Hidden Markov Models
Analyzing Time Series

• Patterns which appear over a space of time
• Deduce the weather from a piece of seaweed - folklore tells us that `soggy' seaweed means wet weather, while `dry' seaweed means sun.
• If it is in an intermediate state (`damp'), then we cannot be sure.
• However, the state of the weather is not restricted to the state of the seaweed, so we may say on the basis of an examination that the weather is probably raining or sunny.
• A second useful clue would be the state of the weather on the preceding day (or, at least, its probable state) - by combining knowledge about what happened yesterday with the observed seaweed state, we might come to a better forecast for today.
Outline

• First we will introduce systems which generate probabilistic patterns in time, such as the weather fluctuating between sunny and rainy.

• We then look at systems where what we wish to predict is not what we observe - the underlying system is hidden.
  – In the above example, the observed sequence would be the seaweed and the hidden system would be the actual weather.

• We then look at some problems that can be solved once the system has been modeled.
Generating Patterns

• Consider a set of traffic lights; the sequence of lights is red - red/amber - green - amber - red. The sequence can be pictured as a state machine, where the different states of the traffic lights follow each other.
Notice that each state is dependent solely on the previous state, so if the lights are green, an amber light will always follow - that is, the system is deterministic. Deterministic systems are relatively easy to understand and analyze, once the transitions are fully known.
Non-deterministic Patterns

• To make the weather example a little more realistic, introduce a third state - cloudy.

• Unlike the traffic light example, we cannot expect these three weather states to follow each other deterministically, but we might still hope to model the system that generates a weather pattern.

• One way to do this is to assume that the state of the model depends only upon the previous states of the model. This is called the Markov assumption and simplifies problems greatly.

• Obviously, this may be a gross simplification and much important information may be lost because of it.
Markov Assumption

• When considering the weather, the Markov assumption presumes that today's weather can always be predicted solely given knowledge of the weather of the past few days - factors such as wind, air pressure etc. are not considered.

• In this example, and many others, such assumptions are obviously unrealistic.

• Nevertheless, since such simplified systems can be subjected to analysis, we often accept the assumption in the knowledge that it may generate information that is not fully accurate.
Weather...
State Changes

• A Markov process is a process which moves from state to state depending (only) on the previous $n$ states.

• The process is called an order $n$ model where $n$ is the number of states affecting the choice of next state.

• The simplest Markov process is a first order process, where the choice of state is made purely on the basis of the previous state.

• Note that this is not the same as a deterministic system, since we expect the choice to be made probabilistically, not deterministically.
Notice that for a first order process with M states, there are $M^2$ transitions between states since it is possible for any one state to follow another.

Associated with each transition is a probability called the state transition probability - this is the probability of moving from one state to another.

These $M^2$ probabilities may be collected together in an obvious way into a state transition matrix.

Note that these probabilities do not vary in time - this is an important (if often unrealistic) assumption.
State Transition Matrix

- The state transition matrix below shows possible transition probabilities for the weather example:

<table>
<thead>
<tr>
<th></th>
<th>Sun</th>
<th>Cloud</th>
<th>Rain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weather</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>yesterday</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun</td>
<td>0.375</td>
<td>0.125</td>
<td>0.375</td>
</tr>
<tr>
<td>Cloud</td>
<td>0.125</td>
<td>0.625</td>
<td></td>
</tr>
<tr>
<td>Rain</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- That is, if it was sunny yesterday, there is a probability of 0.5 that it will be sunny today, and 0.25 that it will be cloudy or rainy. Note that (because the numbers are probabilities) the sum of the entries for each column is 1.
Initial States

To initialize such a system, we need to state what the weather was (or probably was) on the day after creation; we define this in a vector of initial probabilities, called the $\pi$ vector.

$$
\begin{pmatrix}
1.0 & 0.0 & 0.0 \\
\end{pmatrix}
$$

- that is, we know it was sunny on day 1.
Definition of Markov Process

• We have now defined a first order Markov process consisting of
• states: Three states - sunny, cloudy, rainy.
• vector: Defining the probability of the system being in each of the states at time 0.
• state transition matrix: The probability of the weather given the previous day's weather.

• Any system that can be described in this manner is a Markov process.
Summary

• We are trying to recognize patterns in time, and in order to do so we attempt to model the process that could have generated the pattern
• We use discrete time steps, discrete states, and we may make the Markov assumption
• Having made these assumptions, the system producing the patterns can be described as a Markov process consisting of a \( \pi \) vector and a state transition matrix
• An important point about the assumption is that the state transition probabilities do not vary in time - the matrix is fixed throughout the life of the system
Patterns generated by a hidden process
When a Markov process may not be powerful enough

• In some cases the patterns that we wish to find are not described sufficiently by a Markov process

• Returning to the weather example, a hermit may perhaps not have access to direct weather observations, but does have a piece of seaweed

• Folklore tells us that the state of the seaweed is probabilistically related to the state of the weather - the weather and seaweed states are closely linked

• In this case we have two sets of states, the observable states (the state of the seaweed) and the hidden states (the state of the weather)

• We wish to devise an algorithm for the hermit to forecast weather from the seaweed and the Markov assumption without actually ever seeing the weather.
Speech Recognition is not a simple Markov Process

- A more realistic problem is that of recognizing speech
  - the sound that we hear is the product of the vocal chords, size of throat, position of tongue and several other things
  - Each of these factors interact to produce the sound of a word
  - The sounds that a speech recognition system detects are the changing sound generated from the internal physical changes in the person speaking

- Some speech recognition devices work by considering the internal speech production to be a sequence of hidden states, and the resulting sound to be a sequence of observable states generated by the speech process that at best approximates the true (hidden) states
Hidden States

• In both examples it is important to note that the number of states in the hidden process and the number of observable states may be different.
• In a three state weather system (sunny, cloudy, rainy) it may be possible to observe four grades of seaweed dampness (dry, dryish, damp, soggy)
• Pure speech may be described by (say) 80 phonemes, while a physical speech system may generate a number of distinguishable sounds that is either more or less than 80.
• In such cases the observed sequence of states is probabilistically related to the hidden process
• We model such processes using a hidden Markov model where there is an underlying hidden Markov process changing over time, and a set of observable states which are related somehow to the hidden states.
Hidden Markov Model

- The diagram below shows the hidden and observable states in the weather example. It is assumed that the hidden states (the true weather) are modeled by a simple first order Markov process, and so they are all connected to each other.
Probabilities in a HMM

• The connections between the hidden states and the observable states represent the probability of generating a particular observed state given that the Markov process is in a particular hidden state.

• It should thus be clear that all probabilities `entering' an observable state will sum to 1, since in the above case it would be the sum of \( Pr(\text{Obs}|\text{Sun}) \), \( Pr(\text{Obs}|\text{Cloud}) \) and \( Pr(\text{Obs}|\text{Rain}) \).

• In addition to the probabilities defining the Markov process, we therefore have another matrix, termed the confusion matrix, which contains the probabilities of the observable states given a particular hidden state.
Confusion Matrix

- For the weather example the confusion matrix might be:

<table>
<thead>
<tr>
<th></th>
<th>Dry</th>
<th>Dryish</th>
<th>Damp</th>
<th>Soggy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>0.60</td>
<td>0.20</td>
<td>0.15</td>
<td>0.05</td>
</tr>
<tr>
<td>Cloud</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Rain</td>
<td>0.05</td>
<td>0.10</td>
<td>0.35</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Notice that the sum of each matrix row is 1.
HMM - Summary

• We have seen that there are some processes where an observed sequence is probabilistically related to an underlying Markov process.

• In such cases, the number of observable states may be different to the number of hidden states.

• We model such cases using a hidden Markov model (HMM).
HMM - Summary 2

• This is a model containing two sets of states and three sets of probabilities
  – hidden states: the (TRUE) states of a system that may be described by a Markov process (e.g., the weather).
  – observable states: the states of the process that are `visible' (e.g., seaweed dampness).
  – $\pi$ vector: contains the probability of the (hidden) model being in a particular hidden state at time $t = 1$.
  – state transition matrix: holding the probability of a hidden state given the previous hidden state.
  – confusion matrix: containing the probability of observing a particular observable state given that the hidden model is in a particular hidden state.

• Thus a hidden Markov model is a standard Markov process augmented by a set of observable states, and some probabilistic relations between them and the hidden states.
Definition of HMM

• A hidden Markov model (HMM) is a triple $\left( \pi, A, B \right)$

\[
\begin{align*}
\Pi & = (\pi_i) \quad \text{the vector of the initial state probabilities;} \\
A & = (a_{ij}) \quad \text{the state transition matrix;} \\
B & = (b_{ij}) \quad \text{the confusion matrix;}
\end{align*}
\]

Each probability in the state transition matrix and in the confusion matrix
is time independent - that is, the matrices do not change in time as the
system evolves. In practice, this is one of the most unrealistic
assumptions of Markov models about real processes.
Uses associated with HMMs

• Once a system can be described as a HMM, three problems can be solved.

• The first two are pattern recognition problems
  – Finding the probability of an observed sequence given a HMM (evaluation); and
  – finding the sequence of hidden states that most probably generated an observed sequence (decoding)

• The third problem is generating a HMM given a sequence of observations (learning)
Evaluation

• Consider the problem where we have a number of HMMs (that is, a set of \((\pi, A, B)\) triples) describing different systems, and a sequence of observations.

• We may want to know which HMM most probably generated the given sequence.
  – For example, we may have a `Summer' model and a `Winter' model for the seaweed, since behavior is likely to be different from season to season - we may then hope to determine the season on the basis of a sequence of dampness observations.

• We will use the forward algorithm to calculate the probability of an observation sequence given a particular HMM, and hence choose the most probable HMM.

• This type of problem occurs in speech recognition where a large number of Markov models will be used, each one modeling a particular word. An observation sequence is formed from a spoken word, and this word is recognized by identifying the most probable HMM for the observations.
Decoding

• Find the most probable sequence of hidden states that generated some given observed output.

• In many cases we are interested in the hidden states of the model since they represent something of value that is not directly observable.
Decoding Example

- Consider the example of the seaweed and the weather; a blind hermit can only sense the seaweed state, but needs to know the weather, i.e. the hidden states.
- We will use the Viterbi algorithm to determine the most probable sequence of hidden states given a sequence of observations and a HMM.

- Another widespread application of the Viterbi algorithm is in Natural Language Processing, to tag words with their syntactic class (noun, verb etc.)
  - The words in a sentence are the observable states and the syntactic classes are the hidden states (note that many words, such as wind, fish, may have more than one syntactical interpretation).
  - By finding the most probable hidden states for a sentence of words, we have found the most probable syntactic class for a word, given the surrounding context.
  - Thereafter we may use the primitive grammar so extracted for a number of purposes, such as recapturing `meaning`. 
Learning

• The third, and much the hardest, problem associated with HMMs is to take a sequence of observations (from a known set), known to represent a set of hidden states, and fit the most probable HMM
  – that is, determine the \((\pi,A,B)\) triple that most probably describes what is seen

• We will use the forward-backward algorithm which works when the matrices A and B are not directly (empirically) measurable, as is very often the case in real applications
Summary

• HMMs, described by a vector and two matrices \((\pi, A, B)\) are of great value in describing real systems since, although usually only an approximation, they are amenable to analysis.

• Commonly solved problems are:
  • **Evaluation**: Matching the most likely system to a sequence of observations
    – solved using the forward algorithm;
  • **Decoding**: determining the hidden sequence most likely to have generated a sequence of observations
    – solved using the Viterbi algorithm;
  • **Learning**: determining the model parameters most likely to have generated a sequence of observations
    – solved using the forward-backward algorithm.
Forward Algorithm
Exhaustive Search for Solution

• We want to find the probability of an observed sequence given an HMM - that is, the parameters \((\pi,A,B)\) are known.

• Consider the weather example: we have a HMM describing the weather and its relation to the state of the seaweed, and we also have a sequence of seaweed observations.

• Suppose the observations for 3 consecutive days are (dry, damp, soggy) - on each of these days, the weather may have been sunny, cloudy or rainy.

• We can picture the observations and the possible hidden states as a trellis.
Trellis

Observations:  dry   damp   soggy
Trellis Probabilities

- Each column in the trellis shows the possible state of the weather and each state in one column is connected to each state in the adjacent columns.
- Each of these state transitions has a probability provided by the state transition matrix.
- Under each column is the observation at that time; the probability of this observation given any one of the above states is provided by the confusion matrix.

- It can be seen that one method of calculating the probability of the observed sequence would be to find each possible sequence of the hidden states, and sum these probabilities.
- For the above example, there would be $3^3=27$ possible different weather sequences, and so the probability is
  \[ \text{Pr (dry,damp,soggy | HMM)} = \text{Pr (dry,damp,soggy | sunny,sunny,sunny)} + \text{Pr (dry,damp,soggy | sunny,sunny ,cloudy)} + \text{Pr (dry,damp,soggy | sunny,sunny ,rainy)} + \ldots \ldots \text{Pr (dry,damp,soggy | rainy,rainy ,rainy)} \]
  
- Calculating the probability in this manner is computationally expensive, particularly with large models or long sequences, and we find that we can use the time invariance of the probabilities to reduce the complexity of the problem.
Reduction of complexity using recursion

- We consider calculating the probability of observing a sequence recursively given a HMM.
- We will first define a partial probability, which is the probability of reaching an intermediate state in the trellis.
- We then show how these partial probabilities are calculated at times $t=1$ and $t=n (> 1)$. 
Partial Probabilities

• Suppose throughout that the T-long observed sequence is \( (Y_{k_1}, Y_{k_2}, \ldots, Y_{k_T}) \)

Partial probabilities, \((\alpha \ 's)\)
Consider the trellis below showing the states and first-order transitions for the observation sequence dry, damp, soggy

We can calculate the probability of reaching an intermediate state in the trellis as the sum of all possible paths to that state
Partial Probabilities

- The partial probability of state $j$ at time $t$ is $\alpha_t(j)$.
- This partial probability is calculated as
  $$
  \alpha_t(j) = \Pr(\text{observation} | \text{hidden state is } j) \times \Pr(\text{all paths to state } j \text{ at time } t)
  $$
- The partial probabilities for the final observation hold the probability of reaching those states going through all possible paths.

![Diagram showing the states and observations](image-url)
Partial Probabilities

- It follows that the sum of these final partial probabilities is the sum of all possible paths through the trellis, and hence is the probability of observing the sequence given the HMM.
Calculating $\alpha$’s at time $t = 1$

- We calculate partial probabilities as
  
  $$\alpha_t(j) = Pr(\text{observation } | \text{hidden state is } j) \times Pr(\text{all paths to state } j \text{ at time } t)$$

- In the special case where $t = 1$, there are no paths to the state.

- The probability of being in a state at $t = 1$ is therefore the initial probability
  - $Pr(\text{state } | \text{ } t = 1) = \pi(\text{state}),$
  - we therefore calculate partial probabilities at $t = 1$ as this probability multiplied by the associated observation probability

  $$\alpha_1(j) = \pi(j) \delta_{jk_1}$$

  - Thus the probability of being in state $j$ at initialization is dependent on that state's probability together with the probability of observing what we see at that time
Calculating $\alpha$'s at time $t (> 1)$

- We recall that a partial probability is calculated as
  \[ \alpha_t (j) = P(\text{observation} \mid \text{hidden state is } j) \times P(\text{all paths to state } j \text{ at time } t) \]

- We can assume (recursively) that the first term of the product is available, and now consider the term $P(\text{all paths to state } j \text{ at time } t)$
Sum up all the paths

- To calculate the probability of getting to a state through all paths, we can calculate the probability of each path to that state and sum them.
Summing up…

- The number of paths needed to calculate $\alpha$ increases exponentially as the length of the observation sequence increases but the $\alpha$'s at time $t - 1$ give the probability of reaching that state through all previous paths, and we can therefore define $\alpha$'s at time $t$ in terms of those at time $t - 1$

$$\alpha_{t+1}(j) = b_{jk_{t+1}} \sum_{i=1}^{n} \alpha_t(i) a_{ij}$$

- Thus we calculate the probabilities as the product of the appropriate observation probability (that is, that state $j$ provoked what is actually seen at time $t + 1$) with the sum of probabilities of reaching that state at that time - this latter comes from the transition probabilities together with $a$ from the preceding stage.
Finally

• Notice that we have an expression to calculate $\alpha$ at time $t+1$ using only the partial probabilities at time $t$

• We can now calculate the probability of an observation sequence given a HMM recursively – we use $\alpha$'s at $t = 1$ to calculate $\alpha$'s at $t = 2$; $\alpha$'s at $t = 2$ to calculate $\alpha$'s at $t = 3$; and so on until $t = T$. The probability of the sequence given the HMM is then the sum of the partial probabilities at time $t = T$
Reduction of computational complexity

• We can compare the computational complexity of calculating the probability of an observation sequence by exhaustive evaluation and by the recursive forward algorithm.

• We have a sequence of $T$ observations, $O$

• We also have a Hidden Markov Model
  – $L = (\pi, A, B)$, with $n$ hidden states.
• An exhaustive evaluation would involve computing for all possible execution sequences

\[ X_i = (X_{i_1}, X_{i_2}, \ldots, X_{i_T}) \]

the quantity

\[ \sum_X \pi(i_1) b_{i_1k_1} \prod_{j=2}^{T} \alpha_{i_j-1i_j} b_{i_jk_j} \]

which sums the probability of observing what we do
• The load here is exponential in T but using the forward algorithm we exploit knowledge of the previous time step to compute information about a new one
• Thus the load is only be linear in T
Summary

- Our aim is to find the probability of a sequence of observations given a HMM \((\Pr (\text{observations} \mid \lambda))\).
- We reduce the complexity of calculating this probability by first calculating partial probabilities (\(\alpha\)'s). These represent the probability of getting to a particular state \(s\), at time \(t\).
- We see that at time \(t = 1\), the partial probabilities are calculated using the initial probabilities (from the \(\pi\) vector) and \(\Pr (\text{observation} \mid \text{state})\) (from the confusion matrix).
- The partial probabilities at time \(t > 1\) are then calculated using the partial probabilities at time \(t - 1\).
- This definition of the problem is recursive, and the probability of the observation sequence is found by calculating the partial probabilities at time \(t = 1, 2, ..., T\), and adding all \(\alpha\)'s at \(t = T\).
- Note that computing the probability in this way is far less expensive than calculating the probabilities for all sequences and adding them.
Forward algorithm definition

• We use the forward algorithm to calculate the probability of a $T$ long observation sequence $Y = y_{k_1}, \ldots, y_{k_T}$ where each of the $y$ is one of the observable set.

Intermediate probabilities ($\alpha$'s) are calculated recursively by first calculating $\alpha$ for all states at $t = 1$

$$\alpha_1 (j) = \pi(j) . b_{jk_1}$$
• Then for each time step, \( t = 2, \ldots, T \), the partial probability \( \alpha \) is calculated for each state

\[
\alpha_{t+1}(j) = \sum_{i=1}^{n} \left( \alpha_{t}(i) a_{ij} \right) b_{jk},
\]

• that is, the product of the appropriate observation probability and the sum over all possible routes to that state, exploiting recursion by knowing these values already for the previous time step.

• Finally the sum of all partial probabilities gives the probability of the observation, given the HMM, \( \lambda \).

\[
Pr(Y^{(k)}) = \sum_{j=1}^{n} \alpha_{T}(j)
\]

• To recap, each partial probability (at time \( t > 2 \)) is calculated from all the previous states.
• Using the `weather' example, the diagram below shows the calculation for $\alpha$ at $t = 2$ for the cloudy state.

• This is the product of the appropriate observation probability $b$ and the sum of the previous partial probabilities multiplied by the transition probabilities $\alpha$. 

\[
\begin{align*}
\text{SUNNY} & \quad \alpha_1(s) = \pi_s \cdot b_{sw} \\
\text{CLOUDY} & \quad \alpha_1(r) = \pi_c \cdot b_{cw} \\
\text{RAINY} & \quad \alpha_1(c) = \pi_r \cdot b_{cw} \\
\text{CLOUDY} & \quad \alpha_2(c) = \sum_i (\alpha_1(i) \cdot a_{ic}) \cdot b_{cd} \\
\end{align*}
\]
Example of the Forward Algorithm

To use the example follow these steps:
• Enter a number of valid observed states
• Press 'Set' to initialize the matrix
• Use either 'Run' or 'Step' to make the calculations
  – 'Run' will calculate the \( \alpha \) 's for every node and return the probability of the HMM
  – 'Step' will calculate the \( \alpha \) value for the next node only. Its value is displayed in the output window.
• When finished with the current settings press 'Set' to reinitialize with the current settings, or enter a new set of observed states, followed by 'Set'
Applet

- States may be entered in either or a combination of the following (valid separators are comma and space):
  - Dry, Damp, Soggy
  - Dry Damp Soggy
  - If any invalid state or separator is used then the states remain unchanged from their previous settings

## Model

<table>
<thead>
<tr>
<th>Hidden States (weather)</th>
<th>Observed States (seaweed)</th>
<th>Initial State Probabilities (π Vector)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>Dry</td>
<td>Sunny: 0.63</td>
</tr>
<tr>
<td>Cloudy</td>
<td>Dryish</td>
<td>Cloudy: 0.17</td>
</tr>
<tr>
<td>Rainy</td>
<td>Damp</td>
<td>Rainy: 0.20</td>
</tr>
<tr>
<td></td>
<td>Soggy</td>
<td></td>
</tr>
</tbody>
</table>
# State Transition Matrix (A)

<table>
<thead>
<tr>
<th>weather yesterday</th>
<th>Sunny</th>
<th>Cloudy</th>
<th>Rainy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>0.500</td>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>Cloudy</td>
<td>0.375</td>
<td>0.125</td>
<td>0.375</td>
</tr>
<tr>
<td>Rainy</td>
<td>0.125</td>
<td>0.675</td>
<td>0.375</td>
</tr>
</tbody>
</table>
## Confusion Matrix (B)

<table>
<thead>
<tr>
<th>hidden states</th>
<th>observed states</th>
<th>Dry</th>
<th>Dryish</th>
<th>Damp</th>
<th>Soggy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td></td>
<td>0.60</td>
<td>0.20</td>
<td>0.15</td>
<td>0.05</td>
</tr>
<tr>
<td>Cloudy</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Rainy</td>
<td></td>
<td>0.05</td>
<td>0.10</td>
<td>0.35</td>
<td>0.50</td>
</tr>
</tbody>
</table>
Summary

• We use the forward algorithm to find the probability of an observed sequence given a HMM.

• It exploits recursion in the calculations to avoid the necessity for exhaustive calculation of all paths through the execution trellis.

• Given this algorithm, it is straightforward to determine which of a number of HMMs best describes a given observation sequence:
  – the forward algorithm is evaluated for each, and that giving the highest probability selected.
## Family Members

<table>
<thead>
<tr>
<th>Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCGTL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGHSV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCGSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGGTL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCGSS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Probabilities

| Prob(C) | 0.8 | 0.6 |   |   |   |
| Prob(G) | 0.2 | 0.4 | 0.8 |   |   |
| Prob(H) |   |   | 0.2 |   |   |
| Prob(S) |   |   |   | 0.6 | 0.2 |
| Prob(T) |   |   |   |   | 0.4 |
| Prob(L) |   |   |   |   |   | 0.6 |
| Prob(V) |   |   |   |   |   | 0.2 |

The probability of CGGSV is $0.8 \times 0.4 \times 0.8 \times 0.6 \times 0.2 = 0.031$
or better

$log(0.8) + log(0.4) + log(0.8) + log(0.6) + log(0.2) = -3.48$
Possible HMM for Protein ACCY

Delete States
(silent or model deletions)

Insert States
(emit amino acids)

Match States
(emit conserved primary structure of the protein)
Squares are match states, each of which is paired with a null delete state. The match-delete pair is called a fat state. Each fat state is visited exactly once on every path from Start to End.

Diamonds are insert states are used to represent possibly extra amino acids that are not found in most of the sequences in the family being modeled.
Possible HMM for Protein ACCY

The probability of ACCY along this path is
\[0.4 \times 0.3 \times 0.46 \times 0.6 \times 0.97 \times 0.5 \times 0.015 \times 0.73 \times 0.01 \times 1 = 1.76 \times 10^{-6}\] or

\[\log(0.4) + \log(0.3) + \log(0.46) + \log(0.6) + \log(0.97) + \log(0.5) + \log(0.015) + \log(0.73) + \log(0.01) + \log(1) = -13.25\]
HMM with multiple paths for ACCY
Viberti Algorithm

- Most likely path
  1. The probability that the amino acid A was generated by state I0 is computed and entered as the first element of the matrix
  2. The probabilities that C is emitted in state M1 (multiplied by the probability of the most likely transition to state M1 from state I0) and in state I1 (multiplied by the most likely transition to state I1 from state I0) are entered into the matrix element indexed by C and I1/M1
  3. The maximum probability, max(I1, M1), is calculated
  4. A pointer is set from the winner back to state I0
  5. Steps 2-4 are repeated until the matrix is filled
Probabilities for ACCY

- Prob(A in state I0) = 0.4*0.3 = 0.12
- Prob(C in state I1) = 0.05*0.06*0.5 = 0.015
- Prob(C in state M1) = 0.46*0.01 = 0.005
- Prob(C in state M2) = 0.46*0.5 = 0.23
- Prob(Y in state I3) = 0.015*0.73*0.01 = 0.0001
- Prob(Y in state M3) = 0.97*0.23 = 0.22
Matrix for the Viberti Algorithm

<table>
<thead>
<tr>
<th></th>
<th>I0</th>
<th>I1</th>
<th>M1</th>
<th>I2</th>
<th>M2</th>
<th>I3</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>.015</td>
<td></td>
<td>.005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.0001</td>
</tr>
</tbody>
</table>

Prob(A in state I0) = 0.4*0.3=0.12
Prob(C in state I1) = 0.05*0.06*0.5 = .015
Prob(C in state M1) = 0.46*0.01 = 0.005
Prob(C in state M2) = 0.46*0.5 = 0.23
Prob(Y in state I3) = 0.015*0.73*0.01 = .0001
Prob(Y in state M3) = 0.97*0.23 = 0.22

And probability of most likely path is .120 * .015 * .23 * .22
(or as before score is sums of logs)
Review: Forward Algorithm

- In step 3 sum is computed and no back pointers are necessary (sum probabilities in last column)
- Prob(A in state I0) = 0.4*0.3=0.12
- Prob(C in state I1) = 0.05*0.06*0.5 = 0.015
- Prob(C in state M1) = 0.46*0.01= 0.005
- Prob(C in state M2) = (0.005*0.97) 
  +(0.015*0.46)= .012
- Prob(Y in state I3) = .012*0.015*0.73*0.01 = 1.31x10-7
- Prob(Y in state M3) = .012*0.97*0.2 = 0.002
### Matrix for the forward algorithm

<table>
<thead>
<tr>
<th></th>
<th>I0</th>
<th>I1</th>
<th>M1</th>
<th>I2</th>
<th>M2</th>
<th>I3</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>.015</td>
<td>.005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.012</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.000001</td>
</tr>
</tbody>
</table>

- \(\text{Prob}(A \text{ in state } I0) = 0.4 \times 0.3 = 0.12\)
- \(\text{Prob}(C \text{ in state } I1) = 0.05 \times 0.06 \times 0.5 = 0.015\)
- \(\text{Prob}(C \text{ in state } M1) = 0.46 \times 0.01 = 0.005\)
- \(\text{Prob}(C \text{ in state } M2) = (0.005 \times 0.97) + (0.015 \times 0.46) = 0.012\)
- \(\text{Prob}(Y \text{ in state } I3) = 0.012 \times 0.015 \times 0.73 \times 0.01 = 1.31 \times 10^{-7}\)
- \(\text{Prob}(Y \text{ in state } M3) = 0.012 \times 0.97 \times 0.2 = 0.002\)
Local vs Global Scoring

\[
\text{Score}(A) = \log_8(4) + \log_8(3) = -2.12 \\
\text{Score}(C) = \log_8(46) + \log_8(6) = -1.29 \\
\text{Score}(C) = \log_8(97) + \log_8(5) = -0.72 \\
\text{Score}(Y) = \log_8(0.015) + \log_8(0.73) + \log_8(0.01) = -9.12 \\
\]

\[
(-2.12) + (-1.29) + (-0.72) + (-9.12) = -13.25
\]

The log score for ACCY is significantly lowered by A and Y.

The highest scoring sequence is CC (with a score of -2.01)
Building an HMM

• If the state paths for all the training sequences are known, the emission and transition probabilities in the model can be calculated by computing their *expected value*: observing the number of times each transmission or emission occurs in the training set and dividing by the sum of all the transmission probabilities or all the emission probabilities.

• If the state paths are unknown, finding the best model given the training set is an optimization problem which has no closed form solution. It must be solved by iterative methods.
Building an HMM 2 iteratively

- The goal is to find model parameters which maximize the probability of all sequences in the training set.
- The desired model is a model against which all the sequences in the training set will have the best possible scores.
- The parameters are re-estimated after every iteration by computing a score for each training sequence against the previous set of model parameters.
Baum-Welch Algorithm

- The *Baum-Welch* algorithm is a variation of the forward algorithm described earlier.
- It begins with a reasonable guess for an initial model and then calculates a score for each sequence in the training set over all possible paths through this model.
- During the next iteration, a new set of expected emission and transition probabilities is calculated, as described above for the case when state paths are known.
- The updated parameters replace those in the initial model, and the training sequences are scored against the new model.
- The process is repeated until *model convergence*, meaning there is very little change in parameters between iterations.
Viberti

- The *Viterbi* algorithm is less computationally expensive than Baum-Welch.
- As described earlier, it computes the sequence scores over the most likely path rather than over the sum of all paths.
Viberti Algorithm Advantages

The Viterbi algorithm two key advantages

1. It provides the best interpretation given the entire context of the observations as it proceeds through the trellis calculating a partial probability for each cell, together with a back-pointer indicating how that cell could most probably be reached.

2. It also looks at the whole sequence before deciding on the most likely final state, and then `backtracking' through the state pointers to indicate how it might have arisen. This is very useful in `reading through' isolated noise garbles, which are very common in live data.
Guarantee of Success

- There is no guarantee that a model built with either algorithm has parameters which maximize the probability of the training set.
- As in many iterative methods, convergence indicates only that a local maximum has been found.
- Several heuristic methods have been developed to deal with this problem.
- One approach is to start with several initial models and proceed to build several models in parallel.
- When the models converge at several different local optimums, the probability of each model given the training set is computed, and the model with the highest probability wins.
- Another approach is to add noise, or random data, into the mix at each iteration of the model building process.
- Typically, an annealing schedule is used.
  - The schedule controls the amount of noise added during each iteration.
  - Less and less noise is added as iterations proceed.
  - The decrease is either linear or exponential.
  - The effect is to delay the convergence of the model.
  - When the model finally does converge, it is more likely to have found a good approximation to the global maximum.
Sequence Weighting

• Another problem with HMMs is that if there is a small group of sequences in the training set which are highly similar, the model will overspecialize to the small group.

• To prevent this, several methods of sequence weighting have been developed.

• These methods give the outlier sequences, those that do not belong to the highly similar group, additional importance in the calculation of model parameters
Sequence Weighting 2

• The simplest weighting methods are based on tree structures.

• Sequences in a family are placed on branches of a tree, representing their divergence from a common ancestor.

• One interesting approach is to visualize a tree made of conducting wire with a voltage $V$ applied to the root.

• The leaves are set to $V=0$, and the currents flowing through each branch are calculated using Kirchoff’s laws.

• The currents are then used as the sequence weights. Intuitively, the currents will be smaller in a more highly divided area of the tree (the highly similar sub-group), and larger in less divided areas of the tree (the outliers).
According to Kirchoff's laws, the following equations can be derived:

\[ I_0 = I_1 + I_2 \]
\[ I_1 = I_2 \]
\[ I_2 = I_3 + I_4 \]
\[ I_3 = I_4 \]
\[ I_3 = I_5 + I_6 \]
\[ I_5 = I_6 \]
\[ I_4 = I_7 + I_8 \]
\[ I_7 = I_8 \]

With some algebraic manipulation, we can show that:

\[ I_1 = I_2 = 0.5 \times I_0 \]
\[ I_3 = I_4 = 0.25 \times I_1 \]
\[ I_5 = I_6 = I_7 = I_8 = 0.125 \times I_1 \]
Maximum Discrimination Weighting

- In this method, weights are estimated iteratively while the model is being built.
- After each iteration in the model building process, the sequences in the training set are scored against the current model.
- Weights are assigned to each sequence, with poorly scoring sequences (outliers) receiving the highest valued weights.
- During the next iteration, these sequences get more importance than sequences which had good scores during the prior round.
- The process repeats until the model converges.
Statistical Spread Modeling

- A third kind of weighting method is based on trying to make the statistical spread in the model as uniform as possible.
- The position-specific weighting method developed by Henikoff & Henikoff falls into this category.
- Note that it is not possible to identify a single, best weighting method.
- Choice of a weighting method depends both on what the resulting model will be used for and the particular protein group being modeled.
Henikoff Algorithm

• In this method, weights are based on the diversity observed in the columns of a multiple alignment of sequences.

• A weight is computed for each position in a sequence inversely proportional to $m$, the number of different amino acids in the column and $k$, the number of times the amino acid of interest appears in the column.

\[ \frac{1}{mk} \]

• For example, looking at the first sequence in the multiple alignment shown below the weight of amino acid C in position 1 is

\[ \frac{1}{1 \times 4} = \frac{1}{4} \]

• and the weight of amino acid N in position 7 is

\[ \frac{1}{3 \times 1} = \frac{1}{3} \]

• The weight of a sequence is the average of the weights in all positions, normalized to sum to 1

CGGSLLNAN - - TVLTAHHC
CGGSLI DNK - G WILTAHHC
CGGSLI RQG - - WVMTAHHC
CGGSLI REDSSF V LTAHHC
Overfitting

- Clearly, accurate methods of estimating amino acid distributions are necessary to build good HMMs.
- A potential pitfall is illustrated by the example shown in the last slide.
- Consider these four sequences to be members of a training set.
- The first column shown contains one distinct amino acid: C.
- Using the methods described so far, the probability of any of the other 19 amino acids appearing in this position is 0.
- However, if we set all these probabilities to 0 in the model, we end up with a model unable to recognize family members which do not begin with a C.
- Remember that members of the training set are only a small percentage of real members of the protein family.
- At least some of the other members are likely to begin with different amino acids.

CGGSLLNAN - - TVLTAHNC
CGGSLI DNK - G WILTAHNC
CGGSLI RQG - - WVMTAHNC
CGGSLI REDSSF V LTAAHNC
Overfitting and Regularization

- A variety of approaches known as *regularization* have been developed to deal with it.
- The simplest is to use *pseudocounts*: this means that, even if a given amino acid does not appear in a column of an aligned training set, it is given a fake count.
- Fake counts are also added for the amino acids which appear in the column.
- When probabilities are calculated, the fake counts are treated exactly like real observed counts.
- The first column of the four proteins has 4 counts of the amino acid C.
- If a pseudocount of 1 is used for each of the 20 amino acids, in effect, the column contains 24 entries, of which 4 are real and 20 are fake.

CGGSLLNAN - - TVLTAH
CGGSLLI DNK - G WILTAH
CGGSLLI RQG - - WVMTAH
CGGSLLI REDSSF V LTAH
Overfitting and Regularization

The probability of A in the first column is

\[
\frac{0+1}{4+20} = \frac{1}{24}
\]
Overfitting and Regularization

The probability of C in the first column is

\[
\frac{4 + 1}{4 + 20} = \frac{4}{24}
\]

CGGSLLNAN --- TVLTAH
CGGSLLI DNK - G WILTAH
CGGSLLI RQG - - WVMTAH
CGGSLLI REDSSF V LTAAH
Similar approaches

• A sophisticated application of this method is known as *Dirichlet mixtures*.

• The mixtures are created by statistical analysis of the distribution of amino acids at particular positions in a large number of proteins.

• The mixtures are built from smaller components known as *Dirichlet densities*. 
Dirichlet Densities

• A Dirichlet density is a probability density over all possible combinations of amino acids appearing in a given position.
• It gives high probability to certain distributions and low probability to others.
• For example, a particular Dirichlet density may give high probability to conserved distributions where a single amino acid predominates over all others.
• Another possibility is a density where high probability is given to amino acids with a common identifying feature, such as the subgroup of hydrophobic amino acids.
Dirichlet Densities 2

- When an HMM is built using a Dirichlet mixture, a wealth of information about protein structure is factored into the parameter estimation process.
- The pseudocounts for each amino acid are calculated from a weighted sum of Dirichlet densities and added to the observed amino acid counts from the training set.
- The parameters of the model are calculated as described above for simple pseudocounts.
Multiple Alignment

- HMM can automatically create a multiple alignment from a group of unaligned sequences.
- Recall that multiple alignment is the process of taking a group of sequences and identifying amino acids which are *homologous*, structurally or functionally similar.
- The homologous amino acids are aligned in columns.
- A multiple alignment can be generated by using the Viterbi algorithm to find the most likely path through the HMM for each sequence.
  - Each match state in the HMM corresponds to a column in the multiple alignment.
  - A delete state is represented by a dash.
  - Amino acids from insert states are either not shown or are displayed in lower case letters.
- An alternative to Viterbi is a method known as *posterior decoding*.
- By considering all positions in the model and all twenty amino acids, one can calculate the probability that each amino acid occurs at that each position in the model.
- This produces a table of probabilities for all possible pairs of amino acids and positions in the model.
- From the table, it is possible to find the highest probability path through the model for any protein, based on all possible places that any amino acid can occur.
Conclusions

• The HMM is a linear model and is unable to capture higher order correlations among amino acids in a protein molecule.

• These correlations include hydrogen bonds between non-adjacent amino acids in a polypeptide chain, hydrogen bonds created between amino acids in multiple chains, and disulfide bridges, chemical bonds between C (cysteine) amino acids which are distant from each other within the molecule.

• In reality, amino acids which are far apart in the linear chain may be physically close to each other when a protein folds. Chemical and electrical interactions between them cannot be predicted with a linear model.
Conclusions 2

• Another flaw of HMMs lies at the very heart of the mathematical theory behind these models.
• As anyone who has studied stochastic systems knows, the probability of a sequence of events is the multiplicative product of the probabilities of individual events only when the events are independent.
• We assumed that the probability of a protein sequence can be found by multiplying the probabilities of the amino acids in the sequence.
• This claim is only valid if the probability of any amino acid in the sequence is independent of the probabilities of its neighbors.
• In biology, this is not the case.
• There are, in fact, strong dependencies between these probabilities.
  – For example, hydrophobic amino acids are highly likely to appear in proximity to each other.
  – Because such molecules fear water, they cluster at the inside of a protein, rather than at the surface where they would be forced to encounter water molecules.
• These biological realities have motivated research into new kinds of statistical models. Hybrids of HMMs and neural nets, dynamic Bayesian nets, factorial HMMs, Boltzmann trees and hidden Markov random fields are among the areas being