

# Hidden Markov Models (Part 2)

BMI/CS 576

[www.biostat.wisc.edu/bmi576.html](http://www.biostat.wisc.edu/bmi576.html)

Mark Craven

[craven@biostat.wisc.edu](mailto:craven@biostat.wisc.edu)

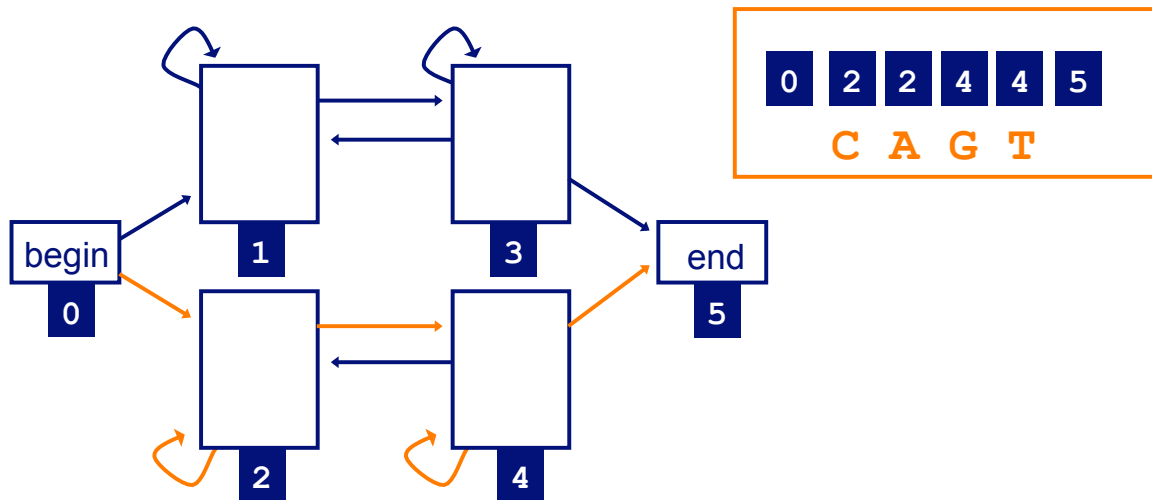
Fall 2011

## 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 information

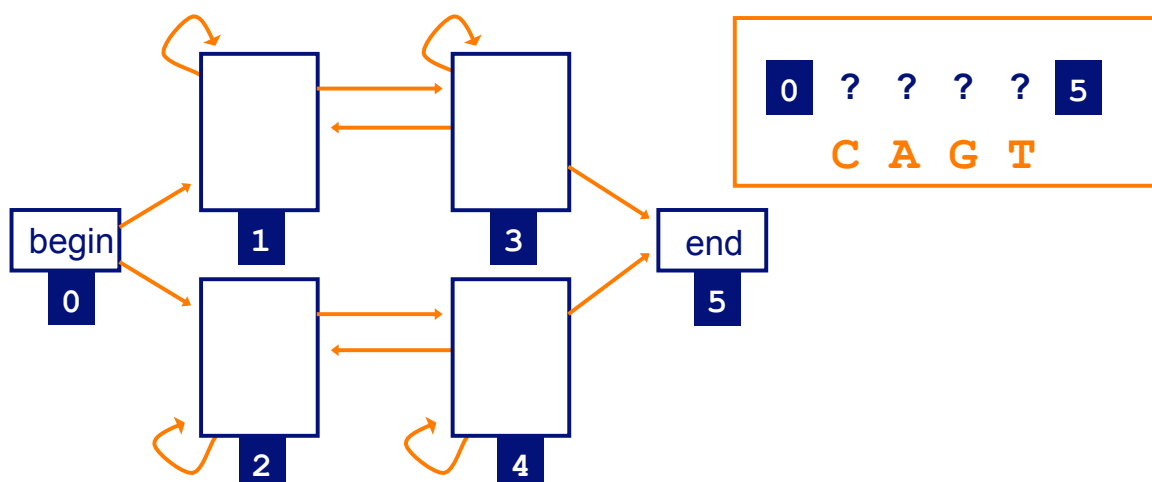
- learning is simple if we know the correct path for each sequence in our training set



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

## Learning with hidden information

- 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 parameter is used across the training set

## Learning parameters

- if we know the state path for each training sequence, learning the model parameters is simple
  - no hidden information during training
  - count how often each parameter is used
  - 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

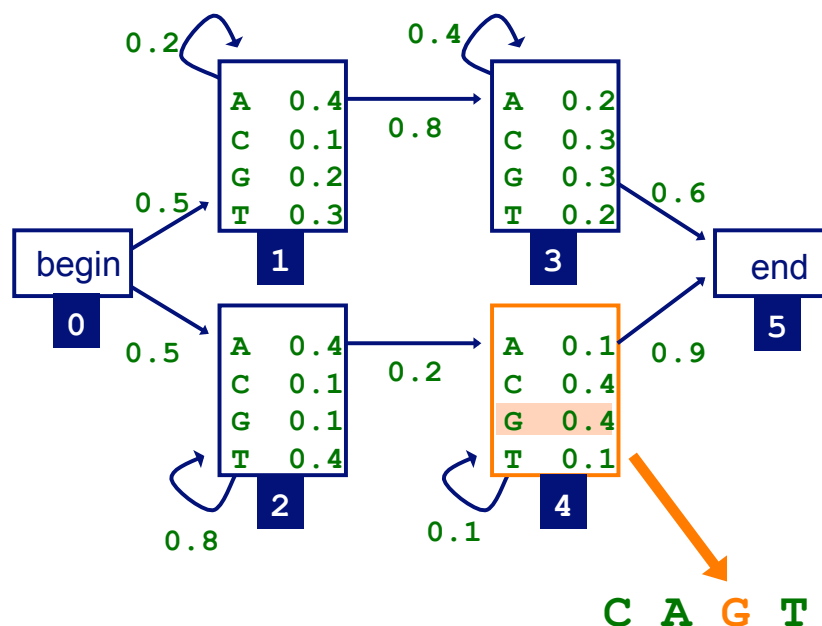
- *a.k.a* the Forward-Backward algorithm
- an *Expectation Maximization* (EM) algorithm
  - EM is a family of algorithms for learning probabilistic models in problems that involve hidden information
- in this context, the hidden information is the path that best explains each training sequence

# 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

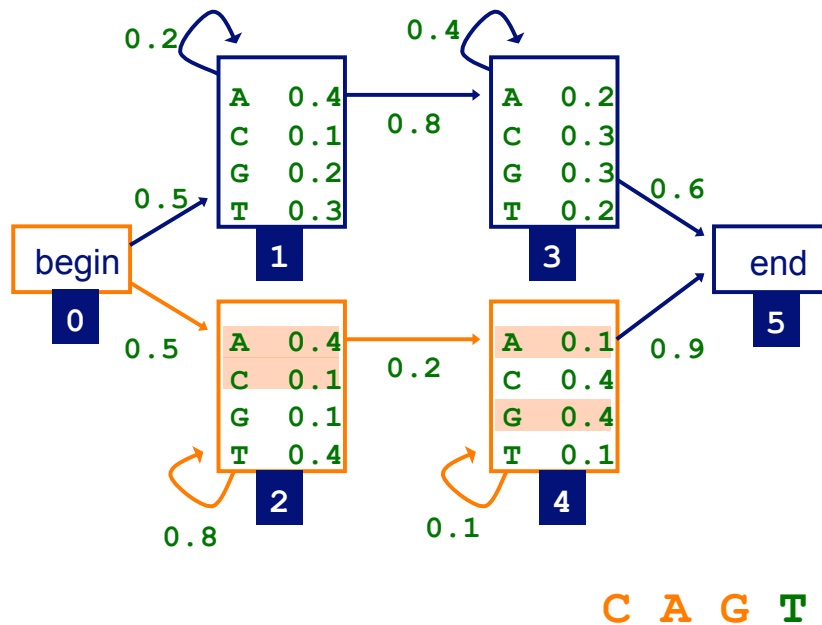
## The expectation step

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



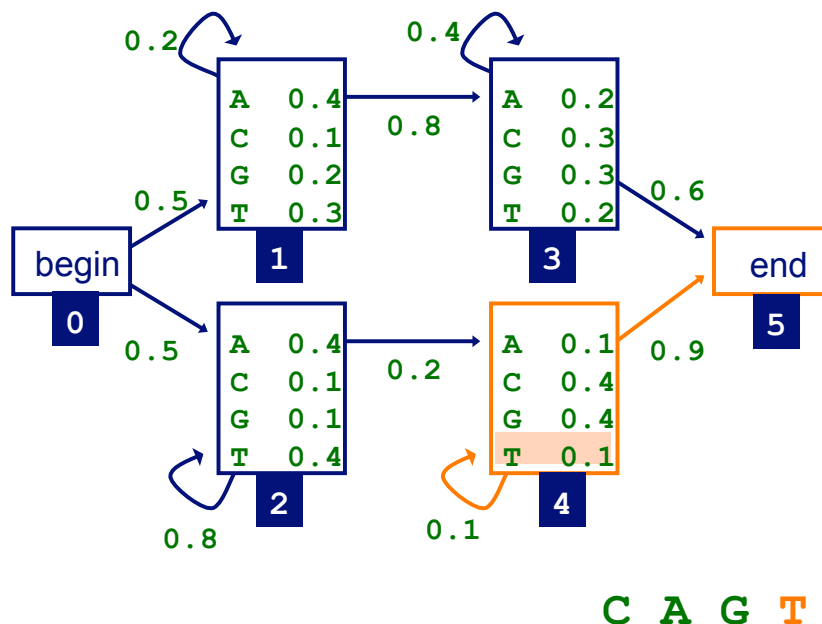
## The expectation step

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



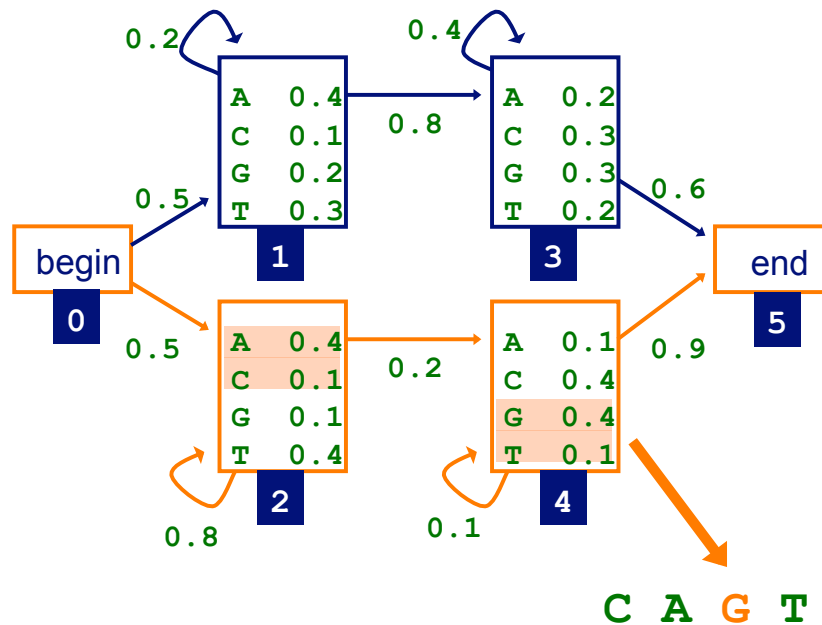
## The expectation step

- 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

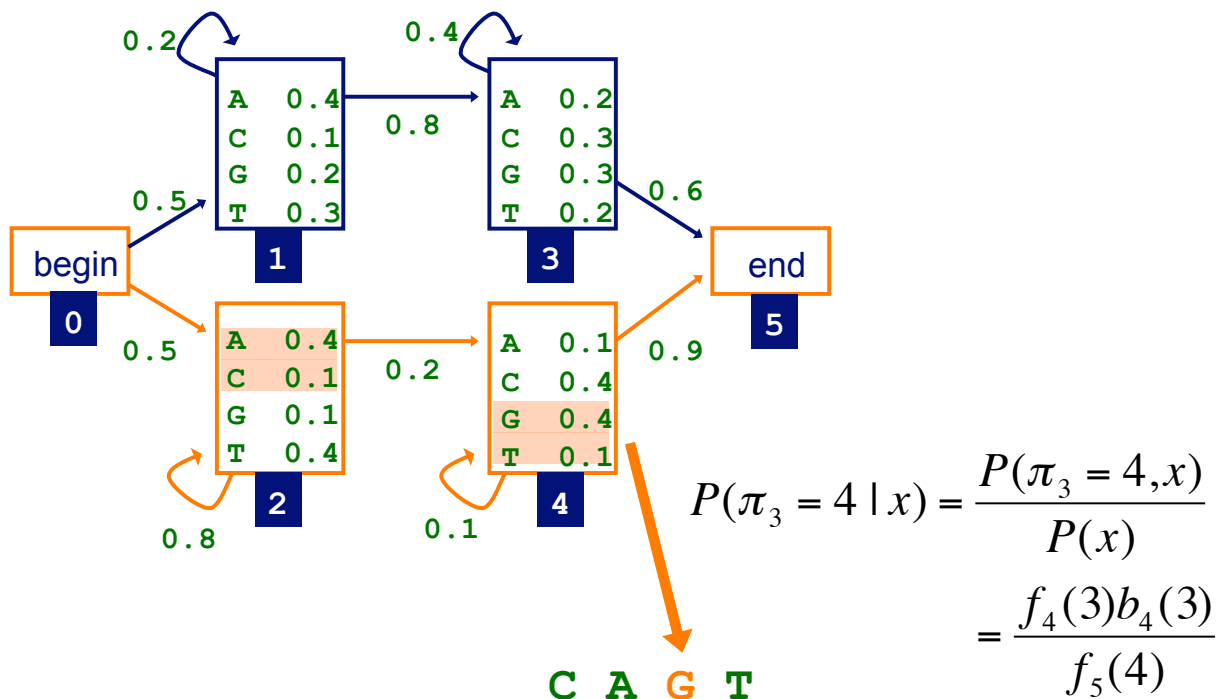


## The expectation step

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



## The expectation step



## The expectation step

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

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

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

## The expectation step

- the probability of producing  $x$  with the  $i$  th symbol being emitted by state  $k$  is

$$P(\pi_i = k, x) = P(x_1 \dots x_i, \pi_i = k) \times P(x_{i+1} \dots x_L | \pi_i = k)$$

- 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 backward algorithm

- initialization:

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

for states with a transition to *end* state

## The backward algorithm

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

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

## The backward algorithm

- termination:

$$P(x_1 \dots x_L) = b_0(0) = \sum_l \left\{ \begin{array}{ll} a_{0l} b_l(0), & \text{if } l \text{ is silent state} \\ a_{0l} e_l(x_1) b_l(1), & \text{otherwise} \end{array} \right\}$$

## The expectation step

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

$$\begin{aligned} P(\pi_i = k \mid x) &= \frac{P(\pi_i = k, x)}{P(x)} \\ &= \frac{f_k(i) b_k(i)}{P(x)} \\ &= \frac{f_k(i) b_k(i)}{f_N(L)} \end{aligned}$$

## 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

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

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

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

- or if  $l$  is a silent state

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

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

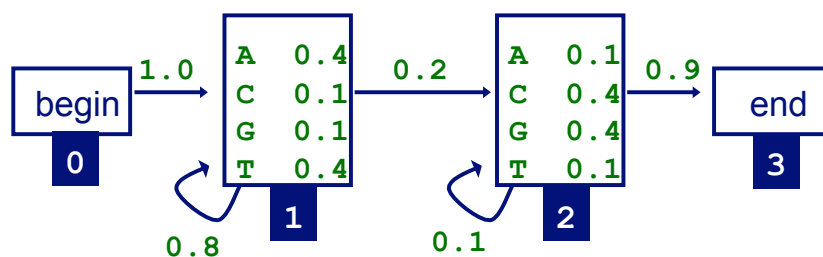
$$a_{kl} = \frac{n_{k \rightarrow l}}{\sum_m n_{k \rightarrow m}}$$

# The Baum-Welch algorithm

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

## 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
$n_{1 \rightarrow 1} =$	$\frac{f_1(1)a_{11}e_1(A)b_1(2)}{f_3(3)}$		$\frac{f_1(1)a_{11}e_1(C)b_1(2)}{f_3(3)}$
$n_{1 \rightarrow 2} =$	$\frac{f_1(1)a_{12}e_2(A)b_2(2) + f_1(2)a_{12}e_2(G)b_2(3)}{f_3(3)}$		$\frac{f_1(1)a_{12}e_2(C)b_2(2) + f_1(2)a_{12}e_2(G)b_2(3)}{f_3(3)}$

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

## Baum-Welch example (cont)

- determining probabilities for state 1

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

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

⋮

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

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

## Computational complexity of HMM algorithms

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

$$O(S^2L)$$

– this assumes that the states are densely interconnected

- Given  $M$  sequences of length  $L$ , the complexity of Baum-Welch on each iteration is

$$O(MS^2L)$$

## 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)

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

## 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  $\Rightarrow$  simple maximum-likelihood or Bayesian estimation
  - correct path not known  $\Rightarrow$  Forward-Backward algorithm + (ML or Bayesian estimation)
- *classification*
  - simple Markov model  $\Rightarrow$  calculate probability of sequence along single path for each model
  - hidden Markov model  $\Rightarrow$  Forward algorithm to calculate probability of sequence along all paths for each model
- *segmentation*
  - hidden Markov model  $\Rightarrow$  Viterbi algorithm to find most probable path for sequence