Introduction to Hidden Markov Models

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

2024-07-10

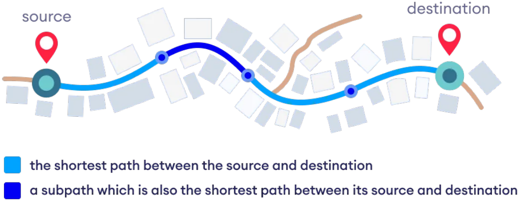
GitHub Repository

Please note the following GitHub Repository for all materials for this tutorial:

https://github.com/wevanjohnson/2024_07_hmm_tutorial

Introduction to Dynamic Programming

Before we study **Hidden Markov Models (HMMs)** it is necessary to begin with an introduction to **dynamic programming (DP)**.


source
destination
🟦 the shortest path between the source and destination
🟦 a subpath which is also the shortest path between its source and destination

Introduction to Dynamic Programming

An instructive example for the use of DP deals with finding the optimal alignment between two DNA strands.

Consider two DNA sequences, $x = \{\mathrm{C\ A\ G\ A\ T\ G\ A\ G\ C\ A}\}$ and $y = \{\mathrm{T\ C\ A\ A\ T\ G\ A\ A\ C\ A}\}$, we note how many times $x_i = y_i$, where x_i is the ith element of x.

Ungapped Alignment

We observe the **ungapped alignment**, which has 6 matches and 4 mismatches:

```
C A G A T G A G C A
      | | | |   | |
T C A A T G A A C A
```

Gapped Alignment

For a **gapped alignment**, denoted by '$-$' for the gaps, to be inserted into the sequences then we can create more matches, in this case 8 matches:

$$\begin{array}{ccccccccccc} - & \mathrm{C} & \mathrm{A} & \mathrm{G} & \mathrm{A} & \mathrm{T} & \mathrm{G} & \mathrm{A} & \mathrm{G} & \mathrm{C} & \mathrm{A} \\ & | & | & & | & | & | & | & & | & | \\ \mathrm{T} & \mathrm{C} & \mathrm{A} & - & \mathrm{A} & \mathrm{T} & \mathrm{G} & \mathrm{A} & \mathrm{A} & \mathrm{C} & \mathrm{A} \end{array}$$

Gapped Alignment

We will denote the set of all gapped sequences of a sequence x by $\mathcal{A}_x$, and thus $x \in \mathcal{A}_x$.

Scoring matches, mismatches, gaps

Suppose we are interested in finding the optimal gapped alignment of two sequences x and y using the scoring function S given by

$$S(x_i, y_i) = \begin{cases} 1 & \text{if } x_i = y_i \\ -1 & \text{if } x_i \neq y_i \\ -2 & \text{if } x_i = - \text{ or } y_i = - \end{cases}$$

Scoring matches, mismatches, gaps

Then the optimal alignment is given by solving

$$\begin{aligned} \max_{(\hat{x},\hat{y}) \in \mathcal{A}_x \times \mathcal{A}_y} \quad & \left\{ \sum_{i}^{len(\hat{x})} S(\hat{x}_i, \hat{y}_i) \right\} \\ \text{s.t.} \quad & len(\hat{x}) = len(\hat{y}) \end{aligned}$$

where $len(x)$ denotes the number of elements in x.

Note: that we do not require $len(x) = len(y)$ but only $len(\hat{x}) = len(\hat{y})$.

Scoring matches, mismatches, gaps

This alignment (shown previously) is the solution to the system given by (1) with a score of 3:

$$\begin{array}{ccccccccccc} - & \mathrm{C} & \mathrm{A} & \mathrm{G} & \mathrm{A} & \mathrm{T} & \mathrm{G} & \mathrm{A} & \mathrm{G} & \mathrm{C} & \mathrm{A} \\ & | & | & & | & | & | & | & & | & | \\ \mathrm{T} & \mathrm{C} & \mathrm{A} & - & \mathrm{A} & \mathrm{T} & \mathrm{G} & \mathrm{A} & \mathrm{A} & \mathrm{C} & \mathrm{A} \end{array}$$

Finding the optimal alignment

One way to solve this system for any given x and y is to check the score of all possible gapped alignments by considering every $\hat{x}$ and $\hat{y}$ with the same length.

This method is obviously computationally inefficient. Dynamic programming allows us to eliminate suboptimal alignments by breaking solving subproblems of the original problem.

Needleman-Wunsch Algorithm

Given two sequences x and y let $len(x) = n$ and $len(y) = m$. We will let F be an $(n + 1) \times (m + 1)$ matrix where F(i,j) represents the score of the optimal alignment for $x_1, x_2, \ldots, x_{i-1}$ and $y_1, y_2, \ldots, y_{j-1}$.

Needleman-Wunsch Algorithm

Since we can get the first column and row of F we can find the rest of the entries by solving the **recursive equation** given by

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + S(x_i, y_i) \\ F(i-1,j) - 2 \\ F(i,j-1) - 2 \end{cases} .$$

Needleman-Wunsch Algorithm

In summary the Needleman-Wunsch algorithm is outlined in the following steps.

1. Initialization: Define $(n+1) \times (m+1)$ matrix F.
    1.1 $F(i,1) = -2(i-1)$ for $i = 1, \dots, n+1$
    1.2 $F(1,j) = -2(j-1)$ for $j = 2, \dots, m+1$
2. Recursion: Let $F(i,j) = \max\{F(i-1,j-1) + S(x_i, y_i), F(i-1,j) - 2, F(i,j-1) - 2\}$ for $i = 2, \dots, n+1$ and $j = 2, \dots, m+1$.
3. Trace back: Record where you came from starting at $F(n+1, m+1)$.

Needleman-Wunsch Example

Let $x = \{\mathrm{T\ A\ C\ A\ T\ G\ C}\}$ and $y = \{\mathrm{T\ T\ C\ A\ G\ C}\}$, and use the scoring function S given by

$$S(x_i, y_i) = \begin{cases} 1 & \text{if } x_i = y_i \\ -1 & \text{if } x_i \neq y_i \\ -2 & \text{if } x_i = - \text{ or } y_i = - \end{cases}$$

and the Needleman-Wunsch algorithm to find the optimal alignment.

Needleman-Wunsch Example

First We will define the following DP matrix:

		t	a	c	a	t	g	c
t								
t								
c								
a								
g								
c								

Needleman-Wunsch Example

Then completing the margins using the Needleman-Wunsch scoring:

		t	a	c	a	t	g	c
	0	-2	-4	-6	-8	-10	-12	-14
t	-2							
t	-4							
c	-6							
a	-8							
g	-10							
c	-12							

Needleman-Wunsch Example

And finishing the entire matrix:

		t	a	c	a	t	g	c
	0	-2	-4	-6	-8	-10	-12	-14
t	-2	**1**	-1	-3	-5	-7	-9	-11
t	-4	-1	**0**	-2	-4	-4	-6	-8
c	-6	-3	-2	**1**	-1	-3	-3	-5
a	-8	-5	-2	-1	**2**	**0**	-2	-2
g	-10	-7	-4	-3	0	1	**1**	-1
c	-12	-7	-6	-3	-2	1	0	**2**

Needleman-Wunsch Algorithm

The optimal alignment is then found by **back-tracing** the source of the optimal subscores (i.e. outlined in black), i.e.,

$$\begin{array}{ccccccc} \mathsf{T} & \mathsf{A} & \mathsf{C} & \mathsf{A} & \mathsf{T} & \mathsf{G} & \mathsf{C} \\ | & & | & | & & | & | \\ \mathsf{T} & \mathsf{T} & \mathsf{C} & \mathsf{A} & - & \mathsf{G} & \mathsf{C} \end{array}$$

Smith-Waterman Algorithm

So far we have assumed we know which sequences we want to align. Suppose now that we want to find the optimal alignment between subsequences of x and y. The optimal alignment of subsequences of two sequences is called the best **local alignment**. This algorithm is nearly the same as Needleman-Wunsch with just a couple differences.

Smith-Waterman Algorithm

The **first difference** is that

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1,j-1) + S(x_i, y_i) \\ F(i-1,j) - 2 \\ F(i,j-1) - 2 \end{cases} .$$

The **second difference** is that the trace back step starts at the highest value of F. From there you trace back in the DP matrix until the first 0 is found. This will yield the best local alignment of x and y.

Smith-Waterman Example

Using the previous sequences and matrix:

		t	a	c	a	t	g	c
	0	0	0	0	0	0	0	0
t	0	1	0	0	0	1	0	0
t	0	1	0	0	0	1	0	0
c	0	0	0	**1**	0	0	0	1
a	0	0	1	0	**2**	0	0	0
g	0	0	0	0	0	1	**1**	0
c	0	0	0	1	0	0	0	**2**

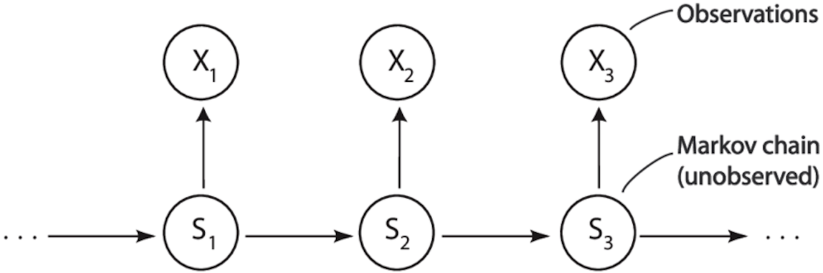
Smith-Waterman Example

Now **back-tracing** from the maximum (i.e. outlined in black), yields multiple three possible local alignments

```
T   A   C   A   T   G   C
|       |   |       |   |
T   T   C   A   –   G   C
```

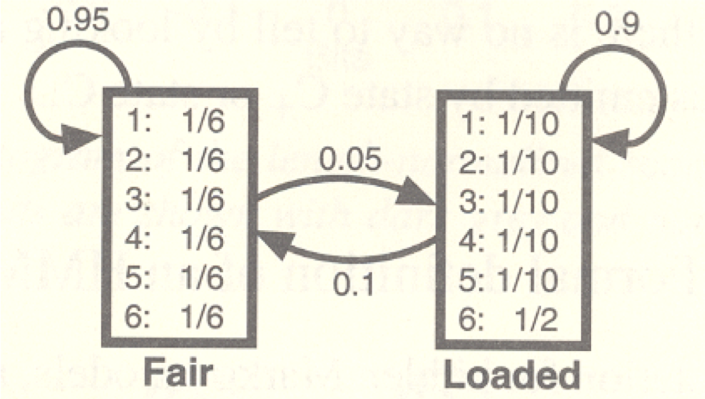
Hidden Markov Models

A HMM is a statistical model where the system is assumed to be a Markov process but the state is not observed. However, one does observe some kind of outcome that comes from being in a state. We call this outcome an **emission**.



Example: Occasionally Dishonest Casino

Suppose that a casino usually uses a fair die but occasionally switches a loaded die using the following process:



Durbin et al., Cambridge University Press. https://doi.org/10.1017/CBO9780511790492.004

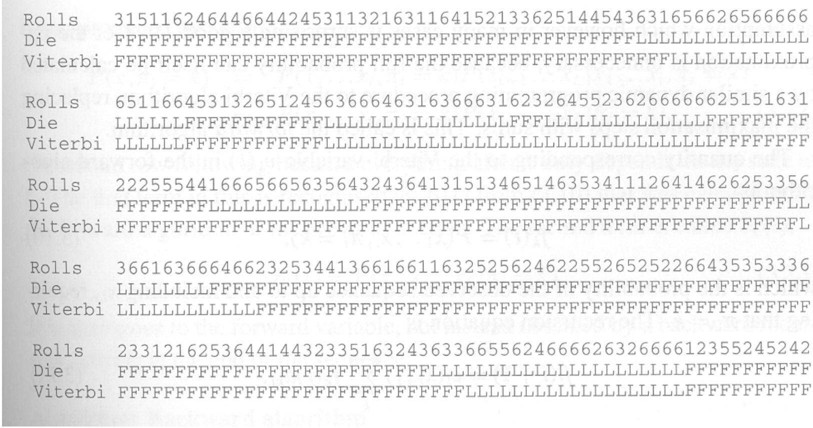
Example: Occasionally Dishonest Casino

The observed *emissions* for this example are a sequence of die rolls.

The hidden process or sequence of *hidden states* consist of which die is being used. We do not observe which die but we assume we know the transition probabilities of the states.

Example: Occasionally Dishonest Casino

The goal will be to estimate or infer the sequence sequence of *hidden states* (which die is being rolled) from the emissions (observed die rolls):



```
Rolls   315116246446644245311321631164152133625144543631656626566666
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLL

Rolls   651166453132651245636664631636663162326455236266666625151631
Die     LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLFFFLLLLLLLLLLLLLLFFFFFFFFF
Viterbi LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFF

Rolls   222555441666566563564324364131513465146353411126414626253356
Die     FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls   366163666466232534413661661163252562462255265252266435353336
Die     LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   233121625364414432335163243633665562466662632666612355245242
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
```

Durbin et al., Cambridge University Press. https://doi.org/10.1017/CBO9780511790492.004

Hidden Markov Models

Formally, let $\mathbf{x} = x_1, x_2, \ldots, x_n$ be the sequence of emissions observed (value of die roll), and let $\mathbf{y} = y_1, y_2, \ldots, y_n$ be the sequence of unobserved states that follow a Markov Process (which die is used).

Let our (known) **transition probabilities** be denoted as

$$T_{k,i} = P(y_{i+1} = i \mid y_i = k),$$

and our (known) **emission probabilities** be denoted by

$$e_i(x_j) = P(x_j \mid y_j = i)$$

.

Viterbi Algorithm

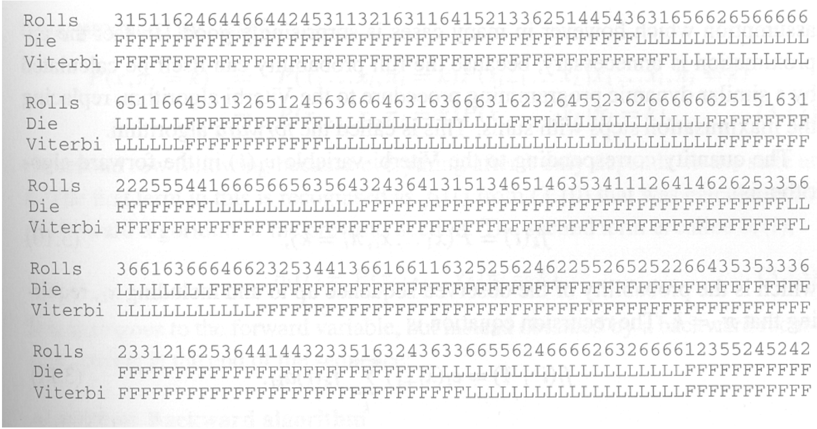
The **Viterbi algorithm** provides one way to solve this problem. It is a dynamic programming algorithm that considers all possible routes to a state but only keeps the most likely one.

Viterbi Algorithm

Let V be a $s \times (n+1)$ matrix be our dynamic programming matrix, defined as follows:

1. Initialization: Define V as a $s \times (n+1)$ matrix.
   1.1 $V(1,0) = 1$;
   1.2 $V(i,0) = 0$ for $i = 2, \ldots, s$
2. Recursion: for $j = 1, \ldots, n$ and $i = 1, \ldots, s$
   2.1 $V(i,j) = e_i(x_j) \max_k \{V(k,,j-1) T_{k,i}\}$
   2.2 Each time before iterating from j to $j+1$ you must scale the jth column of V so that the column sums to 1.
   2.3 Trace back: record $\sup_k \{V(k, j-1) T_{k,i}\}$
3. Termination: Start at $\sup_k \{V(k,n)\}$ and sweep back.

Viterbi Algorithm Example



```
Rolls   315116246446644245311321631164152133625144543631656626566666
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLL

Rolls   651166453132651245636664631636663162326455236266666625151631
Die     LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLFFFLLLLLLLLLLLLLLFFFFFFFFF
Viterbi LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFF

Rolls   222555441666566563564324364131513465146353411126414626253356
Die     FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls   366163666466232534413661661163252562462255265252266435353336
Die     LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   233121625364414432335163243633665562466662632666612355245242
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
```

Durbin et al., Cambridge University Press. https://doi.org/10.1017/CBO9780511790492.004

Viterbi Algorithm Example

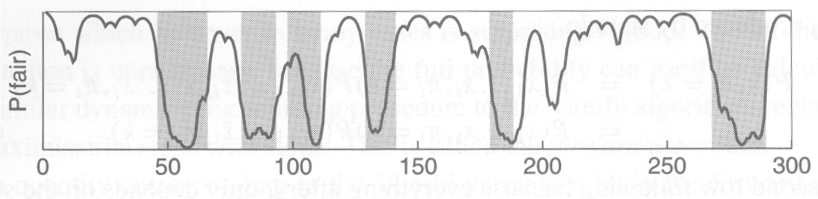
The Viterbi algorithm has advantages and disadvantages.

The *advantage* is that it gives an optimal solution and is fast!

The *disadvantage* is that it doesn't yield actual probabilities–just tells which state is most likely in.

Forward-Backward Algorithm

Instead of the Viterbi solution, often we would like to estimate the probability of the latent **state-space**. We can do this using the **forward-backward algorithm**:



Durbin et al., Cambridge University Press. https://doi.org/10.1017/CBO9780511790492.004

Forward-Backward Algorithm

Before we derive the **forward-backward algorithm**, we review some useful concepts from probability theory.

Recall the definition of conditional probability says that for two events A and B

$$\begin{aligned} P(A \cap B) &= P(A \mid B)P(B) \\ &= P(B)P(A \mid B). \end{aligned}$$

And extended to three events A, B, and C by

$$P(A \cap B \cap C) = P(A)P(B \mid A)P(C \mid B \cap A).$$

Recall also that for a discrete joint density function $f(x, y)$ the marginal density function is defined as

$$f(x) = \sum_{y} f(x, y)dy.$$

Forward-Backward Algorithm

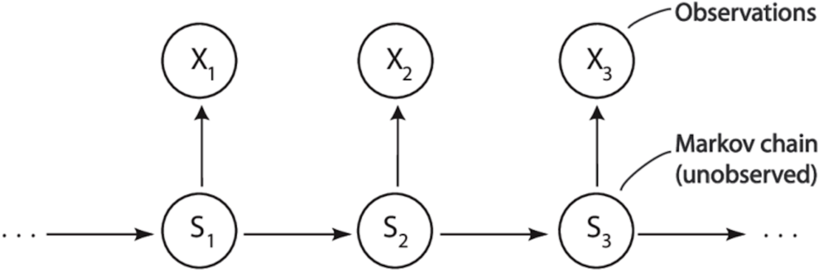
Now we will derive the forward-backward algorithm.

Let $\mathbf{x} = x_1, x_2, \ldots, x_n$ be a vector containing all the emissions. We wish to find $P(y_i = k \mid \mathbf{x})$.

Forward-Backward Algorithm

Using the definition of conditional probability we have

$$\begin{aligned} P(y_i = k \mid \mathbf{x}) &= \frac{P(y_i = k, \mathbf{x})}{P(\mathbf{x})} \\ &= \frac{P(y_i = k, x_1, \ldots, x_i) P(x_{i+1}, \ldots, x_n \mid y_i = k, x_1, \ldots, x_i)}{P(\mathbf{x})} \end{aligned}$$



Forward-Backward Algorithm

We define our **forward equation** to be:

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

and our **backward equation** to be

$$b_k(i) = P(x_{i+1}, \ldots, x_n \mid y_i = k, x_1, \ldots, x_i).$$

Note that $P(\mathbf{x})$ is just a **normalizing constant** that you can use to scale the probabilities so that $\sum_k P(y_i = k \mid \mathbf{x}) = 1$.

This gives us

$$P(y_i = k \mid \mathbf{x}) = \frac{f_k(i) b_k(i)}{P(\mathbf{x})}.$$

Forward Algorithm

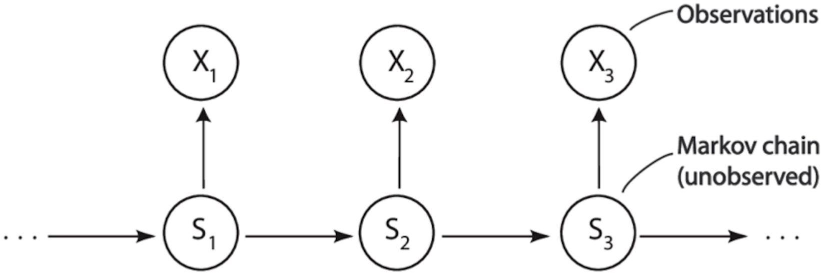
We will now derive an expression for $f_k(i)$. Using the definition of a marginal density function and conditional probability for multiple events we have that

$$\begin{aligned}
f_k(i) &= P(y_i = k, x_1, \ldots, x_i) \\
&= \sum_{j=1}^{s} P(y_i = k, x_1, \ldots, x_i, y_{i-1} = j) \\
& \text{(Assuming Markov Process)} \\
&= \sum_{j=1}^{s} P(x_i \mid y_i = k) P(y_i = k, x_1, \ldots, x_{i-1}, y_{i-1} = j)
\end{aligned}$$

Forward Algorithm

Note the last step is assuming the underlying process $y_1, \ldots, y_n$ is a Markov Process we can say that

$$P(x_i \mid y_i = k, x_1, \ldots, x_{i-1}, y_{i-1} = j) = P(x_i \mid y_i = k).$$



Forward Algorithm

This gives us

$$
\begin{aligned}
f_k(i) &= \sum_{j=1}^{s} P(x_i \mid y_i = k) P(y_i = k, x_1, \ldots, x_{i-1}, y_{i-1} = j) \\
&= \sum_{j=1}^{s} e_k(x_i) P(y_i = k \mid x_1, \ldots, x_{i-1}, y_{i-1} = j) P(x_1, \ldots, x_{i-1}, y_{i-1} = j) \\
&\quad \text{(Assuming Hidden Markov Model)} \\
&= \sum_{j=1}^{s} e_k(x_i) P(y_i = k \mid y_{i-1} = j) P(x_1, \ldots, x_{i-1}, y_{i-1} = j) \\
&= e_k(x_i) \sum_{j=1}^{s} T_{j,k} f_j(i-1)
\end{aligned}
$$

Forward Algorithm

Now that we have a recursive equation for $f_k(i)$ we have an algorithm for the forward equation.

1. Initialization: Define a $s \times (n+1)$ matrix F where $F_{i,j} = f_i(j)$.
   1.1 $f_1(1) = 1$
   1.2 $f_k(1) = 0$ for $k = 2, \ldots, s$
2. Recursion: $\hat{f}_k(j) = e_k(x_i) \sum_{i=1}^{s} T_{i,k} f_i(j-1)$ for $k = 1, \ldots, s$ and $j = 1, \ldots, n$
   2.1 Let $s(j) = \sum_{i=1}^{s} \hat{f}_i(j)$, which is our scaling factor. Before moving to the next column of the matrix by iterating j let $f_k(j) = \hat{f}_k(j)/s(j)$ for $k = 1, \ldots, s$.

Backward Algorithm

Now we derive a recursive equation for the backward equation $b_k(i)$. The derivation is very similar to the forward equation.

$$
\begin{aligned}
b_k(i) &= P(x_{i+1}, \ldots, x_n \mid y_i = k) \\
&= \sum_{j=1}^{s} P(x_{i+1}, \ldots, x_n, y_{i+1} = j \mid y_i = k) \\
&\quad \text{(Assuming Markov Process)} \\
&= \sum_{j=1}^{s} P(x_{i+1} \mid y_{i+1} = j) P(x_{i+2}, \ldots, x_n, y_{i+1} = j \mid y_i = k) \\
&= \sum_{j=1}^{s} e_j(x_{i+1}) P(x_{i+2}, \ldots, x_n \mid y_{i+1} = j, y_i = k) P(y_{i+1} = j \mid y_i = k) \\
&= \sum_{j=1}^{s} e_j(x_{i+1}) b_j(i+1) T_{k,j}
\end{aligned}
$$

Backward Algorithm

This yields the following algorithm for the backward equation.

1. Initialization: Define $s \times n$ matrix B with elements $B_{i,j} = b_i(j)$
    1.1 $b_k(n) = T_{k,1}$ for $k = 1, \ldots, s$
2. Recursion: Let $\hat{b}_k(j) = \sum_{i=1}^{s} e_i(x_{j+1}) b_i(j+1) T_{k,j}$ for $j = n-1, \ldots, 1$ and $k = 1, \ldots, s$
    2.1 Let $s(j) = \sum_{k=1}^{s} \hat{b}_k(j)$ and before each time you iterate i make sure to scale it by letting $b_k(j) = \hat{b}_k(j)/s(j)$.

Forward-Backward Algorithm for Transitions

Suppose that we are trying to estimate the states of a HMM but we don't know the transition matrix $\mathbf{T} = \{T_{k,j}\}$ or the emission probabilities $\mathbf{e} = \{e_k(x_j)\}$ or both. We will used a slightly modified forward-backward algorithm along with the **Baum-Welch algorithm**, is useful for estimating these probabilities. We start with an initial guess for your transition and emission probabilities. After several iterations of the forward-backward algorithm the Baum-Welch algorithm will help us converge to the true probabilities.

Forward-Backward Algorithm for Transitions

We let

$$a_{m,k}(j) = P(y_j = k, y_{j-1} = m \mid \mathbf{x}) = \frac{f_{m,k}(j) b_{m,k}(j)}{P(\mathbf{x})}$$

for $j = 2, \ldots, n$; $m = 1, \ldots, s$ and $k = 1, \ldots, s$.

1. Initialization: Let F be an $s^2 \times n$ matrix where $F_{i,j} = f_{m,k}(j)$. The reason for having s^2 rows is that at each time j we must consider all possible state transitions. For example, a HMM with 2 states we may wish to let $F_{1,j} = f_{1,1}(j)$, $F_{2,j} = f_{1,2}(j)$, $F_{3,j} = f_{2,1}(j)$, and $F_{4,j} = f_{2,2}(j)$.
    1.1 $f_{1,1}(1) = 1$ and $f_{m,k}(1) = 0$ for $m, k > 1$.
2. Recursion: for $i = 2, \ldots, n$; $m = 1, \ldots, s$ and $k = 1, \ldots, s$ set

$$f_{m,k}(j) = e_k(x_j) T_{m,k} \sum_{i=1}^{s} f_{i,m}(j-1)$$

    2.1 Remember to scale.

Baum-Welch (Expectation-Maximation) Algorithm

This algorithm uses the forward-backward algorithm for transitions to estimate T and $e_k(x_j)$.

1. Initialization: Choose or guess initial values for your transition probabilities.

2. E-step: Apply a forward-backward algorithm for transitions to estimate $a_{m,k}(j)$ for $j = 2, \ldots, n$; $m = 1, \ldots, s$ and $k = 1, \ldots, s$. Note that the sequence state probabilities, denoted $p_k(j)$, can be obtained from

   $$\hat{p}_k(j) = P(y_i = k \mid \mathbf{x}) = \sum_{m=1}^{s} P(y_i = k, y_{i-1} = m \mid \mathbf{x}) = \sum_{m=1}^{s} \hat{a}_{m,k}(j)$$

   where $\hat{a}_{m,k}(j)$ and $\hat{p}_k(j)$ represent estimates of $a_{m,k}(j)$ and $p_k(j)$ respectively. These estimates are based on your guesses for T and $e_k(x_j)$.

Baum-Welch (Expectation-Maximation) Algorithm

3. M-step: Estimate the emission and transition probabilities.
   3.1 Loaded die emissions: Letting $I(x_i = t) = 1$ if emission $x_i = t$ and 0 otherwise, then

   $$\hat{e}(t) = \frac{\sum_{i=1}^{n} \hat{p}_k(i) I(x_i = t)}{\sum_{i=1}^{n} \hat{p}_k(i)}$$

   for $t = 1, \ldots, 6$.

   3.2 Transitions:

   $$\hat{T}_{m,k} = \frac{\sum_{i=2}^{n} \hat{a}_{m,k}(i)}{n - 1}$$

4. Repeat steps 2-3 until convergence.

Exercise 1

Download DNAsequences.fa from the GitHub page. This file contiains two long DNA sequences, one from the *C. elegans pha4* gene and the other is from the *C. Briggsae* version of the gene (both species are nematode worms). Note that both sequences are very long (∼5000 bases). Use Needleman-Wunsch algorithm to align sequences int he file. Print the DP matrix to a file, and display your aligned result in a text file containing three lines in the following format:

```
-  C  A  G  A  T  G  A  G  C  A
   |  |     |  |  |  |     |  |
T  C  A  -  A  T  G  A  A  C  A
```

where -' indicates a gap,|' indicates that the sequences are the same for the base and leave a blank space on the second line for the bases that are aligned but are not the same. Evaluate the dynamic programming matrix file and the sequence alignment.

Exercise 2

Using the same DNA sequences as in the previous problem, align the sequences using the Smith-Waterman local alignment algorithm. Based on comparisons of DP matrix and the sequence alignment from the previous problem: what differences do you observe between the two alignments?

Exercise 3

Download `dierolls.txt` from the GitHub page. This file contains two lines of data from the 'Occasionally Dishonest Casino' example from class. The first line contains the die rolls. The second line is the (hidden) Markov state process (which die was used Fair/Loaded). Using only die rolls and the transition and emission probabilities form class, apply the Viterbi algorithm to estimate the most likely sequence of the hidden Markov Chain. Print the DP matrix to a file. Also print both your estimated Markov state sequence as well as the actual sequence to a file. Compare and comment on how the algorithm did in estimating the Markov Chain.

Exercise 4:

Using the `dierolls.txt` data from the GitHub page, apply a forward-backward algorithm to estimate the die state probabilities. Plot this result as well as your Viterbi result and compare the results. Comment on what you observe.

Exercise 5:

Download the die roll data set from `dierolls2.txt` from the GitHub pate. The data set comes from an ‘Occasionally Dishonest Casino’ example where the emission probabilities for the loaded die are not known. Assume you know the fair die probabilities and the transition probabilities are the same as given in the example above. Use an iterative Baum-Welch (EM) algorithm to estimate both the loaded die emission probabilities as well as the state probabilities (Hint: Start with a guess for the loaded die emissions, apply a forward-backward, estimate emissions, apply forward-backward, etc., until convergence). Write your forward, backward, probability and emission results to a file and evaluate the result.

Exercise 5:

Derive the recursion for the forward-backward algorithm for estimating

$$P(y_i = m, y_{i-1} = k \mid \mathbf{x}).$$

For extra credit, derive the complete forward-backward algorithm given in the slides above!

Exercise 6:

Download the die roll data set from `dierolls3.txt` from the GitHub page. The data set comes from an ‘Occasionally Dishonest Casino’ example where the emission probabilities for the loaded die and the transitions are not known. Assume you know the fair die probabilities. Use an iterative Baum-Welch (EM) algorithm to estimate the loaded die emission probabilities, the transition probabilities and the state probabilities. Write your forward, backward, probability, transition, and emission results to a file and evaluate the results.