1. Introduction

A Hidden Markov Model, or HMM, is used to identify and annotate features in a given sequence. An HMM is comprised of states which emit some observed symbol. A path through the model is made by transitioning from state to state emitting a sequence of symbols such as a nucleic acid sequence. To illustrate, consider the following simplified model for an E.coli gene, based on the model from the previous lecture:

Figure 1. A toy HMM for an E.coli gene.
A transition from state U to state $B_1$, $a_{U,B_1}$, has a probability of $\frac{1}{2}$. The probability of observing the emission of an A in state $B_1$, $e_{B_1}(A)$, is $\frac{7}{8}$. In general the transitions and emissions are written as $a_{k,l}$ and $e_k(b)$, where $k$ and $l$ are states and $b$ is some residue. The model is, therefore, defined as:

The model $\theta$: 

- $\{a_{kl}\}$ The probability of transitioning from state $k$ to state $l$, for all states $k, l$. 
- $\{e_k(x)\}$ The probability of emitting base $b$ in state $k$ for all states $k$ and bases $b$.

Suppose there is a sequence: $X = x_1 \ldots x_L$ and a sequence of transitions (a path) $\pi = \pi_1 \ldots \pi_L$. The probability of that sequence and its path could be written as the product of the path and emission probabilities. It is formally defined as:

$$P(x, \pi|\theta) = \prod_{i=1}^{L} a_{\pi_{i-1,\pi_i}} e_{\pi_i}(x_i)$$

or

$$P(x, \pi|\theta) = a_{\text{begin}, \pi_1} e_{\pi_1}(x_1) a_{\pi_1, \pi_2} e_{\pi_2}(x_2) \ldots a_{\pi_{L-1,\pi_L}} e_{\pi_L}(x_L) a_{\pi_L, \text{end}}$$

Key Equations

$$(1)$$

While one could generate random data from an HMM, it is more useful to use the HMM to predict the features of a given sequence. The HMM for, say E.coli protein coding genes given Figure 1, can be used to answer the following questions: 

1. What is the probability of a given sequence $X$ under the given model, i.e. what is $P(X|\theta)$? This asks how typical $x$ is as a segment of E.coli DNA that contains a protein coding gene.
2. What is the most likely state sequence by which the HMM will generate $X$, i.e. what path, $\pi$, maximizes $P(X, \pi|\theta)$. This asks where the gene begins and ends in $x$ and where are all the coding boundaries.
3. What are the parameters of the model that best represent E.coli genes, i.e. what parameters of the model, $\theta$, maximize $P(X|\theta)$? We could use the maximum likelihood principle to take a training set of E.coli genes and use it to set parameters $a_{kl}$ and $e_k(b)$ in Figure 1 to values that are more realistic and appropriate than the ones we have given. These problems are often referred to as evaluating, decoding, and learning. The Forward, Viterbi, and Baum Welch algorithms are used to solve each of these problems respectively. In what follows, we will drop the conditioning on $\theta$ to simplify the notation.
2. The Forward Algorithm

The Forward Algorithm finds the probability of a sequence $X$ with respect to the HMM defined by parameters $\theta$. This probability is $P(X|\theta)$. Since many paths can produce a given sequence, the sum of the probability of all paths is needed. A brute force approach is impractical, so a dynamic programming method is used. Let $f_k(i)$ be the probability of emitting $x_1\ldots x_i$, starting in the begin state and ending in state $k$. Thus,

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

Hence

$$f_i(i+1) = P(x_1\ldots x_{i+1}, \pi_{i+1} = l)$$

$$= \sum_{\pi_i} P(x_1\ldots x_i, \pi_1\ldots\pi_i, \pi_{i+1} = l) e_l(x_{i+1})$$

$$= \sum_k \sum_{\pi_i} P(x_1\ldots x_i, \pi_1\ldots\pi_{i-1}, \pi_i = k) a_{k,l} e_l(x_{i+1})$$

$$= e_l(x_{i+1}) \sum_k f_k(i) a_{k,l}$$

The recursion has the following

Initialization $i=1$: $f_k(1) = a_{\text{begin},k} e_k(x_1)$

Termination: $P(X) = \sum_k f_k(L) a_{k,end}$

This algorithm is applied to the DNA sequence GCTATGCATGTG-TAAG using the E.coli gene in Figure 1. The arrows in Figure 2 show where the summation values for $f_k$ are coming from. The values for each cell tend to be very small and the summations make it difficult to implement the algorithm in log space, but you need to do so.
3. The Viterbi Algorithm

The Viterbi Algorithm is used to calculate the most probable path: $\pi^* = \arg\max_{\pi} P(X, \pi | \theta)$. Let $V_k(i)$ be the probability of the most probable path that starts in the begin state and ends in the state $k$ and emits $x_1...x_i$. Thus

$$V_k(i) = \max_{\pi_1...\pi_{i-1}} P(x_1...x_i, \pi_1...\pi_{i-1}, \pi_i = k).$$

Hence

$$V_k(i+1) = \max_{\pi_1...\pi_i} P(x_1...x_i, \pi_1...\pi_i, x_{i+1}, \pi_{i+1} = l)$$

$$= \max_{\pi_1...\pi_i} P(x_1...x_i, \pi_1...\pi_i, \pi_{i+1} = l) e_l(x_{i+1})$$

$$= \max_{\pi_1...\pi_i} \max_k P(x_1...x_i, \pi_1...\pi_i, \pi_i = k) a_{ki} e_l(x_{i+1})$$

$$= e_l(x_{i+1}) \max_k V_k(i) a_{ki}.$$
This algorithm is very similar to the forward algorithm. The only difference is that a max is used during the recursion instead of a summation. The first few steps of the Viterbi Algorithm are computed in Figure 3. The arrows indicate where the max value came from for each cell. When this algorithm is implemented, back-pointers are used to keep track of the most probable path through the states. Also, since the values tend to get very small, the algorithm should be implemented in log space to avoid numerical precision problems.

![Figure 3. An example of the first few steps in the Viterbi Algorithm](image)

### 4. The Backward Algorithm

The Baum Welch Algorithm, also known as the Forward-Backward Algorithm, is used to calculate the posterior probability of states and emissions given an input sequence $X$. We need to define the Backward Algorithm. The Backward Algorithm calculates and sums the probability of the partial sequence $x_{i+1} \ldots x_L$ conditioned on the $r^{th}$ state.
\[ b_k(i) = P(x_{i+1} \ldots x_N | \pi_i = k) = \sum_{\pi_{i+2} \ldots \pi_N} P(x_{i+1} \ldots x_N, \pi_{i+1} \ldots \pi_N | \pi_i = k) \]

\[ = \sum_l \sum_{\pi_{i+2} \ldots \pi_N} P(x_{i+1} \ldots x_N, \pi_{i+1} = l, \pi_{i+2} \ldots \pi_N | \pi_i = k) \]

\[ = \sum_l \epsilon_l(x_{i+1}) a_{kl} \sum_{\pi_{i+2} \ldots \pi_N} P(x_{i+2} \ldots x_N, \pi_{i+2} \ldots \pi_N | \pi_{i+1} = l) \]

\[ = \sum_l a_{k,l} \epsilon_l(x_{i+1}) b_l(i+1) \]

The recursion has the following:

Initialization \( i=L: \)

\[ b_k(L) = a_{k,end} \]

Termination:

\[ P(X) = \sum_l a_{\text{begin},l} \epsilon_l(x_1) b_l(1) \]

| \( b_k(L) \) | G | C | T | A | A | T | G | C | A | T | G | T | G | T | A | A | A | G |
| \( U \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( B_1 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( B_2 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( B_3 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( C_1 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( C_2 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( C_3 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( S_1 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( S_2 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( S_3 \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( D \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) | \( \ldots \) |

Figure 4. An example of the first few steps of the Backward Algorithm.
5. Posterior Probability

We can now calculate the posterior probability of being in a particular state \( \pi_i = k \) at time \( i \), given that the entire input is \( x = x_1 \ldots x_L \), where \( 1 \leq i \leq L \).

\[
P(\pi_i = k | x) = \frac{P(\pi_i = k, x)}{\sum_{\pi: \pi_i = k} P(x, \pi)}
\]

\[
= \left[ \sum_{\pi_1 \ldots \pi_{i-1}} P(x_1 \ldots x_i, \pi_1 \ldots \pi_{i-1}, \pi_i = k) \right] \times \left[ \sum_{\pi_{i+1} \ldots \pi_N} P(x_{i+1} \ldots x_N, \pi_{i+1} \ldots \pi_N | \pi_i = k) \right]
\]

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

Note that \( P(x) \) for the last term, is a constant that does not depend on \( k \). Since \( P(\pi_i = k | x) \) must sum to 1 over all possible \( k \), it follows that:

\[
P(\pi_i = k | x) = \frac{f_k(i) b_k(i)}{\sum_l f_l(i) b_l(i)}
\]

This gives a way of computing \( P(\pi_i = k | x) \) for any \( i \) and \( k \). First we fill in a table for the forward values \( f_k(i) \). Then we fill in another table for the backward values \( b_k(i) \). Finally we use the above formula to combine these into a table for \( P(\pi_i = k | x) \) for all \( i \) and \( k \).

Another way to look at this is to consider that \( P(x) = \sum_{\pi} P(x, \pi) \). Thus:

\[
P(\pi_i = k | x) = \frac{\sum_{\pi: \pi_i = k} P(x, \pi)}{\sum_{\pi} P(x, \pi)}
\]

That is, \( P(\pi_i = k | x) \) is a ratio of the sum of the probabilities of all paths where the \( i \)th state is \( k \), to the sum of the probabilities of all paths. The forward and backward algorithms allow us to compute these sums efficiently.

6. The Baum Welch Algorithm

The Baum Welch Algorithm, is used to train the model by estimating, for a given training set of sequences, \( A_{kl} \), the number of transitions from state
k to state 1, and $E_k(b)$, the number of times base b is emitted from state k. It then uses these values to update the model parameters $a_{kl}$ and $e_k(b)$ and repeats until the parameters stop changing.

Suppose the training data is N sequences $x^1 \ldots x^N$. It can be shown that the expected number of transitions $A_{kl}$ is

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) b^j_k(i + 1) a_{kl} e_i(x^j_{i+1})$$

Where $f_k^j(i)$ and $b_k^j(i)$ are the forward and backward values for the jth input sequence $x^j$. The value of $E_k(b)$ is calculated likewise as:

$$E_k(b) = \sum_j \frac{1}{P(x^j)} \sum_{i | x_i^j = b} f_k^j(i) b_k^j(i)$$

The values of $a_{kl}$ and $e_i(x_i)$ are computed by taking the maximum likelihood estimate using the parameters as counts.

$$a_{kl} = \frac{A_{kl}}{\sum_m A_{km}} \quad \text{and} \quad e_k(b) = \frac{E_k(b)}{\sum_{a \in \{A,C,G,T\}} E_k(a)}$$

This process is iterated several times until the change in parameters is small. It can be shown that this is a special case of the more general Expectation Maximization algorithm discussed in an earlier lecture. There are a few drawbacks to this method, however. First, it is possible to overtrain the model with the maximum likelihood estimation. This can be solved by adding pseudocounts to $A_{kl}$ and $E_k(b)$, as discovered in the previous lecture on ML versus MAP and mean posterior estimation. These pseudocounts can have a formal meaning in the context of MAP estimation. Second, the Baum Welch Algorithm may find a local maximum instead of a global maximum for the parameters because of the initial values; this becomes worse for large models. It should be tried from many different starting points.

7. Summary
The HMM is used to model and annotate features of an observed sequence. The features, or states, of a sequence are hidden and need to be calculated. The Viterbi Algorithm decodes the sequence by finding the most probable path of hidden states. This is done through a dynamic programming
method which calculates the probability of a series of subsequences to derive the whole sequence. Before the sequence can be decoded, however, the model needs to be calibrated such that its probabilities of transitions and emissions reflect real biology. The Baum Welch Algorithm, with the help of the Forward Backward Algorithm is used to optimize the parameters of the model \( a_{kl} \) and \( c_k(b) \). The training sequences used to estimate these parameters should be typical for the features being modeled. After the parameters of the model are estimated from the training data, the model can then be used to annotate new sequences. At this point the model could also be connected to other HMMs to detect structures that are more complex, as discussed in a previous lecture.