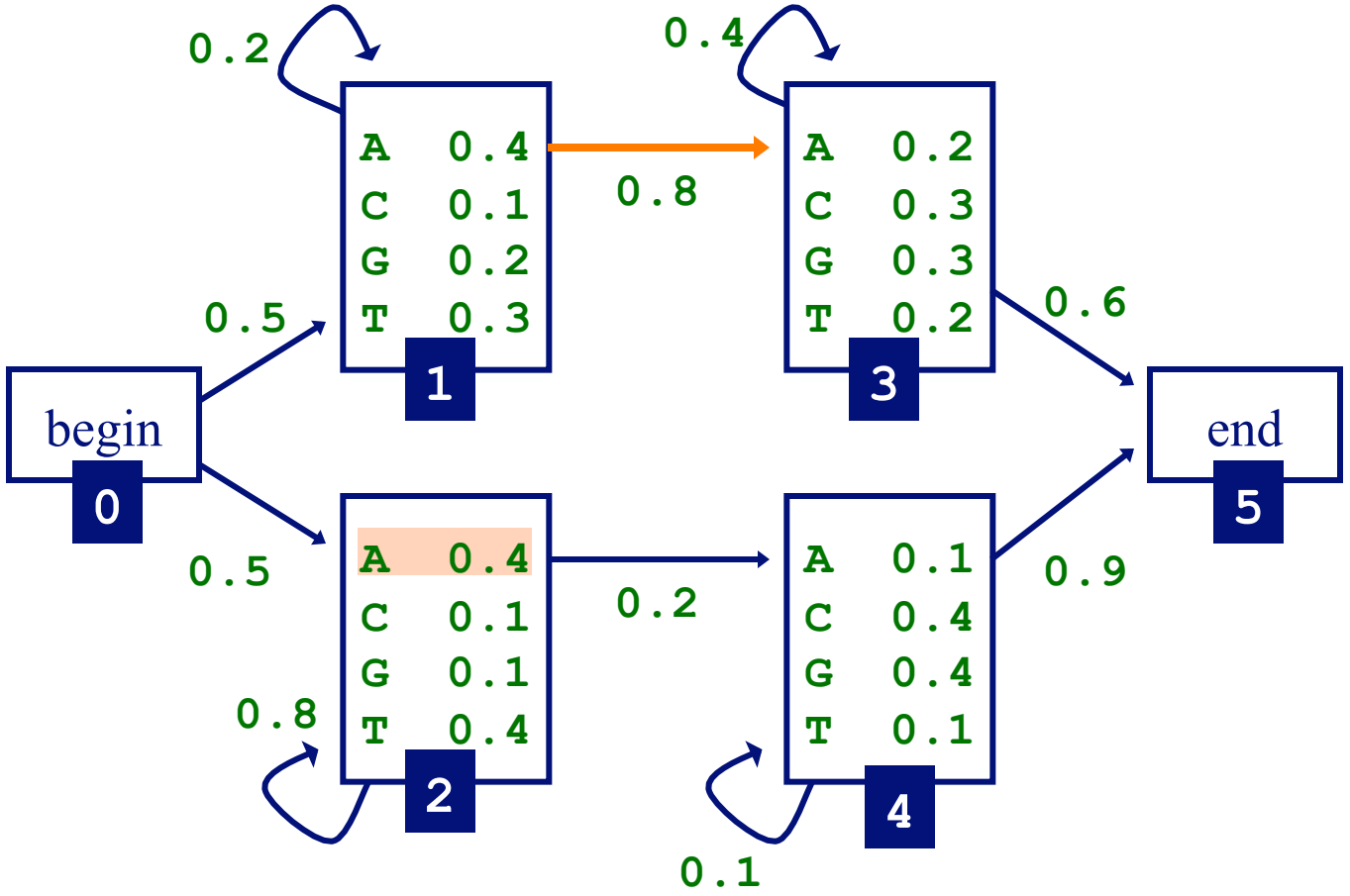
$$e_k(b) = \Pr(X_i = b \mid \pi_i = k)$$

probability of emitting character b in state k

A Simple HMM with Emission Parameters

a_{13} probability of a transition from state 1 to state 3

$e_2(\mathrm{A})$ probability of emitting character *A* in state 2



Three Important Questions

- How likely is a given sequence?
  the Forward algorithm

- What is the most probable "path" for generating a given sequence?
  the Viterbi algorithm

- How can we learn the HMM parameters given a set of sequences?
  the Forward-Backward (Baum-Welch) algorithm

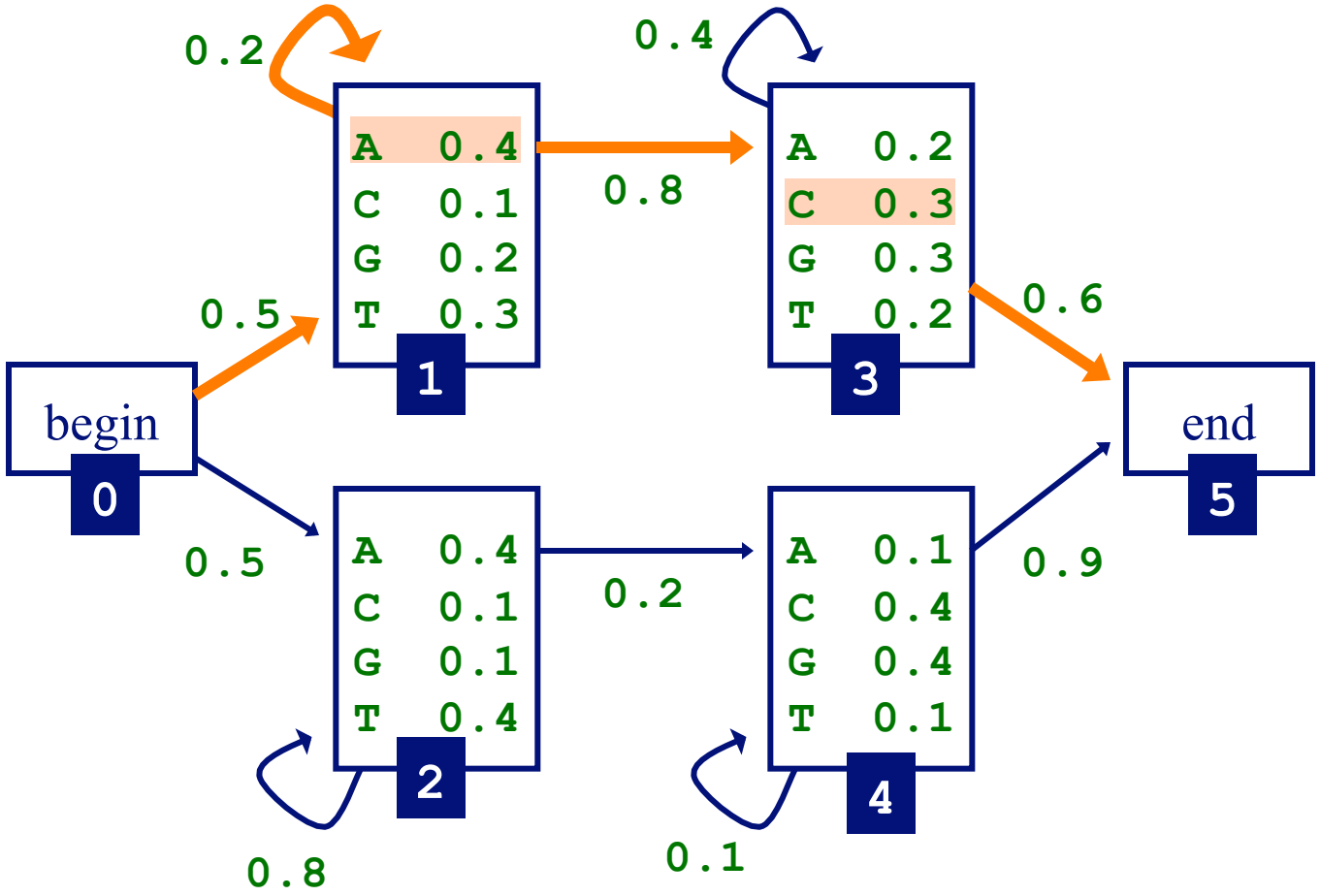
How Likely is a Given Sequence?

- the probability that the path $\pi_1 \ldots \pi_L$ is taken and the sequence $X_1 \ldots X_L$ is generated:

$$\Pr(X_1 \ldots X_L, \pi_1 \ldots \pi_L) = a_{0\pi_1} a_{\pi_L N} \prod_{i=1}^{L-1} a_{\pi_i \pi_{i+1}} \prod_{i=1}^{L} e_{\pi_i}(X_i)$$

(assuming begin/end are the only silent states on path)

How Likely Is A Given Path and Sequence?



$$\Pr(\text{AAC}, \pi) = a_{01} \times e_1(\text{A}) \times a_{11} \times e_1(\text{A}) \times a_{13} \times e_3(\text{C}) \times a_{35}$$

$$= 0.5 \times 0.4 \times 0.2 \times 0.4 \times 0.8 \times 0.3 \times 0.6$$

How Likely is a Given Sequence?

- the probability over *all* paths is:

$$\Pr(X_1 \ldots X_L) = \sum_{\pi} \Pr(X_1 \ldots X_L, \underbrace{\pi_1 \ldots \pi_L}_{\pi})$$

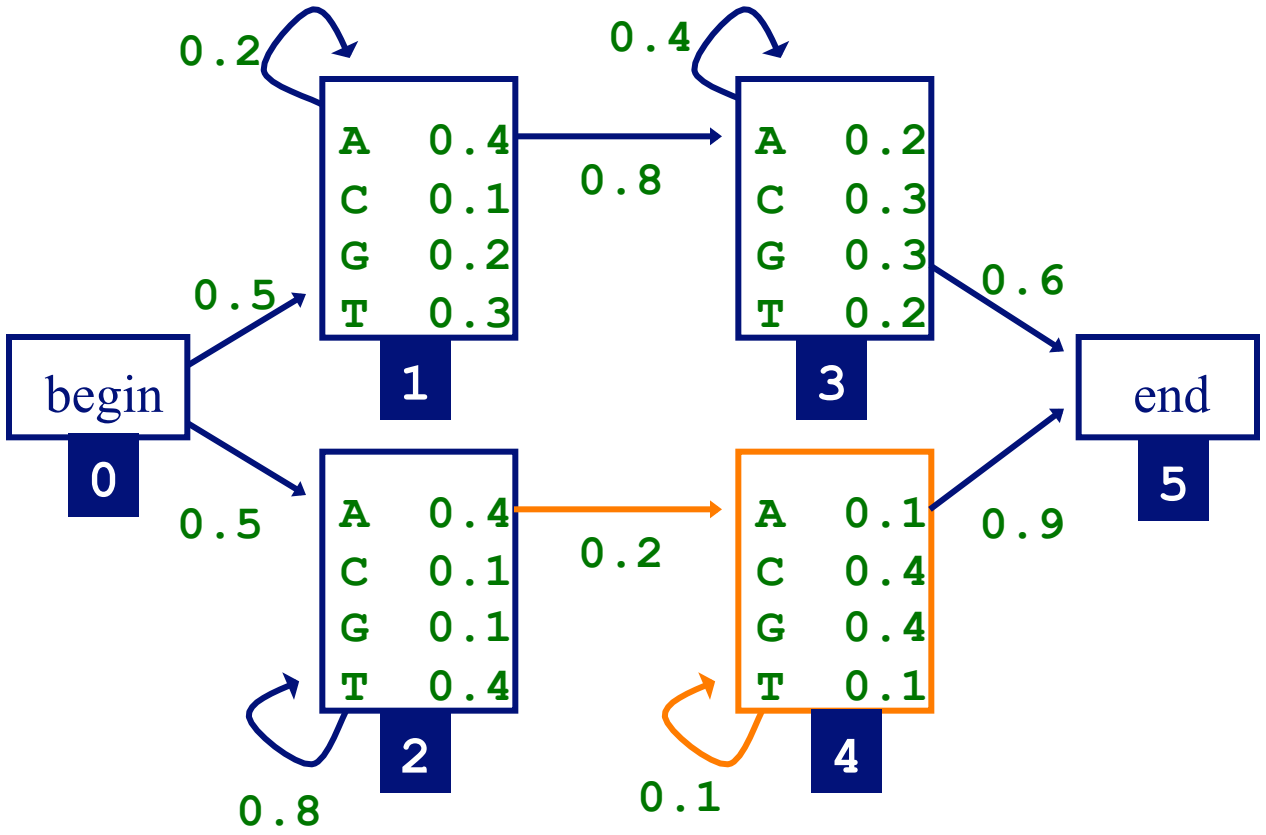
- but the number of paths can be exponential in the length of the sequence...
- the Forward algorithm enables us to compute this efficiently

How Likely is a Given Sequence: The Forward Algorithm

- A dynamic programming solution
- subproblem: define $f_k(i)$ to be the probability of being in state k having observed the first i characters of x
- we want to compute $f_N(L)$, the probability of being in the end state having observed all of x
- can define this recursively

The Forward Algorithm

- because of the Markov property, don't have to explicitly enumerate every path



- e.g. compute $f_4(i)$ using $f_2(i-1),\ f_4(i-1)$

The Forward Algorithm

- initialization:

$$f_0(0) = 1$$

probability that we're in start state and
have observed 0 characters from the sequence

$$f_k(0) = 0, \quad \text{for } k \text{ that are not silent states}$$

The Forward Algorithm

- recursion for emitting states ($i = 1 \ldots L$):

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

- recursion for silent states:

$$f_l(i) = \sum_k f_k(i) a_{kl}$$

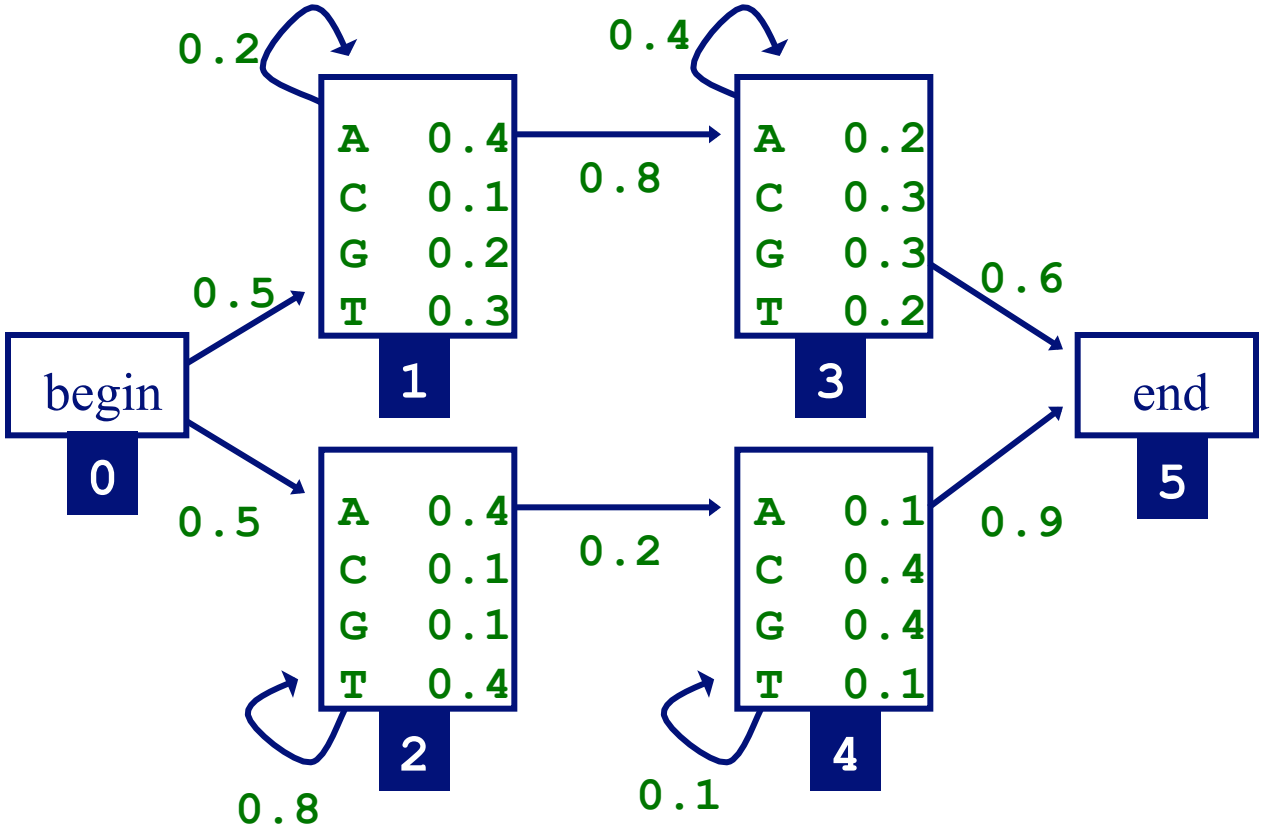
The Forward Algorithm

- termination:

$$\Pr(X) = \Pr(X_1 \ldots X_L) = f_N(L) = \sum_k f_k(L) a_{kN}$$

probability that we're in the end state and have observed the entire sequence

Forward Algorithm Example



- given the sequence x = **TAGA**

Forward Algorithm Example

- given the sequence x = TAGA
- initialization

$$f_0(0) = 1 \qquad f_1(0) = 0 \ \ldots \ f_5(0) = 0$$

- computing other values

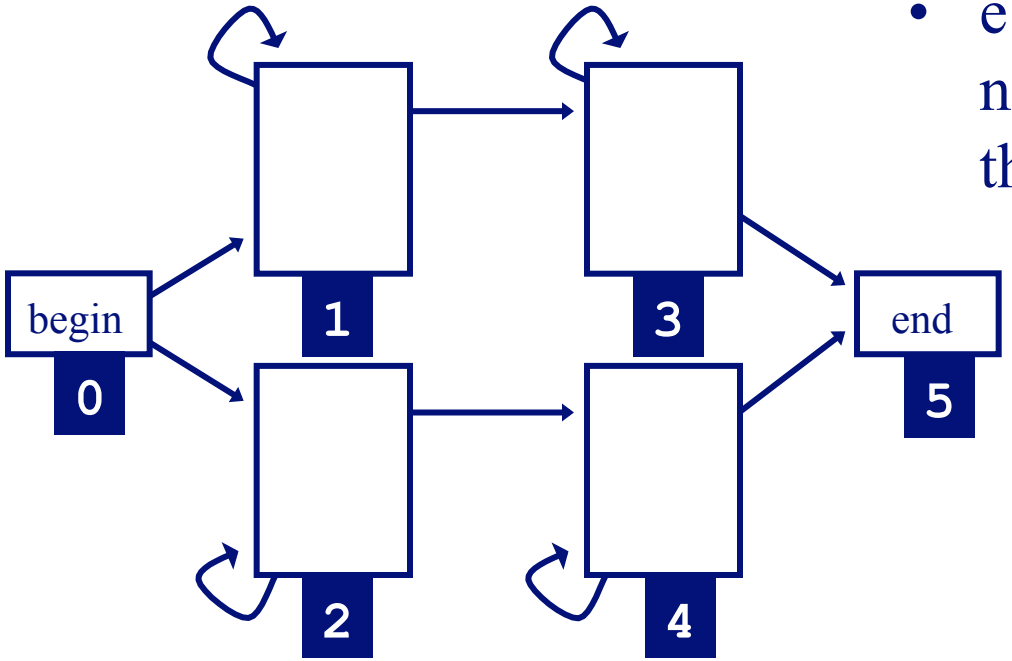
$$f_1(1) = e_1(T) \times (f_0(0) \times a_{01} + f_1(0) a_{11}) =$$
$$0.3 \times (1 \times 0.5 + 0 \times 0.2) = 0.15$$
$$f_2(1) = 0.4 \times (1 \times 0.5 + 0 \times 0.8)$$
$$f_1(2) = e_1(A) \times (f_0(1) \times a_{01} + f_1(1) a_{11}) =$$
$$0.4 \times (0 \times 0.5 + 0.15 \times 0.2)$$

• • •

$$\Pr(TAGA) = f_5(4) = (f_3(4) \times a_{35} + f_4(4) a_{45})$$

Forward Algorithm Note

- in some cases, we can make the algorithm more efficient by taking into account the minimum number of steps that must be taken to reach a state



- e.g. for this HMM, we don't need to initialize or compute the values

$$f_3(0),\ f_4(0),$$
$$f_5(0),\ f_5(1)$$

Three Important Questions

- How likely is a given sequence?
- What is the most probable "path" for generating a given sequence?
- How can we learn the HMM parameters given a set of sequences?

Finding the Most Probable Path: The Viterbi Algorithm

- Dynamic programming approach, again!
- subproblem: define $v_k(i)$ to be the probability of the most probable path accounting for the first *i* characters of *x* and ending in state *k*

- we want to compute $v_N(L)$, the probability of the most probable path accounting for all of the sequence and ending in the end state
- can define recursively
- can use DP to find $v_N(L)$ efficiently

Finding the Most Probable Path: The Viterbi Algorithm

- initialization:

$$v_0(0) = 1$$

$$v_k(0) = 0, \quad \text{for } k \text{ that are not silent states}$$

The Viterbi Algorithm

- recursion for emitting states ($i = 1 \ldots L$):

$$v_l(i) = e_l(x_i) \max_k \left[v_k(i-1) a_{kl} \right]$$

$$\text{ptr}_l(i) = \arg\max_k \left[v_k(i-1) a_{kl} \right]$$

keep track of most probable path

- recursion for silent states:

$$v_l(i) = \max_k \left[v_k(i) a_{kl} \right]$$

$$\text{ptr}_l(i) = \arg\max_k \left[v_k(i) a_{kl} \right]$$

The Viterbi Algorithm

- termination:

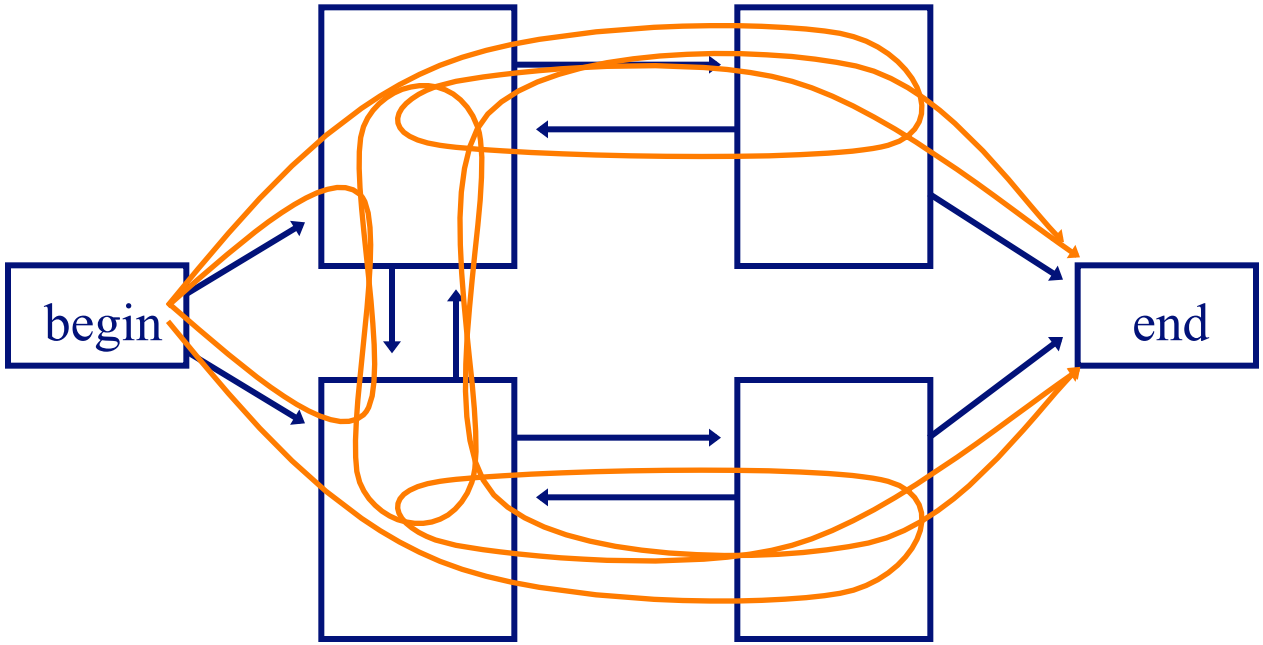
$$\Pr(x,\pi^*) = \max_k \left(v_k(L)a_{kN}\right)$$

$$\pi_L^* = \arg\max_k \left(v_k(L)a_{kN}\right)$$

- traceback: follow pointers back starting at π_L^*

Forward & Viterbi Algorithms

- Forward/Viterbi algorithms effectively consider all possible paths for a sequence
  - Forward to find probability of a sequence
  - Viterbi to find most probable path
- consider a sequence of length 4…

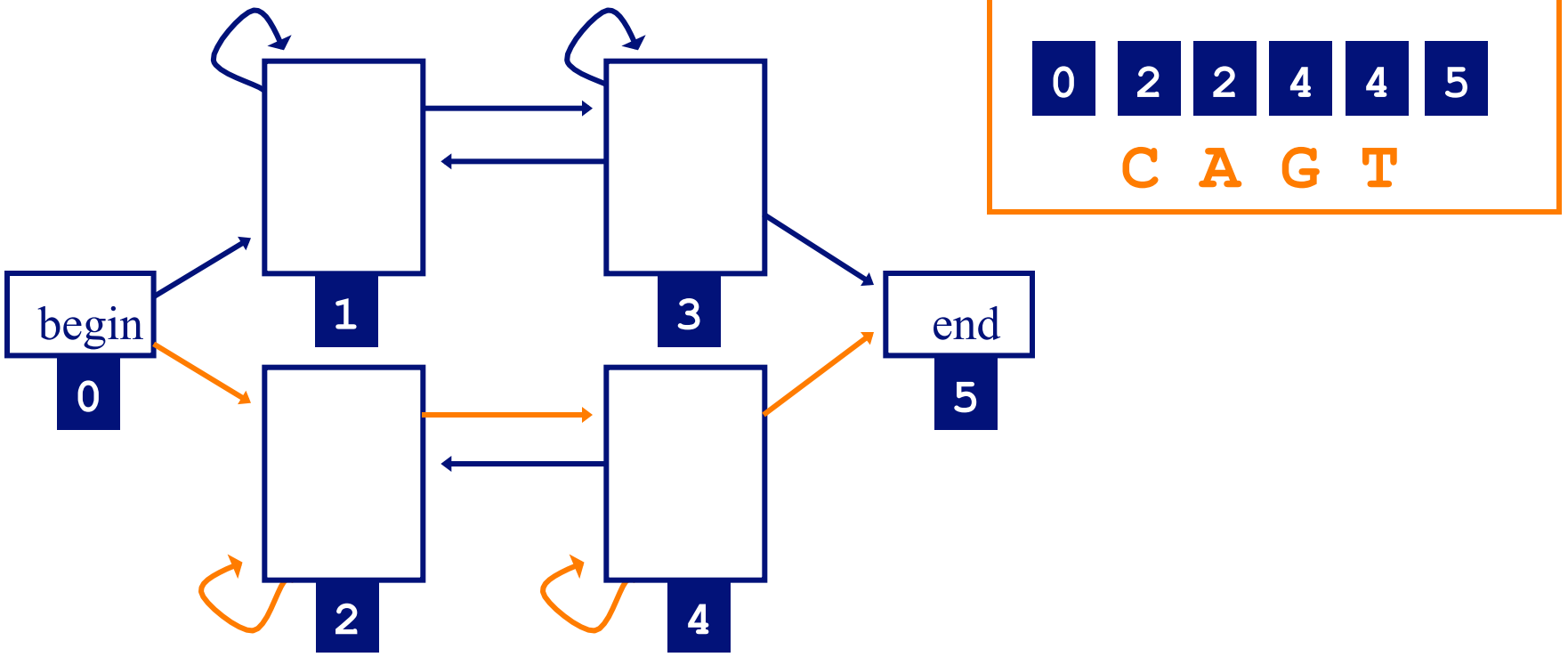


Three Important Questions

- How likely is a given sequence?
- What is the most probable "path" for generating a given sequence?
- How can we learn the HMM parameters given a set of sequences?

Learning without Hidden State

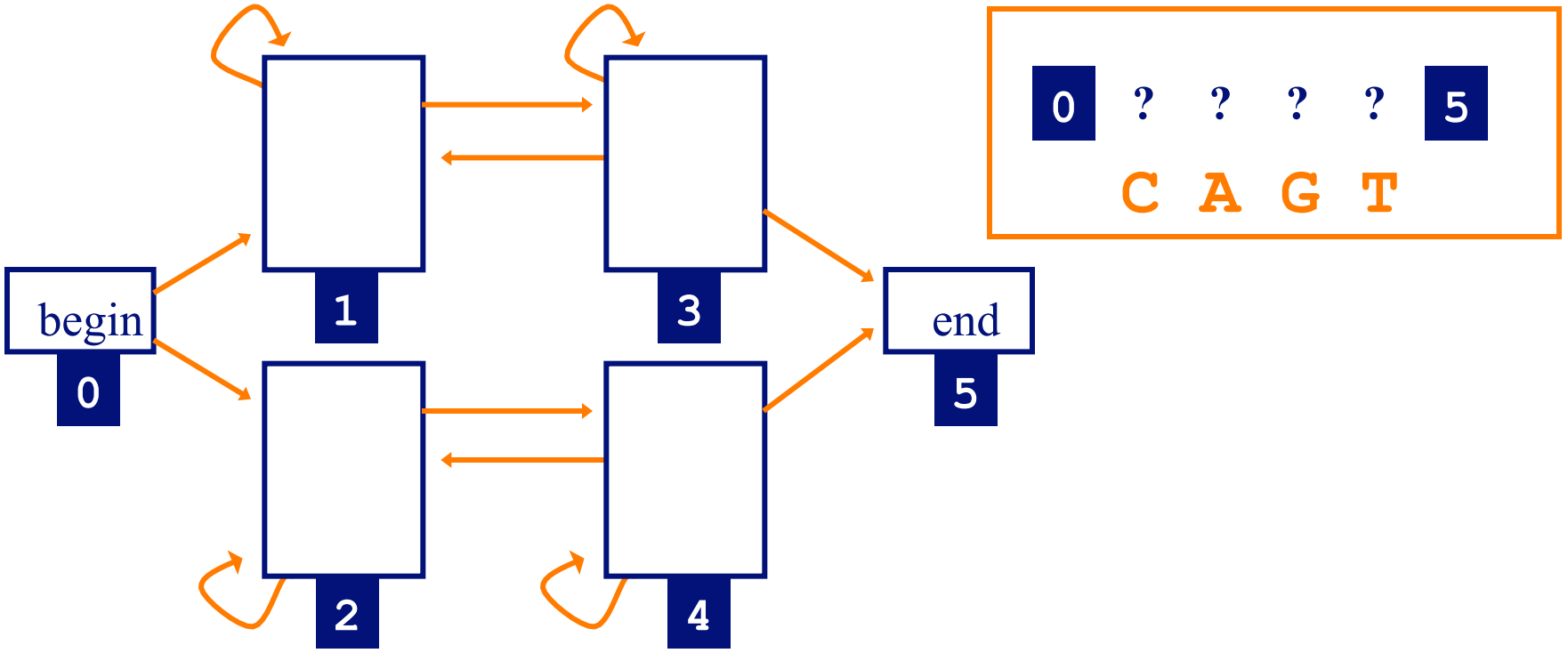
- learning is simple if we know the correct path for each sequence in our training set



- estimate parameters by counting the number of times each parameter is used across the training set

Learning with Hidden State

- if we don't know the correct path for each sequence in our training set, consider all possible paths for the sequence



- estimate parameters through a procedure that counts the expected number of times each transition and emission occurs across the training set

Learning Parameters

- if we know the state path for each training sequence, learning the model parameters is simple
  - no hidden state during training
  - count how often each transition and emission occurs
  - normalize/smooth to get probabilities
  - process is just like it was for Markov chain models

- if we don't know the path for each training sequence, how can we determine the counts?
  - key insight: estimate the counts by considering every path weighted by its probability

Learning Parameters: The Baum-Welch Algorithm

- *a.k.a* the Forward-Backward algorithm
- an *Expectation Maximization* (EM) algorithm
  - EM is a family of algorithms for learning probabilistic models in problems that involve hidden state
- in this context, the hidden state is the path that explains each training sequence

Learning Parameters: The Baum-Welch Algorithm

- algorithm sketch:
  - initialize parameters of model
  - iterate until convergence
    - calculate the *expected* number of times each transition or emission is used
    - adjust the parameters to *maximize* the likelihood of these expected values

The Expectation Step

- first, we need to know the probability of the *i* th symbol being produced by state *k,* given sequence *x*

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

- given this we can compute our expected counts for state transitions, character emissions

The Expectation Step

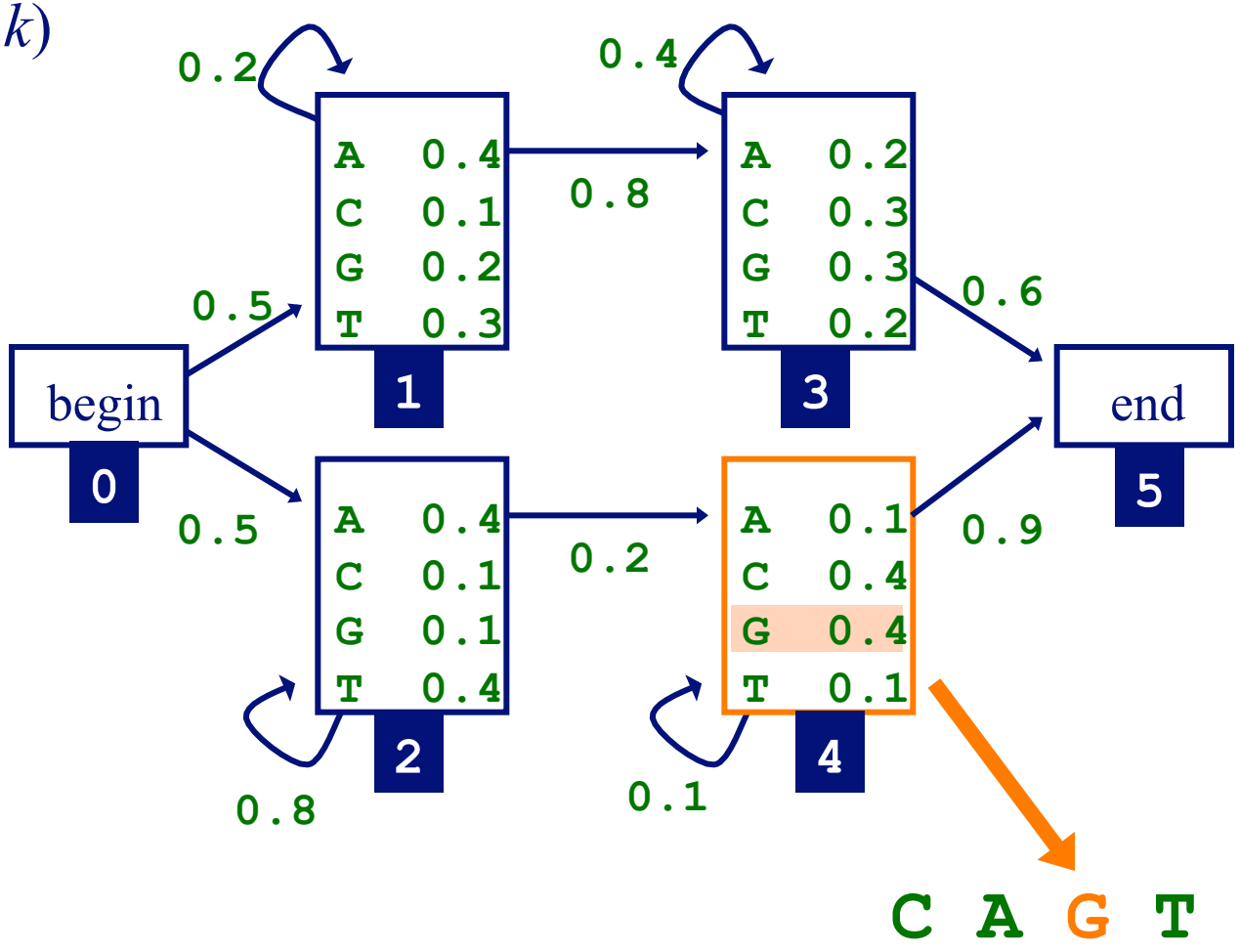
- the probability of of producing *x* with the *i* th symbol being produced by state *k* is

$$\Pr(\pi_i = k, x) = \Pr(x_1 \ldots x_i, \pi_i = k) \times \Pr(x_{i+1} \ldots x_L \mid \pi_i = k)$$

- the first term is $f_k(i)$, computed by the forward algorithm
- the second term is $b_k(i)$, computed by the backward algorithm

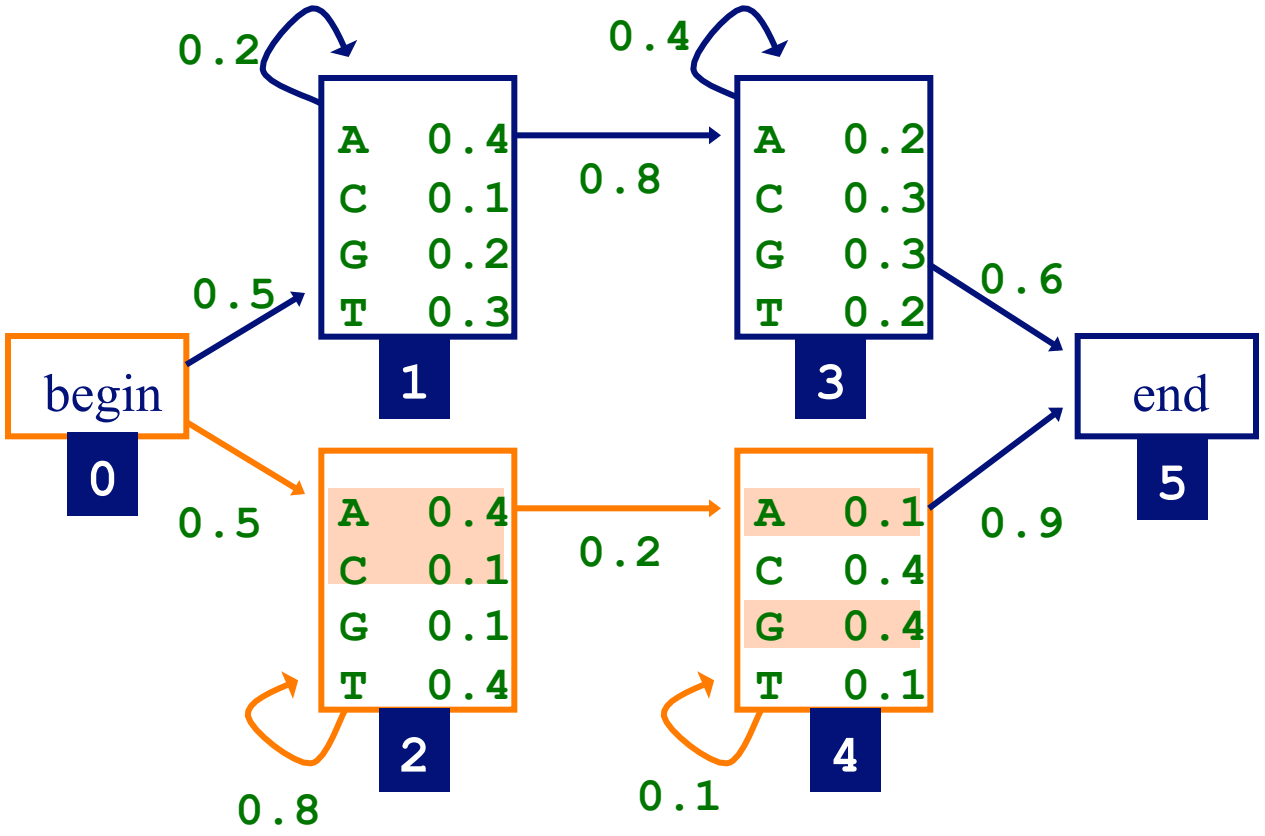
The Expectation Step

- we want to know the probability of producing sequence x with the i th symbol being produced by state k (for all x, i and k)



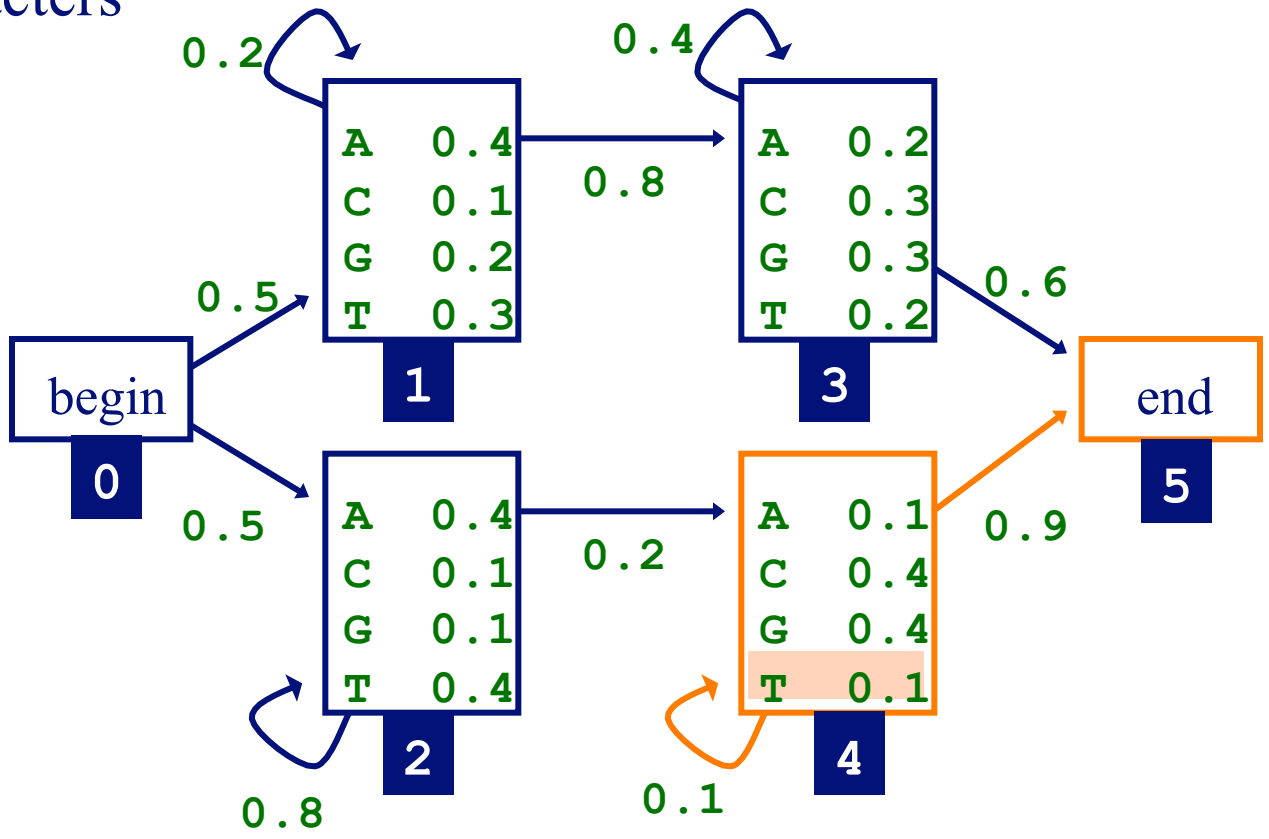
The Expectation Step

- the forward algorithm gives us $f_k(i)$, the probability of being in state *k* having observed the first *i* characters of *x*



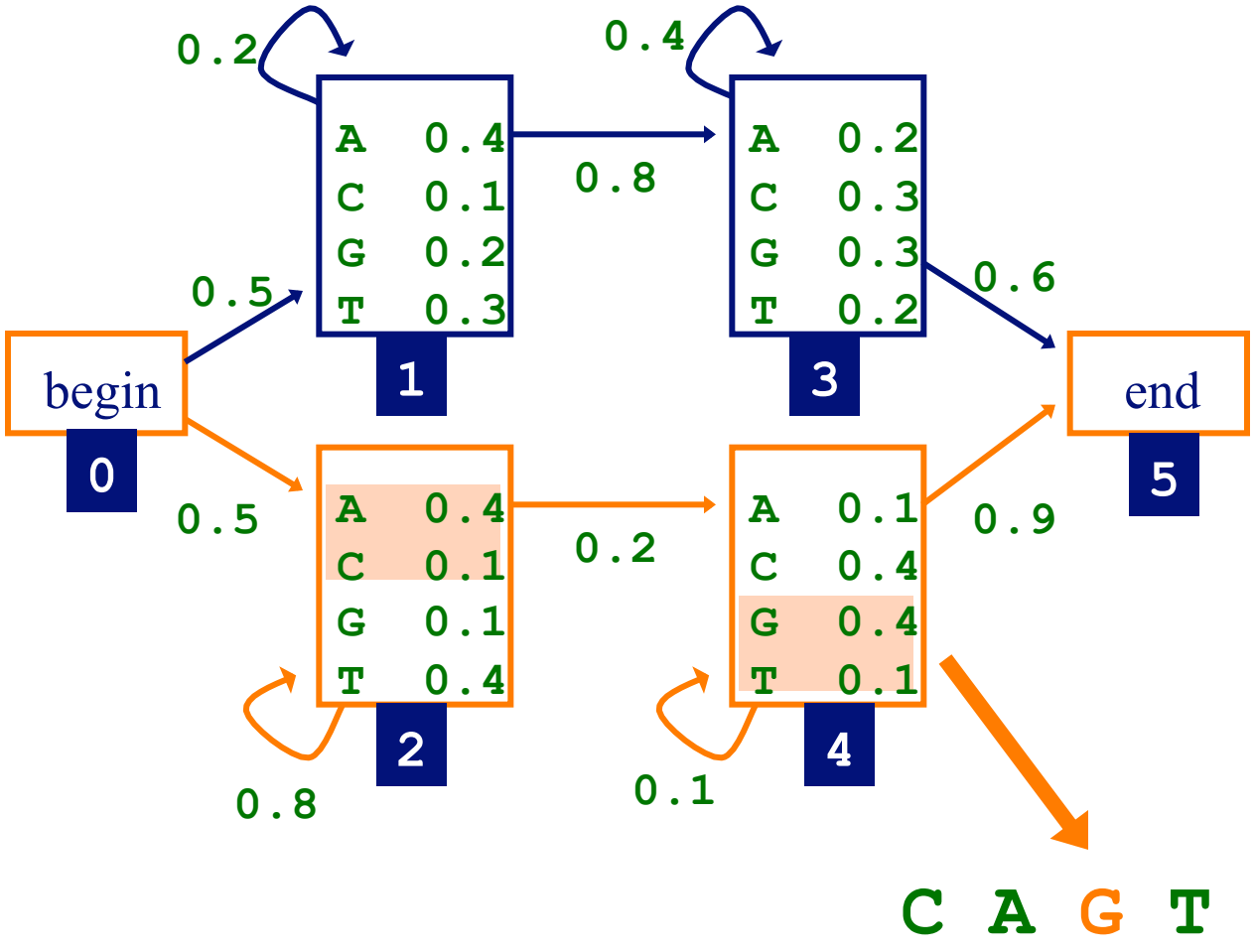
The Expectation Step

- the *backward algorithm* gives us $b_k(i)$, the probability of observing the rest of x, given that we're in state *k* after *i* characters



The Expectation Step

- putting forward and backward together, we can compute the probability of producing sequence x with the i th symbol being produced by state k



The Backward Algorithm

- initialization:

$$b_k(L) = a_{kN}$$

for states with a transition to *end* state

The Backward Algorithm

- recursion ($i = L \ldots 1$):

$$b_k(i) = \sum_l \begin{cases} a_{kl} b_l(i), & \text{if } l \text{ is silent state} \\ a_{kl} e_l(x_{i+1}) b_l(i+1), & \text{otherwise} \end{cases}$$

The Backward Algorithm

- termination:

$$\Pr(x) = \Pr(x_1 \ldots x_L) = \sum_l \begin{cases} a_{0l} b_l(0), & \text{if } l \text{ is silent state} \\ a_{0l} e_l(x_1) b_l(1), & \text{otherwise} \end{cases}$$

The Expectation Step

- now we can calculate the probability of the *i* th symbol being produced by state *k*, given *x*

$$\Pr(\pi_i = k \mid x) = \frac{\Pr(\pi_i = k, x)}{\Pr(x)}$$

$$= \frac{f_k(i) b_k(i)}{\Pr(x)}$$

$$= \frac{f_k(i) b_k(i)}{f_N(L)}$$

The Expectation Step

- now we can calculate the expected number of times letter *c* is emitted by state *k*
- here we've added the superscript *j* to refer to a specific sequence in the training set

$$n_{k,c} = \sum_{x^j} \left[\frac{1}{f_N^j(L)} \sum_{\{i|x_i^j = c\}} f_k^j(i) b_k^j(i) \right]$$

sum over sequences

sum over positions where *c* occurs in x

The Expectation Step

- and we can calculate the expected number of times that the transition from *k* to *l* is used

$$n_{k\to l} = \sum_{x^j} \frac{\sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i+1)}{f_N^j(L)}$$

- or if *l* is a silent state

$$n_{k\to l} = \sum_{x^j} \frac{\sum_i f_k^j(i) a_{kl} b_l^j(i)}{f_N^j(L)}$$

The Maximization Step

- Let $n_{k,c}$ be the expected number of emissions of c from state k for the training set
- estimate new emission parameters by:

$$e_k(c) = \frac{n_{k,c}}{\sum_{c'} n_{k,c'}}$$

- just like in the simple case
- but typically we'll do some "smoothing" (e.g. add pseudocounts)

The Maximization Step

- let $n_{k \to l}$ be the expected number of transitions from state k to state l for the training set
- estimate new transition parameters by:

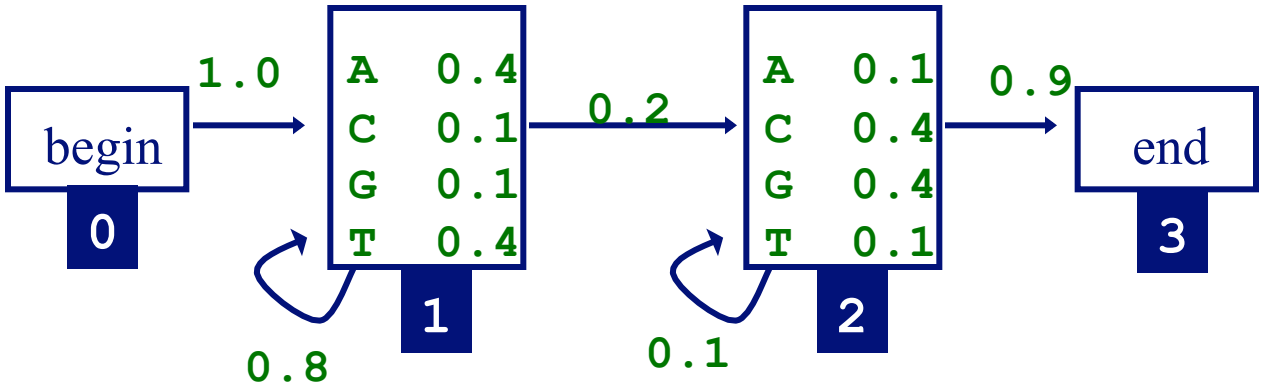
$$a_{kl} = \frac{n_{k \to l}}{\sum_{m} n_{k \to m}}$$

The Baum-Welch Algorithm

- initialize the parameters of the HMM
- iterate until convergence
  - initialize $n_{k,c}$, $n_{k \to l}$ with pseudocounts
  - **E-step**: for each training set sequence $j = 1 \ldots n$
    - calculate $f_k(i)$ values for sequence j
    - calculate $b_k(i)$ values for sequence j
    - add the contribution of sequence j to $n_{k,c}$, $n_{k \to l}$
  - **M-step**: update the HMM parameters using $n_{k,c}$, $n_{k \to l}$

Baum-Welch Algorithm Example

- given
  - the HMM with the parameters initialized as shown
  - the training sequences **TAG, ACG**



- we'll work through one iteration of Baum-Welch

Baum-Welch Example (Cont)

- determining the forward values for TAG

$$f_0(0) = 1$$

$$f_1(1) = e_1(T) \times a_{01} \times f_0(0) = 0.4 \times 1 = 0.4$$

$$f_1(2) = e_1(A) \times a_{11} \times f_1(1) = 0.4 \times 0.8 \times 0.4 = 0.128$$

$$f_2(2) = e_2(A) \times a_{12} \times f_1(1) = 0.1 \times 0.2 \times 0.4 = 0.008$$

$$f_2(3) = e_2(G) \times (a_{12} \times f_1(2) + a_{22} \times f_2(2)) =$$

$$0.4 \times (0.0008 + 0.0256) = 0.01056$$

$$f_3(3) = a_{23} \times f_2(3) = 0.9 \times 0.01056 = 0.009504$$

- here we compute just the values that represent events with non-zero probability
- in a similar way, we also compute forward values for ACG

Baum-Welch Example (Cont)

- determining the backward values for TAG

$$b_3(3) = 1$$

$$b_2(3) = a_{23} \times b_3(3) = 0.9 \times 1 = 0.9$$

$$b_2(2) = a_{22} \times e_2(G) \times b_2(3) = 0.1 \times 0.4 \times 0.9 = 0.036$$

$$b_1(2) = a_{12} \times e_2(G) \times b_2(3) = 0.2 \times 0.4 \times 0.9 = 0.072$$

$$b_1(1) = a_{11} \times e_1(A) \times b_1(2) + a_{12} \times e_2(A) \times b_2(2) =$$

$$0.8 \times 0.4 \times 0.072 + 0.2 \times 0.1 \times 0.036 = 0.02376$$

$$b_0(0) = a_{01} \times e_1(T) \times b_1(1) = 1.0 \times 0.4 \times 0.02376 = 0.009504$$

- here we compute just the values that represent events with non-zero probability
- in a similar way, we also compute backward values for ACG

Baum-Welch Example (Cont)

- determining the expected emission counts for state 1

contribution of TAG | contribution of ACG | pseudocount

$$n_{1,A} = \frac{f_1(2)b_1(2)}{f_3(3)} + \frac{f_1(1)b_1(1)}{f_3(3)} + 1$$

$$n_{1,C} = \frac{f_1(2)b_1(2)}{f_3(3)} + 1$$

$$n_{1,G} = 1$$

$$n_{1,T} = \frac{f_1(1)b_1(1)}{f_3(3)} + 1$$

*note that the forward/backward values in these two columns differ; in each column they are computed for the sequence associated with the column

Baum-Welch Example (Cont)

- determining the expected transition counts for state 1 (not using pseudocounts)

contribution of TAG

contribution of ACG

$$n_{1\to 1} = \frac{f_1(1)a_{11}e_1(A)b_1(2)}{f_3(3)} + \frac{f_1(1)a_{11}e_1(C)b_1(2)}{f_3(3)}$$

$$n_{1\to 2} = \frac{f_1(1)a_{12}e_2(A)b_2(2) + f_1(2)a_{12}e_2(G)b_2(3)}{f_3(3)} + \frac{f_1(1)a_{12}e_2(C)b_2(2) + f_1(2)a_{12}e_2(G)b_2(3)}{f_3(3)}$$

- in a similar way, we also determine the expected emission/ transition counts for state 2

Baum-Welch Example (Cont)

- determining probabilities for state 1

$$e_1(A) = \frac{n_{1,A}}{n_{1,A} + n_{1,C} + n_{1,G} + n_{1,T}}$$

$$e_1(C) = \frac{n_{1,C}}{n_{1,A} + n_{1,C} + n_{1,G} + n_{1,T}}$$

$$\vdots$$

$$a_{11} = \frac{n_{1\to 1}}{n_{1\to 1} + n_{1\to 2}}$$

$$a_{12} = \frac{n_{1\to 2}}{n_{1\to 1} + n_{1\to 2}}$$

Computational Complexity of HMM Algorithms

- given an HMM with N states and a sequence of length L, the time complexity of the Forward, Backward and Viterbi algorithms is

$$O(N^2 L)$$

  - this assumes that the states are densely interconnected
- Given M sequences of length L, the time complexity of Baum-Welch on each iteration is

$$O(MN^2 L)$$

Baum-Welch Convergence

- some convergence criteria
  - likelihood of the training sequences changes little
  - fixed number of iterations reached
- usually converges in a small number of iterations
- will converge to a *local* maximum (in the likelihood of the data given the model)

$$\log \Pr(\text{sequences} \mid \theta) = \sum_{x^j} \log \Pr(x^j \mid \theta)$$

parameters

Learning and Prediction Tasks

- *learning*

  Given: a model, a set of training sequences

  Do: find model parameters that explain the training sequences with relatively high probability (goal is to find a model that *generalizes* well to sequences we haven't seen before)

- *classification*

  Given: a set of models representing different sequence classes, a test sequence

  Do: determine which model/class best explains the sequence

- *segmentation*

  Given: a model representing different sequence classes, a test sequence

  Do: segment the sequence into subsequences, predicting the class of each subsequence

Algorithms for Learning and Prediction Tasks

- *learning*

  correct path known for each training sequence ⇒ simple maximum-likelihood or Bayesian estimation

  correct path not known ⇒ Forward-Backward algorithm + (ML or Bayesian estimation)

- *classification*

  simple Markov model ⇒ calculate probability of sequence along single path for each model

  hidden Markov model ⇒ Forward algorithm to calculate probability of sequence along all paths for each model

- *segmentation*

  hidden Markov model ⇒ Viterbi algorithm to find most probable path for sequence

Assessing the Accuracy of a Trained Model

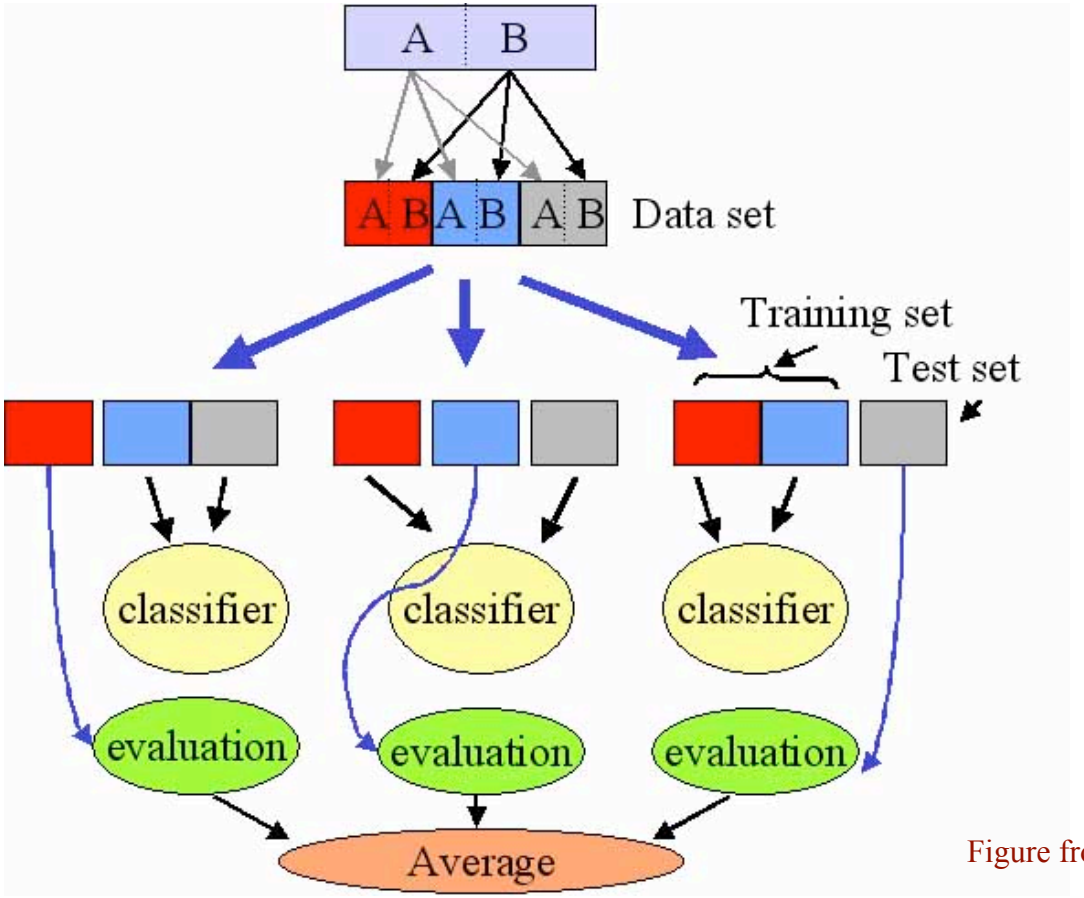
- two issues
  - What data should we use?
  - Which metrics should we use?

- Can we measure accuracy on the data set that was used to train the model?

  NO! This will result in accuracy estimates that are biased (too high).

Assessing the Accuracy of a Trained Model

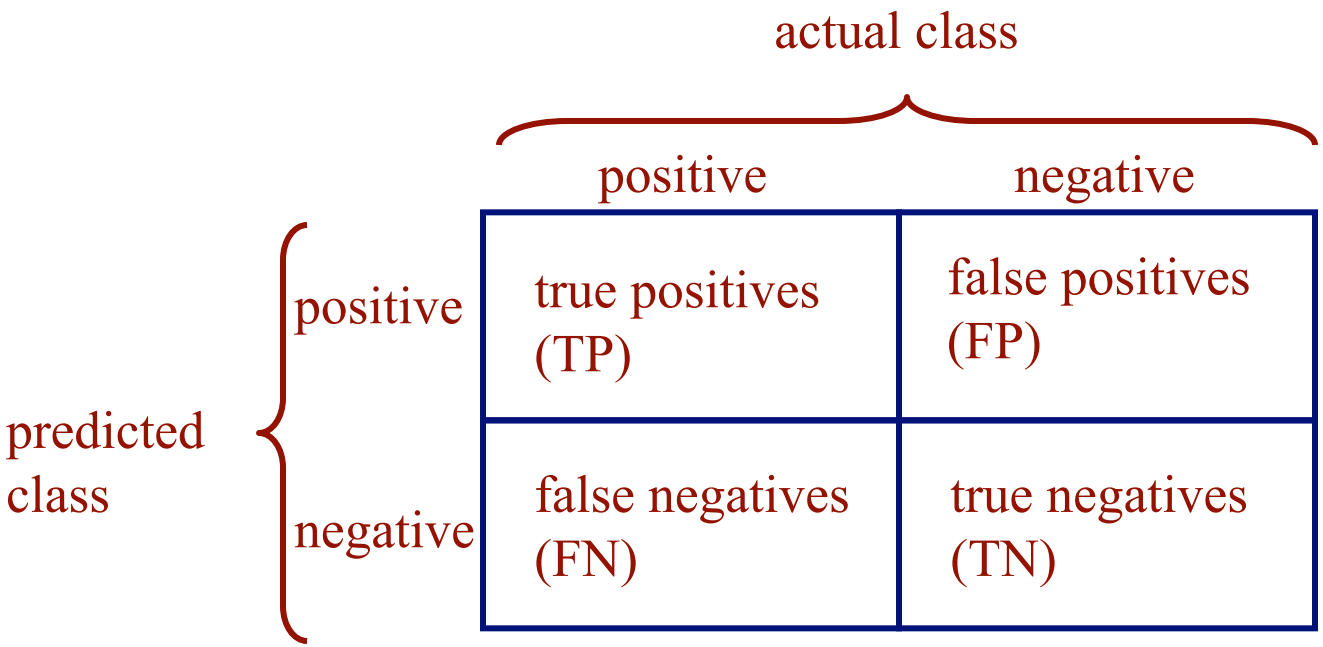
- need to have a *test set* that is disjoint from the training set
- more generally, can use *cross validation*



3-fold CV illustrated

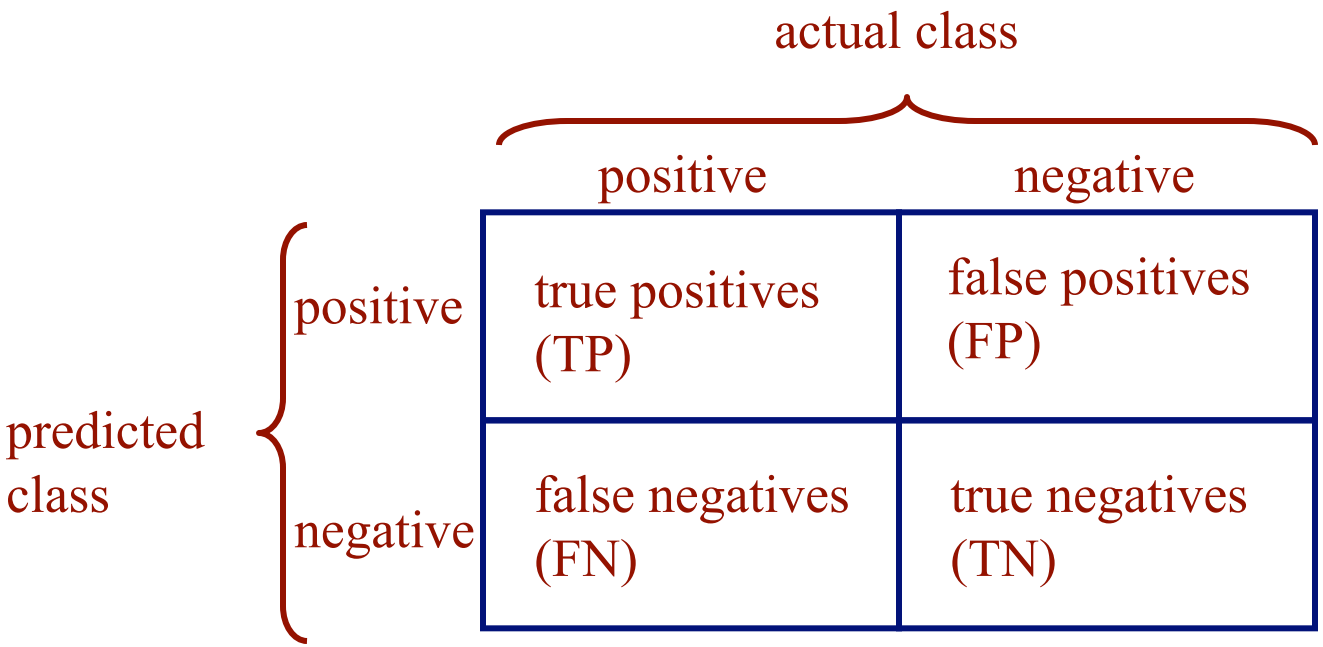
Figure from http://gepas.bioinfo.cipf.es/

Accuracy



$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy Metrics



$$\text{sensitivity (recall)} = \frac{TP}{\text{actual pos}} = \frac{TP}{TP + FN}$$

$$\text{specificity} = \frac{TN}{\text{actual neg}} = \frac{TN}{TN + FP}$$

$$\text{precision} = \frac{TP}{\text{predicted pos}} = \frac{TP}{TP + FP}$$

sometimes specificity is defined this way