1. HMM with well-defined states

   Suppose you are stalking somebody. One of the premier rules of stalking is that the person being stalked will be less suspicious if you arrive at a common destination first. Say you have class with the person you are stalking. You sit far enough away from this person so that you cannot hear what this person is saying. However, you can pick up on hand gestures, body language and overall mood. Now, your objective is to infer from these signals where that person plans to go for lunch on a particular day and to beat this person to the destination.

   So in this model, the observed variables are the mood, hand guestures, and body language of a person on a particular day. The states are a set of restaurants frequented by the person being stalked. The state emission probabilities map each state to a probability of observing a type of signal. For example, we could have a state termed *Arbys*. In this state, the emission probabilities could be S1:0.2, S2: 0.6, S3: 0.2 and S4: 0.0, where S1...S4 are catagorizations of signals. The transition probabilities map the probability of going to a restaurant on a certain day, given the restaurant visited on the previous day. For example, we know the probability of the stalkee frequenting the same restaurant two days in a row is relatively low.

   In order to train the model, we first conduct a training period where we follow the stalkee after class to a lunch destination. From the training data, we can determine the transmission probabilities and emission probabilities because the states (restaurants) are well defined.

2) Shorter answer than for #1: suppose we use an HMM to model rainfall. The amount of rain on a given day depends on the state of the clouds, wind pattern, pressure, etc. Those states depend on the states of the previous day. We can use a latent variable to model the clouds, etc., and the rainfall is then dependent on those states. It's not obvious to me what states we should use, so one option might be to build a model with K states, where the state is undefined, and essentially allow the rainfall patterns to cluster (with a temporal aspect) to those K states ... whatever the states are. Training the model would look a lot like the single-sequence HMM described above (assign default params, use E step to assign days to states, then M step to modify params, then repeat).

The problem is that it's not clear what value K should take. As the number is increased, the likelihood of the model will increase, but at some point the gain in likelihood isn't worth the increase in complexity. I suppose there's plenty of theory about where the cutoff of gain-to-complexity is reached, but I'm not familiar with it. I'd guess you could try a wide range of K values, and pick the one that maximizes $\ln lk - cK$ (or $\ln lk/cK$), for some constant value c (or just increase K until that metric peaks). In this way, models are penalized for having a large number of parameters, to counteract the inproved likelihood seen by those using the large number.

3) Provide some of the details to derive 13.17.
<u>ans</u> (13.12) defines

$$Q(\theta,\theta^{\text{old}}) = \sum_z p(z|x,\theta^{\text{old}}) ln p(x,z|\theta)$$
$$= E_z[ln p(x,z|\theta)|x,\theta^{\text{old}}] \tag{3.1}$$

From (13.10),

$$p(x,z|\theta) = p(z|\pi,A)p(x|z,\phi) \tag{3.2}$$

where

$$p(z|\pi,A) = p(z_1|\pi)\prod_{n=2}^{N} p(z_n|z_{n-1},A), \tag{3.3}$$

$$p(x|z,\phi) = \prod_{m=1}^{N} p(x_m|z_m,\phi) \tag{3.4}$$

(3.3) becomes

$$p(z|\pi,A) = \prod_{k=1}^{K}\pi_k^{z_{1k}}\prod_{n=2}^{N}\prod_{j=1}^{K}\prod_{k=1}^{K} A_{jk}^{z_{n-1,j}z_{nk}} \tag{3.5}$$

while (3.4) is

$$p(x|z,\phi) = \prod_{m=1}^{N}\prod_{k=1}^{K} p(x_m|\phi_k)^{z_{mk}} \tag{3.6}$$

Using (3.5) and (3.6), (3.2) becomes

$$p(x,z|\theta) = \prod_{k=1}^{K}\pi_k^{z_{1k}}\prod_{n=2}^{N}\prod_{j=1}^{K}\prod_{k=1}^{K} A_{jk}^{z_{n-1,j}z_{nk}}\prod_{m=1}^{N}\prod_{k=1}^{K} p(x_m|\phi_k)^{z_{mk}} \tag{3.7}$$

Take logs in (3.7) and get

$$ln p(x,z|\theta) = \sum_{k=1}^{K} z_{1k} ln\pi_k + \sum_{n=2}^{N}\sum_{j,k} z_{n-1,j}z_{nk} ln A_{jk} + \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk} ln p(x_n|\phi_k) \tag{3.8}$$

Substitute (3.8) in (3.1) to get

$$Q(\theta,\theta^{\text{old}}) = \sum_{k=1}^{K} E[z_{1k}|x,\theta^{\text{old}}] ln\pi_k + \sum_{n=2}^{N}\sum_{j,k} E[z_{n-1,j}z_{nk}|x,\theta^{\text{old}}] ln A_{jk} + \sum_{n=1}^{N}\sum_{k=1}^{K} E[z_{nk}|x,\theta^{\text{old}}] ln p(x_n|\phi_k) \tag{3.9}$$

$$= \sum_{k=1}^{K}\gamma(z_{1k}) ln\pi_k + \sum_{n=2}^{N}\sum_{j,k}\zeta(z_{n-1,j}z_{nk}) ln A_{jk} + \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk}) ln p(x_n|\phi_k) \tag{3.10}$$

To get (3.10) from (3.9), I used the definitions in (13.13) and (13.14) which are equivalent to

$$\gamma(z_{nk}) = p(z_{nk}=1|x,\theta^{\text{old}}) = E[z_{nk}|x,\theta^{\text{old}}]$$
$$\zeta(z_{n-1,j}z_{nk}) = p(z_{n-1,j}=z_{nk}=1|x,\theta^{\text{old}}) = E[z_{n-1,j}z_{nk}|x,\theta^{\text{old}}]$$

4. The Viterbi algorithm is used to determine a most likely sequence of latent variables given an observed data sequence. This is done by maximizing the likelihood of the complete data assuming the observed variables are fixed at the values in the observed data. Therefore it amounts to a straightforward application of the max-sum algorithm to the tree representation of the HMM in Fig. 13.15 in the text. It is possible to determine the most likely sequence of latent variables by computing the probabilities of each and every path in the trellis representation of the HMM given in Fig. 13.16. However the number of such paths grows exponentially with the number of observed data points. Viterbi algorithm reduces the number of computations by examining all possible transitions of latent variables between two successive time intervals and retaining only K out of the K^2 possible paths for further consideration. The probability of any such path is the product of the transition probability involved and the emission probability of the data point given the latent variable state after the transition. By considering the log of the probabilities it turns out that this strategy is nothing but the max-sum algorithm as in equation 13.68.

   Equation 13.68 should have been

$$\omega(\mathbf{z}_{n+1}) = \ln p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) + \max_{\mathbf{z}_n} \{\ln p(\mathbf{z}_{n+1}|\mathbf{z}_n) + \omega(z_n)\}$$

   Viterbi algorithm is good for detemining a most likely sequence of latent variables given observed data. On the other hand the alpha-beta algorithm is good for computing the marginals of individual or couplets of latent variables to be subsequently used in the M-step of the EM algorithm for parameter estimation. In other words, Viterbi algorithm is the max-sum algorithm and the alpha-beta algorithm is the sum-product algorithm on the tree graphical representation of a HMM.

(5) Matlab Project: MLE & Viterbi Algorithm

(a)

If we assume that any person is equally likely to be talking, our transition probabilities are all equal. For the observed sequence 1-1-0-0, the log-MLE is -5.7559, which corresponds to the speaker sequence AACC.

(b)

Changing the parameters of the transition matrix **A** and starting state probability vector $\boldsymbol{\pi}$ will *certainly* influence the MLE for the observed sequence. For instance, note that Ann (the lead in our MLE sequence above) will no longer start the conversation; Bridgette is now most likely to start the conversation given the observation "1." Further, Ann never speaks after women, and Bridgette loves to talk, so the second speaker is again most likely to be Bridgette, given the observation "1." Only Doug is courageous enough to speak after Bridgette (rather than Chris as before), so he is most likely the speaker of third comment, given the observation "0."

Indeed, the MLE with the new transition matrix is now -4.2813, corresponding to the sequence BBDC.

(c)

The new transition matrix **A** is as follows:

$z_{n-1,j}$

		A	B	C	D
	A	0.4	0	0	0.7
	B	0.2	0.8	0.5	0
$z_{n,k}$	C	0.2	0	0.5	0.3
	D	0.2	0.2	0	0

The starting state probability vector $\boldsymbol{\pi}$ is <0, 0.5, 0.5, 0> for A, B, C, and D, respectively.

(d)

To discover the MLE of the speaker sequence for the given set of 80 observations, we implemented the Viterbi algorithm in Matlab (see Appendix). The implementation is typical. In a forward pass, it calculates the MLE (Omega) values and stores, in a $K \times N$ matrix, the index to the previous state in the maximal likelihood path ending at each state z_{nk} in the state trellis (note K=4). The algorithm then backtracks through the $K \times N$ indices matrix to extract the most probable speaker sequence.

The log-MLE is calculated as -64.0020, corresponding to the speaker sequence

BBBBBDCCCBBBBBBBBDACCBBBBBBBBDCBBDCBDACC
CCBBBBBBBBBBBBBBBBBBBBBDAACCBBBDAACCBBBB

```
% MLE of an HMM:
%   seq: sequence of observed X
%   S: lead-off probabilities
%   A: transition probability matrix
%   E: emission probability matrix
function [omega_max, path] = viterbi (seq, S, A, E)
    states = 4;
    paths  = zeros (states,length(seq));

    % leftmost leaf "h" factor h(z_1) = ln{p(z_1)p(x_1|z_1)}
    omega = log(S) + log(E((seq(1)+1),:))
    for n=1 : (length(seq)-1)
        for k=1 : states
            [paths(k,n+1), transit(k)] = max_index(log(A(k,:)) + omega);
            k=k+1;
        end
        omega = log(E((seq(n+1)+1),:)) + transit;
        n=n+1;
    end
    [last, omega_max] = max_index (omega);
    path = backtrack (paths, last);
    path = char(path);
end

function [ind, val] = max_index (states)
    ind = 1; val = states(1);
    for i=2 : length(states)
        if states(i) > val
            ind = i;
            val = states(i);
        end
        i=i+1;
    end
end

function path = backtrack (paths, last)
    sz = size(paths);
    path = zeros(1,sz(1,2));
    i = sz(1,2);

    % first (the end) node
    k = last;
    while i>0
        path(i) = node(k);
        k = paths(k,i);
        i=i-1;
    end
end

function val = node (k)
    switch k
        case 1
            val = 'A';
        case 2
            val = 'B';
        case 3
            val = 'C';
        case 4
            val = 'D';
    end

end
```

6. PRML Problem 13.5

   To get 13.18, we need to maximize

$$\sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k$$

   wrt π_k, subject to the contraint

$$\sum_{k=1}^{K} \pi_k = 1$$

   Using Lagrange multipliers, this is equivalent to maximizing

$$\sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k + \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right)$$

   Differentiating wrt π_k, we get

$$\frac{\gamma(z_{1k})}{\pi_k} + \lambda = 0 \Rightarrow$$

$$\pi_k = -\frac{\gamma(z_{1k})}{\lambda}$$

   Plugging in the constraint, we get

$$1 = \sum_{k=1}^{K} \pi_k = -\frac{\sum_{k=1}^{K} \gamma(z_{1k})}{\lambda} \Rightarrow$$

$$\lambda = -\sum_{k=1}^{K} \gamma(z_{1k})$$

   Substituting the term for λ into π_k, we get

$$\pi_k = \frac{\gamma(z_{1k})}{\sum_{j=1}^{K} \gamma(z_{1j})}$$

   which is 13.18.

To get 13.19, we need to maximize

$$\sum_{n=2}^{N}\sum_{j=1}^{K}\sum_{k=1}^{K}\xi(z_{n-1,j},z_{nk})\ln A_{jk}$$

wrt A_{jk}, subject to the contraint

$$\sum_{k=1}^{K}A_{jk}=1$$

Using Lagrange multipliers, this is equivalent to maximizing

$$\sum_{n=2}^{N}\sum_{j=1}^{K}\sum_{k=1}^{K}\xi(z_{n-1,j},z_{nk})\ln A_{jk}+\lambda\left(\sum_{k=1}^{K}A_{jk}-1\right)$$

Differentiating wrt A_{jk}, we get

$$\frac{\sum_{n=2}^{N}\xi(z_{n-1,j},z_{nk})}{A_{jk}}+\lambda=0\Rightarrow$$

$$A_{jk}=-\frac{\sum_{n=2}^{N}\xi(z_{n-1,j},z_{nk})}{\lambda}$$

Plugging in the constraint, we get

$$1=\sum_{k=1}^{K}A_{jk}=-\frac{\sum_{k=1}^{K}\sum_{n=2}^{N}\xi(z_{n-1,j},z_{nk})}{\lambda}\Rightarrow$$

$$\lambda=-\sum_{k=1}^{K}\sum_{n=2}^{N}\xi(z_{n-1,j},z_{nk})$$

Substituting the term for λ into A_{jk}, we get

$$A_{jk}=\frac{\sum_{n=2}^{N}\xi(z_{n-1,j},z_{nk})}{\sum_{l=1}^{K}\sum_{n=2}^{N}\xi(z_{n-1,j},z_{nl})}$$

which is 13.19.