# Fundamentals of Probabilistic Data Mining Lab 2

Zeinab Abdallah, Lucas Batier, Predrag Pilipovic

December 23, 2019

## 2 Hidden Markov Models

In this lab session we will be discussing Hidden Markov Models and their application to automatic speech recognition, and more precisely to phoneme recognition. We will be using the Hidden Markov Model toolkit (HTK: `http://htk.eng.cam.ac.uk/`). We will provide you the necessary tools to use HTK, the documentation and the data. The focus of this lab is to understand the behavior of the model(s) and algorithm(s). You have the following resources:

- The HTK documentation [`www.gipsa-lab.fr/~thomas.hueber/cours/htkbook.pdf`].
- The speech data pre-processed for use [provided later on].
- The compiled HTK routines [provided later on] (or the source code [provided later on]).
- The basis python script that we strongly suggest to use [provided later on].

### 2.1 Preliminary work

Do this before the class. Questions about this part will be answered only at the beginning of the practical session

1. **Describe the forward-backward algorithm of the EM for HMM. Recall the forward and backward recursions. What is this algorithm computing (provide a formula and an explanation)? What is the relationship between the state occupancy and the forward and backward recursions?**

   Hidden Markov Model has two types of states, the first one is the observation, i.e. the data we have, and the second one is called the hidden state, because we do not have a lot of knowledge of those.

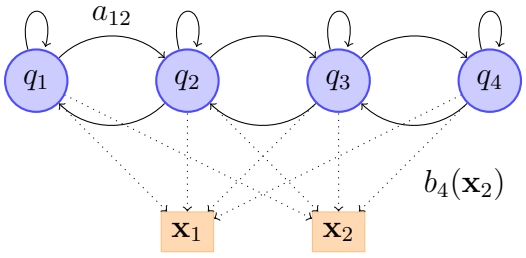



Figure 1: An HMM with 4 states which can emit 2 vectors $\mathbf{x}_1$ or $\mathbf{x}_2$. In this particular HMM, states can only reach themselves or the adjacent state.

The only thing we know of the hidden states is that they are dependent on each other in the Markov way. The probability from one hidden state to an another is called the **transition probability**, and we write $a_{ij}(k)$ to denote probability from states $q_i$ to state $q_j$ between times $(k-1)$ and $k$. The probability from a hidden state to an observation is called the **emission probability** and we write $\alpha_i$ for state $i$. An illustration of Hidden Markov Model is shown Figure 1. We can notice in this figure, that we did not represented time variable in our graph. There are just states, and possibilities going from one state to another, without keeping track of time. If we want to represent time, we would use graphs like the one presented in Figure 2.

The main task here is to find the probability for being in a certain state by knowing a specific observation. Now, let us describe this problem more formally. Let $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_N)$ be a independent random vector we will denote as observations, where $\mathbf{X}_i$ can be continuous or discrete random vector. On the other hand, let $\mathbf{Q} = (Q^{(1)}, Q^{(2)}, ..., Q^{(N)})$ be random vector we will denote as hidden or latent variables, where each $Q^{(j)}$ must be discrete random variable. Each random variable $Q^{(j)}$ represents state at the time $j$ and $Q^{(j)}$ is a discrete random variable with $L$ possible values $\{1, 2, ..., L\}$. Let us assume we are given data $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$. Also, let say that a particular sequence of states is described as $\mathbf{q} = (q_1, q_2, ..., q_N)$ where $q_j \in \{1, 2, ..., L\}$ is the state at time $j$. For the defining all the parameters we will just use states $\{1, 2, ..., L\}$, but later in our calculation we will use $\mathbf{q}$. There are two conditional independence assumptions made about these random variables that make associated algorithms tractable. These independence assumptions are

(a) the $k^{\text{th}}$ hidden variable, given the $(k-1)^{\text{th}}$ hidden variable, is independent of all previous variables, or

$$P(Q^{(k)} \mid Q^{(k-1)}, \mathbf{X}_{k-1}, ...\mathbf{X}_2, \mathbf{X}_1) = P(Q^{(k)} \mid Q^{(k-1)}),$$

(b) the $k^{\text{th}}$ observation, given the $k^{\text{th}}$ hidden variable, is independent of other variables, or

$$P(\mathbf{X}_k \mid Q^{(k)}, ...Q^{(2)}, Q^{(1)}, \mathbf{X}_{k-1}, ...\mathbf{X}_2, \mathbf{X}_1) = P(\mathbf{X}_k \mid Q^{(k)}).$$

Now, let us define parameters of the model. Firstly, we define initial probability as

$$\pi_i = P(Q^{(1)} = i)$$

which is the probability to be in state $i$ at the time $k = 1$, and because $Q^{(1)}$ is discrete random variables it must be

$$\sum_{i=1}^{L} \pi_i = 1.$$

Then, we will define transition probabilities

$$a_{ij} = P(Q^{(k+1)} = j \mid Q^{(k)} = i).$$

We can see that $a_{ij}$ does not depend on time $k$, the only thing that matters is that we switched from state $i$ to state $j$. Because of this property, we say that Hidden Markov model is **time-homogeneous**. At the end, in our case we will define the emission probability as

$$b_j(\mathbf{x}) = p_{\mathbf{X}|Q^{(k)}=j}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mu_j, \Sigma_j),$$

because we assumed that $\mathbf{X} \mid Q^{(k)} = j$ is multivariate Gaussian distribution. In the general case, the emission probability can be anything, like density function of the Gaussian distribution, or density function of GMM or even a deep neural network. Now, we are ready to introduce the **forward - backward algorithms**. In the case of **forward algorithm** we want to compute the probability of knowing a historical partial sequence of observations $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k)$ and being in state $l$ at time $k$, while in the **backward case**, we know a set of future observation $(\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, ..., \mathbf{x}_N)$, and we have to compute the probability for being in state $l$ at a time step $k$. For the forward algoritham we want to compute

$$\alpha_l(k) = p_{\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_k, Q^{(k)}|\Theta}(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, l) = p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, Q^{(k)} = l \mid \Theta),$$

where $\Theta$ represents parameters $\Pi$, or the vector of initial probabilities $\pi_i$, transition probability matrix $\mathbf{A} = [a_{ij}]_{i,j=1}^{L}$, the vector of the emission probabilities $\mathbf{b}$, and the parameters of Gaussian distributions $\mu$ and $\Sigma$. So, our $\Theta$ is given by

$$\Theta = (\Pi, \mathbf{A}, \mathbf{b}, \mu, \Sigma).$$

If we are speaking of the forward algorithm we have next recursive formula

$$\alpha_l(1) = p(\mathbf{x}_1, Q^{(1)} = l) = b_l(\mathbf{x_1})\pi_l$$

$$\alpha_l(k+1) = b_l(\mathbf{x}_{k+1}) \sum_{i=1}^{L} \alpha_i(k) a_{il}.$$

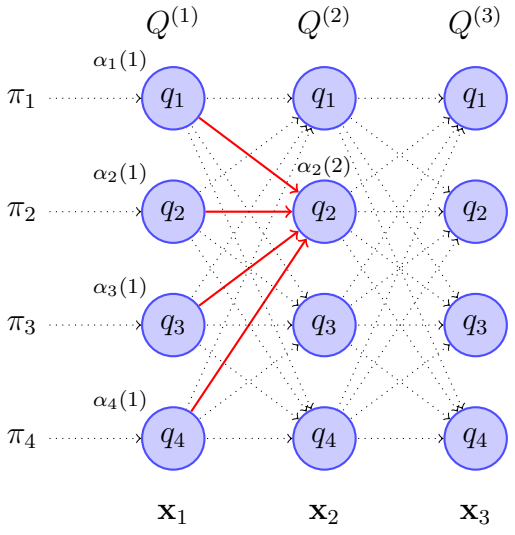



Figure 2: HMM representation of time versus states. The thick arrows indicate all the states used to calculate $\alpha_2(2)$. In that purpose, we used weighted mean of all $\alpha_i(1)$.

This means, we are computing the next value of sequence $\alpha_i$ by computing the weighted mean of all previous values. Let us see an example in Figure 2.

Moving to the backward algorithm. This time, we want to compute the forward variable $\beta_l(k)$, i.e. the joint probability of the partial observation sequence $(\mathbf{x}_{k+1}, \mathbf{x}_{k+2}..., \mathbf{x}_N)$ given that the hidden state at time $k$ is $l$, or

$$\beta_l(k) = p_{\mathbf{X}_{k+1},\mathbf{X}_{k+2},...,\mathbf{X}_N,|Q^{(k)}=l,\Theta}(\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, ..., \mathbf{x}_N) = p(\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, ..., \mathbf{x}_N \mid Q^{(k)} = l, \Theta).$$

We can compute this with the recursive formula

$$\beta_l(N) = 1$$
$$\beta_l(k-1) = b_l(\mathbf{x}_k) \sum_{i=1}^{L} \beta_i(k) a_{li}$$

At the end let us define the **state occupation probability** or the probability that given observation $\mathbf{X}$ is generated by the state $l$ at the time $k$, i.e. $P(Q^{(k)} = l \mid \mathbf{X}, \Theta)$. This can be seen as an equivalent of the GMM responsibilities. Let us try and find the connection between state occupation probability and probabilities in forward-backward algorithm. Firstly, after using the Independence of $\mathbf{X}$ we have

$$\begin{aligned} p(\mathbf{x}, Q^{(k)} = l \mid \Theta) &= p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_N, Q^{(k)} = l \mid \Theta) \\ &= p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, Q^{(k)} = l \mid \Theta) p(\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, ..., \mathbf{x}_N \mid Q^{(k)} = l, \Theta) \\ &= \alpha_l(k)\beta_l(k). \end{aligned}$$

At the end, we can use Bayes rule and total probability law, to have

$$\gamma_l(k) = P(Q^{(k)} = l \mid \mathbf{X} = \mathbf{x}, \Theta) = \frac{p_{\mathbf{X},Q^{(k)}|\Theta}(\mathbf{x}, l)}{p_{\mathbf{X}|\Theta}(\mathbf{x})} = \frac{p_{\mathbf{X},Q^{(k)}|\Theta}(\mathbf{x}, l)}{\sum_{l=1}^{L} p_{\mathbf{X},Q^{(k)}|\Theta}(\mathbf{x}, l)} = \frac{\alpha_l(k)\beta_l(k)}{\sum_{l=1}^{L} \alpha_l(k)\beta_l(k)}.$$

Note here, that we can use whatever $Q^{(k)}$ in the sum in the nominator, because we are doing marginal probability over all possible states for $Q^{(k)}$. Later on, we will choose $Q^{(1)}$. For now, let us move on. We will also need to compute

$$\begin{aligned} \xi_{ij}(k) &= P(Q^{(k)} = i, Q^{(k+1)} = j \mid \mathbf{X} = \mathbf{x}, \Theta) = \frac{p_{\mathbf{X},Q^{(k)},Q^{(k+1)}|\Theta}(\mathbf{x}, i, j)}{p_{\mathbf{X}|\Theta}(\mathbf{x})} \\ &= \frac{p(\mathbf{x}_1, ..., \mathbf{x}_k, Q^{(k)} = i \mid \Theta) a_{ij} b_j(\mathbf{x}_{k+1}) p(\mathbf{x}_{k+2}, ..., \mathbf{x}_N \mid Q^{(k+1)} = j, \Theta)}{p_{\mathbf{X}|\Theta}(\mathbf{x})} \\ &= \frac{\alpha_i(k) a_{ij} b_j(\mathbf{x}_{k+1}) \beta_j(k+1)}{\sum_{i=1}^{L} \sum_{j=1}^{L} \alpha_i(k) a_{ij} b_j(\mathbf{x}_{k+1}) \beta_j(k+1)}. \end{aligned}$$

We can rewrite the nominator in this way, because we have marginal distribution of $\mathbf{X}$, from $(\mathbf{X}, Q^{(k)}, Q^{(k+1)})$.

Remember that we set $\mathbf{q} = \{q_1, q_2, ..., q_N\}$ to be particular sequence of states, where $q_j$ is in $\{1, 2, ..., L\}$, and it is the state at time $j$. Before going to show the EM algorithm, we firstly need some probabilities. To begin with, after using the independence of $\mathbf{X}$ we have

$$p_{\mathbf{X}|\mathbf{Q}=\mathbf{q},\Theta}(\mathbf{x}) = \prod_{k=1}^{N} p_{\mathbf{X}_k|Q^{(k)}=q_k,\Theta}(\mathbf{x}_k) = \prod_{k=1}^{N} b_{q_k}(\mathbf{x}_k).$$

Using the chain rule and Markov property we have

$$\begin{aligned} p_{\mathbf{Q}|\Theta}(\mathbf{q}) &= P(Q^{(1)} = q_1, Q^{(2)} = q_2, ..., Q^{(N)} = q_N \mid \Theta) \\ &= P(Q^{(1)} = q_1 \mid \Theta) \cdot \prod_{k=2}^{N} P(Q^{(k)} = q_k \mid Q^{(k-1)} = q_{k-1}, \Theta) \\ &= \pi_{q_1} \prod_{k=2}^{N} a_{q_{k-1},q_k}. \end{aligned}$$

After applying Bayes rule we obtain

$$p_{\mathbf{X},\mathbf{Q}|\Theta}(\mathbf{x}, \mathbf{q}) = p_{\mathbf{X}|\mathbf{Q}=\mathbf{q},\Theta}(\mathbf{x}) P(\mathbf{Q} = \mathbf{q} \mid \Theta) = \pi_{q_1} b_{q_1}(\mathbf{x}_1) \prod_{k=2}^{N} a_{q_{k-1},q_k} b_{q_k}(\mathbf{x}_k).$$

And finally, we can see from Bayes rule, that

$$p_{\mathbf{Q}|\mathbf{X}=\mathbf{x},\Theta}(\mathbf{q}) = \frac{p_{\mathbf{X},\mathbf{Q}|\Theta}(\mathbf{x}, \mathbf{q})}{p_{\mathbf{X}|\boldsymbol{\Theta}}(\mathbf{x})} = \frac{p_{\mathbf{X},\mathbf{Q}|\Theta}(\mathbf{x}, \mathbf{q})}{\sum_{i=1}^{L} \alpha_i(1)\beta_i(1)}.$$

Now is the time to estimate parameters $\Theta$ by the EM algorithm. Like always, we need to compute function $Q\left(\Theta, \Theta^{(t)}\right)$, and than we need to maximize it over all $\Theta$. Let $\mathcal{Q}$ be space of all state sequences of length $N$. We have

$$\begin{aligned} Q\left(\Theta, \Theta^{(t)}\right) &= \mathbb{E}_{\mathbf{Q}|\mathbf{X}=\mathbf{x},\Theta^{(t)}}[\log p_{\mathbf{X}}(\mathbf{x}, \mathbf{Q} \mid \Theta)] = \sum_{\mathbf{q}\in\mathcal{Q}} p_{\mathbf{Q}|\mathbf{X}=\mathbf{x},\Theta^{(t)}}(\mathbf{q}) \log p_{\mathbf{X},\mathbf{Q}|\Theta}(\mathbf{x}, \mathbf{q}) \\ &= \sum_{\mathbf{q}\in\mathcal{Q}} \frac{p_{\mathbf{X},\mathbf{Q}|\Theta^{(t)}}(\mathbf{x}, \mathbf{q})}{\sum_{i=1}^{N} \alpha_i^{(t)}(1)\beta_i^{(t)}(1)} \log \left( \pi_{q_1} b_{q_1}(\mathbf{x}_1) \prod_{k=2}^{N} a_{q_{k-1},q_k} b_k(\mathbf{x}_k) \right) \\ &= \frac{1}{\sum_{i=1}^{N} \alpha_i^{(t)}(1)\beta_i^{(t)}(1)} \left( \sum_{\mathbf{q}\in\mathcal{Q}} p_{\mathbf{X},\mathbf{Q}|\Theta^{(t)}}(\mathbf{x}, \mathbf{q}) \log \pi_{q_1} + \right. \\ &\quad \left. \sum_{\mathbf{q}\in\mathcal{Q}} \sum_{k=2}^{N} p_{\mathbf{X},\mathbf{Q}|\Theta^{(t)}}(\mathbf{x}, \mathbf{q}) \log a_{q_{k-1},q_k} + \sum_{\mathbf{q}\in\mathcal{Q}} \sum_{k=1}^{N} p_{\mathbf{X},\mathbf{Q}|\Theta^{(t)}}(\mathbf{x}, \mathbf{q}) \log b_{q_k}(\mathbf{x}_k) \right). \end{aligned}$$

Firstly, we need to notice that we want to estimate parameters $(\Pi, \mathbf{A}, \mathbf{b})$. Because those parameters are independently split into the three sums in the previous line, we can optimize each term individually. For the first sum, we can say that

$$\begin{aligned}\sum_{\mathbf{q}\in\mathcal{Q}} p_{\mathbf{X},\mathbf{Q}|\Theta^{(t)}}(\mathbf{x},\mathbf{q})\log\pi_{q_1} &= \sum_{i=1}^{L} p(\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_N, Q^{(1)}=i \mid \Theta^{(t)})\log\pi_i \\ &= \sum_{i=1}^{L}\alpha_i^{(t)}(1)\beta_i^{(t)}(1)\log\pi_i.\end{aligned}$$

Since by selecting all $\mathbf{q}\in\mathcal{Q}$, we are simply repeatedly selecting the values of $Q^{(1)}$, so the right hand side is just the marginal expression for time $k=1$. Also, we put $(t)$ at the $\alpha_i^{(t)}$ and $\beta_i^{(t)}$ as a note that they depend on parameters $\Theta^{(t)}$, and not $\Theta$. If we want to maximize the $Q\left(\Theta,\Theta^{(t)}\right)$ function with respect to $\pi_i$ and having in mind constrain $\sum_{i=1}^{L}\pi_i = 1$, we will have

$$\frac{\partial}{\partial\pi_i}\left(\frac{\sum_{i=1}^{L}\alpha_i^{(t)}(1)\beta_i^{(t)}(1)\log\pi_i}{\sum_{i=1}^{L}\alpha_i^{(t)}(1)\beta_i^{(t)}(1)} + \lambda\left(1-\sum_{i=1}^{L}\pi_i\right)\right) = 0,$$

from where we have

$$\pi_i^{(t+1)} = \frac{\alpha_i^{(t)}(1)\beta_i^{(t)}(1)}{\sum_{i=1}^{L}\alpha_i^{(t)}(1)\beta_i^{(t)}(1)} = \gamma_i^{(t)}(1).$$

The second sum in the function $Q\left(\Theta,\Theta^{(t)}\right)$ becomes

$$\sum_{\mathbf{q}\in\mathcal{Q}}\sum_{k=2}^{N} p_{\mathbf{X},\mathbf{Q}|\Theta^{(t)}}(\mathbf{x},\mathbf{q})\log a_{q_{k-1},q_k} = \sum_{i=1}^{L}\sum_{j=1}^{L}\sum_{k=2}^{N} p(\mathbf{x}, Q^{(k-1)}=i, Q^{(k)}=j \mid \Theta^{(t)})\log a_{ij}$$

because for this term, we are, for each time $k$ looking over all transitions from $i$ to $j$ and weighting that by the corresponding probability. So, the right hand side is just the joint-marginal for times $k-1$ and $k$. In a similar way, we can use a Lagrange multiplier with the constraint $\sum_{j=1}^{L} a_{ij} = 1$ to obtain

$$\begin{aligned}a_{ij}^{(t+1)} &= \frac{\sum_{k=2}^{N} p(\mathbf{x}, Q^{(k-1)}=i, Q^{(k)}=j \mid \Theta^{(t)})}{\sum_{k=2}^{N} p(\mathbf{x}, Q^{(k-1)}=i \mid \Theta^{(t)})} \\ &= \frac{\sum_{k=2}^{N}\xi_{ij}^{(t)}(k-1)}{\sum_{k=2}^{N}\gamma_i^{(t)}(k-1)} = \frac{\sum_{k=1}^{N-1}\xi_{ij}^{(t)}(k)}{\sum_{k=1}^{N-1}\gamma_i^{(t)}(k)}.\end{aligned}$$

Finally, the third sum in the function $Q\left(\Theta, \Theta^{(t)}\right)$ becomes

$$\sum_{\mathbf{q}\in\mathcal{Q}}\sum_{k=1}^{N} p_{\mathbf{X},\mathbf{Q}|\Theta^{(t)}}(\mathbf{x},\mathbf{q})\log b_{q_k}(\mathbf{x}_k) = \sum_{i=1}^{L}\sum_{k=1}^{N} p(\mathbf{x}, Q^{(k)} = i \mid \Theta^{(t)})\log b_i(\mathbf{x}_k),$$

because for this term, we are, for each time $k$, looking at the emissions for all states and weighting each possible emission by the corresponding probability. So, the right hand side is just the sum of the marginal for time $k$. In our case, we presumed that $b_j(\mathbf{x}_k)$ is multivariate Gaussian density function, so from know on it is well known procedure to find an update rules for vector $\mu^{(t+1)}$ and covariance matrix $\Sigma^{(t+1)}$, which we will skiped here.

2. **Describe the Viterbi algorithm. Recall its recursion. What is this algorithm computing (provide a formula and an explanation)?**

   The **Viterbi algorithm** is a derivation of the forward-backward algorithm, it can be seen as an approximation of the forward-backward procedure. This algorithm is used for finding the most likely sequence of hidden states, which is called **Viterbi path**, that results in Hidden Markov Model. We want to compute the joint probability of the observation sequence together with the most likely state sequence when coming to state $l$ at the time $k$, i.e.

$$\delta_l(k) = \max_{q_1,q_2,\ldots,q_{k-1}} p(Q^{(1)} = q_1, ..., Q^{(k-1)} = q_{k-1}, \mathbf{x}_1, ..., \mathbf{x}_k, Q^{(k)} = l \mid \Theta).$$

   So, the general idea for computing $\delta_l(k)$ is this. If the best path ending in $Q^{(k)} = l$ goes through $Q^{(k-1)} = i$ then it should coincide with best path ending in $Q^{(k-1)} = i$. This is bring us to the recursive formula

$$\begin{aligned}
\delta_l(k) &= \max_{q_1,q_2,\ldots,q_{k-1}} p(q_1, q_2, ..., q_{k-1}, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, Q^{(k)} = l \mid \Theta) \\
&= \max_{1\leqslant i\leqslant L}\left\{a_{il}b_l(\mathbf{x}_k)\max_{q_1,q_2,\ldots,q_{k-2}} p(q_1, q_2, ..., q_{k-2}, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{k-1}, Q^{(k-1)} = i \mid \Theta)\right\} \\
&= \max_{1\leqslant i\leqslant L}\left\{a_{il}b_l(\mathbf{x}_k)\delta_i(k-1)\right\}.
\end{aligned}$$

   Of course, the initialization of this recursive formula is

$$\delta_l(1) = p(\mathbf{x}_1, Q^{(1)} = l) = b_l(\mathbf{x}_1)\pi_l.$$

   To retrieve the Viterbi path, we also need to keep track of the argument which was maximized for each time $k$ and state $l$. We will do this with new sequence $\psi_l$. The initialization step of every state will be equal to 0 because no specific argument coming from the initial probability maximized the value of the first state. So,

$$\psi_l(1) = 0.$$

We have next formula

$$\psi_l(k) = \arg\max_{1 \leqslant i \leqslant L} \{a_{il} b_l(\mathbf{x}_k) \delta_i(k-1)\}.$$

Now, the only thing left is to find the sequence of hidden states by backtracking through the sequence $\psi$. For that we are using next formula

$$\hat{q}_{k-1} = \psi_{\hat{q}_k}(k),$$

which means that in order to find $\hat{q}$ at time $k-1$, we are basically looking in the sequence $\psi$ of the state $\hat{q}_k$ at time $k$. Let us now see in Figure 3 how graph of Viterbi path looks like.

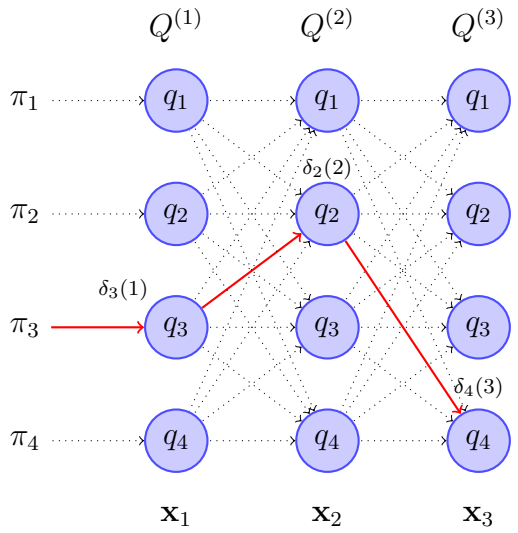



Figure 3: Now, the thick arrows indicate Viterbi path, i.e. the most likely sequence of hidden states, given observations. For computing $\delta_2(2)$, instead of using the weighted mean of all $\delta_i(1)$, this time we used "weighted" maximum of $\delta_i(1)$. In this example, $\psi_4(3) = 2$, $\psi_2(2) = 3$, and $\psi_3(1) = 0$.

So, Viterbi algorithm gives the answer to the question: Given the trained parameter matrices and data, what is the choice of states such that the joint probability reaches maximum? In other words, what is the most likely choice given the data and the trained model?

3. **What is the main difference between the forward and the Viterbi algorithms? Explain this in your own words.**

   Like the forward algorithm, Viterbi is a kind of dynamic programming that makes uses of a dynamic programming, where dynamic programming refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. The Viterbi algorithm is similar to the forward algorithm, but the difference is we replaced summation by maximization. Let us recall the forward and Viterbi

recursions

$$\alpha_l(k) = b_l(\mathbf{x}_k) \sum_{i=1}^{L} \alpha_i(k-1) a_{il},$$
$$\delta_l(k) = b_l(\mathbf{x}_k) \max_{1 \leqslant i \leqslant L} \{\delta_i(k-1) a_{il}\}.$$

Of course, the initial values $\alpha_l(1)$ and $\delta_l(1)$ are the same. The key point is that for forward algorithm we consider the contribution of all possible previous state at time $k-1$ in order to calculate $\alpha_l(k)$. On the contrary, for the Viterbi algorithm we consider the contribution of just one previous state at time $k-1$ with the maximum chance of happening.

4. **In HTK, the EM for HMM (also called Baum-Welch) is implemented in the HRest routine. What are the main input and output variables of this routine?**

   HRest is taking an initial HMM as an input, where the initial HMM is computed by the HInit routine. After that, HRest uses the Baum-Welch algorithm until convergence in order to estimate HMM parameters. The convergence criterion is based on the log likelihoo, i,e. if

   $$\log p\left(\mathbf{x}, \mathbf{q}|\theta^{(t+1)})\right) - \log p\left(\mathbf{x}, \mathbf{q}|\theta^{(t)}\right) < \varepsilon,$$

   for some threshold $\varepsilon > 0$, then the algorithm converges. At the end HRest will return the final HMM, i.e. the model parameters.

5. **As for any EM algorithm, the Baum-Welch algorithm needs to be initialised. In HTK, the initialisation routine is HInit. What are the main input and output variables of this routine? Describe this routine (both for the Gaussian and the GMM emission probability case). What is the rationale of this routine?**

   HInit takes as input a prototype HMM and returns the HMM with initial parameters we want to estimate. So, in case where the emitted probability is a Gaussian, we need to provide:

   - Number of states [scalar]
   - Gaussian parameters for each state ($\mu_k$ [vector], $\Sigma_k$ [vector], for state $k$)
   - Transition probability between each state [matrix]

   If the emitted probability is a Mixture of Gaussian we need to provide:

   - Number of states [scalar]
   - Number of mixtures ([scalar] implicit in the declaration of each Gaussian)
   - GMM parameters for each state and each mixture ($\mu_k$ [vector], $\Sigma_k$ [vector], mixing probability $\pi_{ki}$ for state $k$ and mixture $i$)
   - Transition probability between each state [matrix]

We are creating this prototype to let the routine know the form of the HMM, which means that the actual numbers used in the definition are not important. Well, it is important should we put zero or not for the transition probability. We will prove this for the forward-backward algorithm (HInit uses Viterbi algorithm, and this property can be proven in that case, too) in the question 5 of the section 2.2.1. Which means that the allowable transitions between states should be indicated by putting non-zero values in the corresponding elements of the transition matrix and zeros elsewhere. The rows of the transition matrix must sum to one except for the final row which should be all zero. Apart from that, all mean values can be zero, but diagonal variances should be positive and covariance matrices should have positive diagonal elements. Also, all state definitions can be identical. After creating the prototype file we can run HInit routine. HInit will iteratively run the Viterbi algorithm in order to compute the initial HMM parameters. The Viterbi algorithm is used to find the most likely state sequence corresponding to each training example. Then, parameters are updated. As a side-effect of finding the Viterbi state alignment, the log likelihood of the training data can be computed. Hence, the whole estimation process can be repeated until no further increase in likelihood is obtained.

## 2.2 HMM with Gaussian emission probabilities

In this section we will be using HMMs with Gaussian emission probabilities.

### 2.2.1 Isolated training

1. **Take a look to the provided script. How many hidden Markov models are we going to train?**

   For isolated training we are going to train one HMM per every phoneme, which means we will have 37 models in total.

2. **Run the isolated training. Report the parameters of the model for vowel "a". How many states are we using to model vowel "a"? And to model vowel "o"?**

   After running isolated training, the results of the EM algorithm can be found in the folder `tp_reco_htk_2019\models\hrest` for all the phonemes separately or in the file `tp_reco_htk_2019\models\hrest_0` for all the phonemes together. For every phoneme we will train HMM with 3 states. Although, we will use 5 states but the first one and the last one will be there just to connect different models, we call them non-emitting states. The reason for using 3 phoneme in continuous speech recognition is because the phoneme acoustic is affected by preceding phoneme and succeeding phoneme. For that reason it is more precise to split each phoneme on 3 parts: the transition from previous phoneme at the beginning, stable middle and transition to the next phoneme at the end. If phoneme would be isolated and stable, then 1 state would be enough. It is also possible to use 5 states for single phoneme in continuous speech, but it does not greatly improve the accuracy.

So, in our case here, we used 3 states for single phoneme and for emission probabilities we used multivariate Gaussian. This means that for phoneme "a" we have next parameters:

- 3 vectors of dimension 26 for the means $\mu_1^{(a)}$, $\mu_2^{(a)}$, and $\mu_3^{(a)}$, for each state;
- 3 diagonal covariance matrices of dimension $26 \times 26$, i.e. $\Sigma_1^{(a)}$, $\Sigma_2^{(a)}$, and $\Sigma_3^{(a)}$, for each state;
- and $5 \times 5$ transition probability matrix $\mathbf{A}^{(a)} = \left[a_{ij}^{(a)}\right]_{i,j}^{5}$, which looks like

$$\mathbf{A}^{(a)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.58 & 0.42 & 0 & 0 \\ 0 & 0 & 0.84 & 0.16 & 0 \\ 0 & 0 & 0 & 0.71 & 0.29 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

From the last matrix we can see that we efficiently using just three states. As far as initial probabilities constrained in HMM for speech recognition, we will always start from state 1, so $\pi_1^{(0)} = 1$, and all the rest initial probabilities will be zeros. Now, we can see HMM graph for this model in Figure 4.

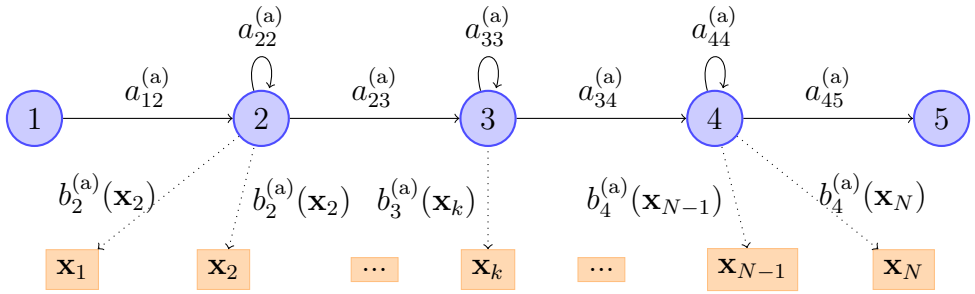


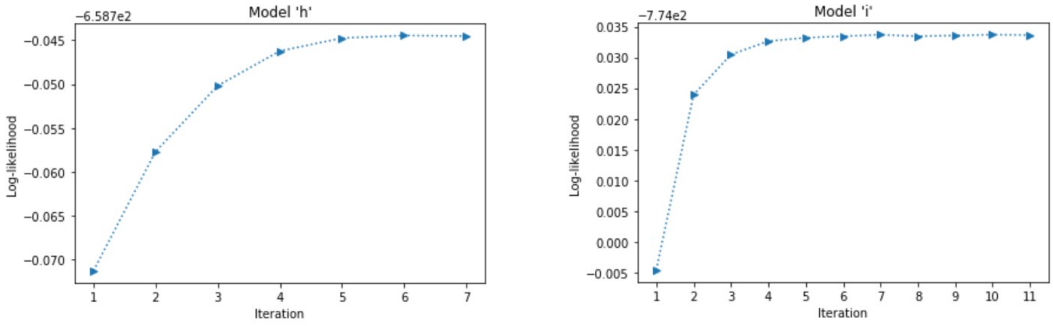

Figure 4: An HMM for phoneme "a" with 5 states, of which 3 are emitting. For the emitting probabilities we have $b_i^{(a)}(\mathbf{x}_k) = \mathcal{N}\left(\mathbf{x}_k; \mu_i^{(a)}, \Sigma_i^{(a)}\right)$, for $i = 1, 2, 3$, and $k = 1, 2, ..., N$.

For the parameters of mean and variance we are not going to provide all of them, but for example, we can see the beginning and ending of parameters $\mu_1^{(a)}$ and $\Sigma_1^{(a)}$.

$$\mu_1^{(a)} = \begin{bmatrix} -1.218066 \\ -10.05744 \\ \vdots \\ -2.144164 \\ 0.8303064 \end{bmatrix}, \qquad \Sigma_1^{(a)} = \begin{bmatrix} 21.94459 & 0 & \dots & 0 & 0 \\ 0 & 32.44511 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 2.634948 & 0 \\ 0 & 0 & \dots & 0 & 0.7090271 \end{bmatrix}.$$

3. **Launch the training. Identify and monitor the progress of the log-likelihood. Provide a plot for two different models.**

   We chose two phonemes with different speed of convergence for this plot. In our case those are "h" and "i". We can see plots of convergences of log-likelihood for those phonemes in Figure 5.




(a) Phonem 'h' (b) Phonem 'i'

Figure 5: Example of log-likelihood evolution over Baum-Welch iterations

4. **Discuss the transition probabilities for "a" and "o". Do you observe a pattern? Is the pattern repeated for other vowels?**

   We have already seen the transition probability matrix for phoneme "a". Let us now put one by one next to each other to try and see the pattern.

$$\mathbf{A}^{(\mathrm{a})} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.58 & 0.42 & 0 & 0 \\ 0 & 0 & 0.84 & 0.16 & 0 \\ 0 & 0 & 0 & 0.71 & 0.29 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{A}^{(\mathrm{o})} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.56 & 0.44 & 0 & 0 \\ 0 & 0 & 0.83 & 0.17 & 0 \\ 0 & 0 & 0 & 0.67 & 0.33 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

   As we can see above those two transition probability matrices are almost the same, with similar probabilities. This result motivated us to search for other vowels with the same pattern. We found that vowels "e", "u", and phoneme "ê" have more or less the same matrices with the same probabilities. But we will not focus now on the fact that all the vowels have the same transition matrix, but the fact that all the phonemes will have the matrix which follows next pattern

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & a_{22} & 1-a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 1-a_{33} & 0 \\ 0 & 0 & 0 & a_{44} & 1-a_{44} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

5. **Take a look to the model specifications in `models\proto`. Report the initialisation of all the parameters. Taking into account the initialisation of**

**the weights, and the forward-backward and the re-estimation formulae of the transition probabilities: could you justify the pattern you observed for the transition probabilities? Provide a graphic representation of the state machine associated to this pattern.**

For the initialization part we have $\mu_1^{(0)} = \mu_2^{(0)} = \mu_3^{(0)} = \mathbf{0}$ and $\Sigma_1^{(0)} = \Sigma_2^{(0)} = \Sigma_3^{(0)} = \mathbf{I}$, where $\mathbf{I}$ is matrix with all ones on diagonal and the rest are zeros. The transition probability matrix is

$$\mathbf{A}^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.6 & 0.4 & 0 & 0 \\ 0 & 0 & 0.6 & 0.4 & 0 \\ 0 & 0 & 0 & 0.6 & 0.4 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

We have already seen one way of presenting HMM for one phoneme in Figure 4, but we cannot see time in this plot, so let us look at the Figure **??**. We want to show that transition probability matrix will not change its pattern through iteration. In other words, we want to prove that if $a_{ij}^{(t)} = 0$, then $a_{ij}^{(t+1)} = 0$. But this the consequence of the updating rule for $a_{ij}$ from the EM algorithm using forward-backward we saw in the first question of preparatory work.

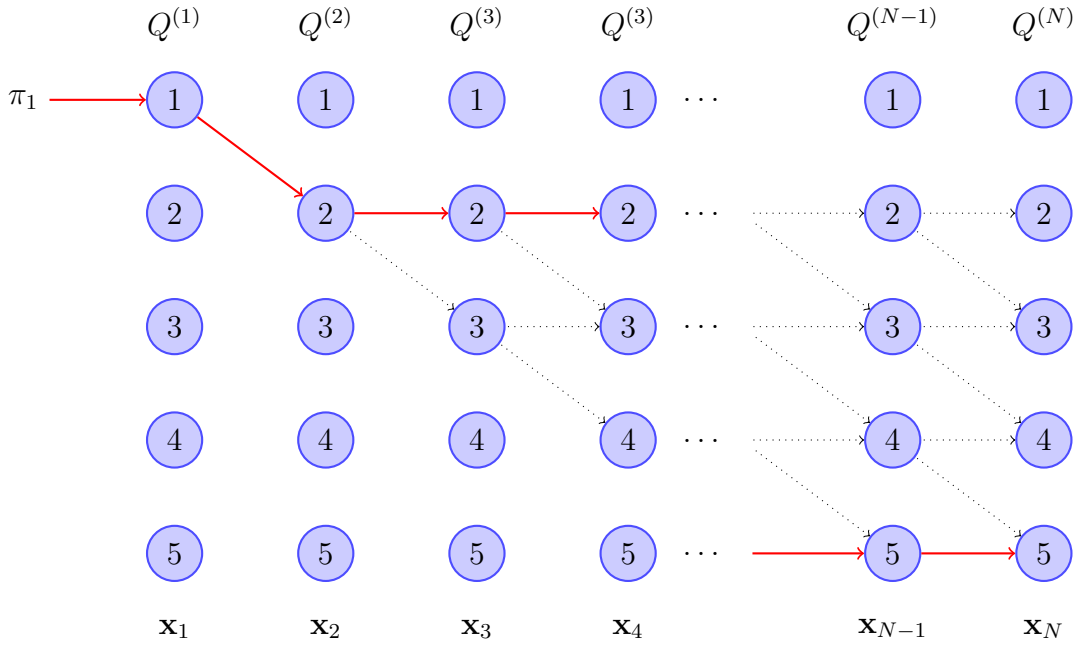



Figure 6: HMM for the initial parameters, represented through time. Dotted line present all the possible ways of going from $Q^{(1)}$ to $Q^{(N)}$, and red line is one example.

As we already know, the update rule for $a_{ij}$ is

$$a_{ij}^{(t+1)} = \frac{\sum_{k=1}^{N-1} \xi_{ij}^{(t)}(k)}{\sum_{k=1}^{N-1} \gamma_i^{(t)}(k)} = \frac{\sum_{k=1}^{N-1} \alpha_i^{(t)}(k) a_{ij}^{(t)} \beta_j^{(t)}(k+1)}{\sum_{k=1}^{N-1} \alpha_i^{(t)}(k)\beta_i^{(t)}(k)} = a_{ij}^{(t)} \frac{\sum_{k=1}^{N-1} \alpha_i^{(t)}(k)\beta_j^{(t)}(k+1)}{\sum_{k=1}^{N-1} \alpha_i^{(t)}(k)\beta_i^{(t)}(k)}.$$

Now, it is easy to see that if $a_{ij}^{(t)}$ is zero, $a_{ij}^{(t+1)}$ must be also zero.

6. **Report the accuracy performance in the test set.**

   Once the optimal alignment has been found, the number of substitution errors ($S$), deletion errors ($D$) and insertion errors ($I$) can be calculated. The correctness is then

$$\text{Correctness} = \frac{N - S - D}{N},$$
$$\text{Accuracy} = \frac{N - S - D - I}{N},$$

   where $N$ is the number of all phonemes. So, we got the next results

   `%Corr = 79.42, %Acc = 77.43 [H = 359, D = 38, S = 55, I = 9, N = 452].`

   The previous line gives the phoneme level statistics and indicates that of the 452 phonemes in total, 359 (79.42%) were recognised correctly. There was 38 deletion errors, 55 substitution errors and 9 insertion errors. The accuracy figure (Acc) of 77.43% is lower than the percentage correct (Cor) because it takes account of the insertion errors which the latter ignores.

### 2.2.2 Connected training (embedded parameter re-estimation)

1. **Launch the connected training using HERest routine. What is the difference between the connected and isolated training? Hint: let us consider the word "couleur". Let us suppose that a few observations at the beginning of the sound "ou" are wrongly labeled as the end of sound "c". What would happen in isolated training? And in connected training?**

   In the isolated training, the model is trained on individual sounds extracted from a real speech. This can produce good whole word models on isolated discrete speech. On the other hand, sub-word models produced using this approach for continuous speech will be sub-optimal, due to the difficulty in determining where one sound finishes and another begins. Also, the hand labeling of the data that is required is a time consuming task. To avoid these problems, connected training is usually used when training on continuous speech. A single composite HMM is formed from all the models corresponding to the training statements. Using an initial set of model parameters, the model states are automatically aligned to frames in the training data sequences. When all the training sentences have been processed, the parameters are re-estimated

against the new alignment. The new parameters replace the initial parameters in the model, and the process is repeated until the parameters converge. In our example with the word "couleur", for isolated training the wrongly labeled data will influence the final model. On the other hand, in connected training we will use the whole word "couleur", so there will be no error.

2. **Report the parameters of the same models you reported in question 3). Do you observe any changes?**

   After running connected training, the results of the EM algorithm can be found in the file `tp_reco_htk_2019\models\herest_10`. Like before, for every phoneme we will train HMM with 5 states, although, we will use only 3 states emitting states. We will have the same parameter set and the results are

$$\mathbf{A}^{(\mathrm{a})} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.59 & 0.41 & 0 & 0 \\ 0 & 0 & 0.84 & 0.16 & 0 \\ 0 & 0 & 0 & 0.65 & 0.35 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

   Again, for the parameters of mean and variance we are not going to provide all of them, but we will show just the beginning and ending of parameters $\mu_1^{(\mathrm{a})}$ and $\Sigma_1^{(\mathrm{a})}$.

$$\mu_1^{(\mathrm{a})} = \begin{bmatrix} -1.382108 \\ -9.452225 \\ \vdots \\ -2.390631 \\ 0.9118489 \end{bmatrix}, \qquad \Sigma_1^{(\mathrm{a})} = \begin{bmatrix} 20.07755 & 0 & \dots & 0 & 0 \\ 0 & 36.25691 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 2.811613 & 0 \\ 0 & 0 & \dots & 0 & 0.7912306 \end{bmatrix}.$$

   As we can see on the example of the phoneme "a", there is no much difference between the isolated and connected training. The thing which is mostly different is the transition probabilities $a_{44}$ and $a_{45}$. We took a look at the rest of the transition matrices and we noticed that sometimes there is a big difference in those probabilities. For example, let us look at phoneme "f"

$$\mathbf{A}_{\mathrm{iso}}^{(\mathrm{f})} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.73 & 0.27 & 0 & 0 \\ 0 & 0 & 0.9 & 0.1 & 0 \\ 0 & 0 & 0 & 0.72 & 0.28 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{A}_{\mathrm{con}}^{(\mathrm{f})} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.71 & 0.29 & 0 & 0 \\ 0 & 0 & 0.89 & 0.11 & 0 \\ 0 & 0 & 0 & 0.62 & 0.38 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

   We can see that the transition probability of the last step has a difference of 0.1 between connected and isolated training, this happens because of the link we do after consonants and before vowels.

3. **Report the accuracy performance in the test set. Discuss any differences with respect to the isolated training.**

   Let us remember the results from the isolated training

   %Corr = 79.42, %Acc = 77.43 [H = 359, D = 38, S = 55, I = 9, N = 452],

   and the results from the connected training

   %Corr = 79.87, %Acc = 76.11 [H = 361, D = 31, S = 60, I = 17, N = 452]

   We can see that there is no noticeable difference between the two results, this is because we do not have a lot of training data.

## 2.3 Mandatory additional questions

### 2.3.1 HMM with GMM emission probabilities

In this section we will be using HMMs with GMM emission probabilities.

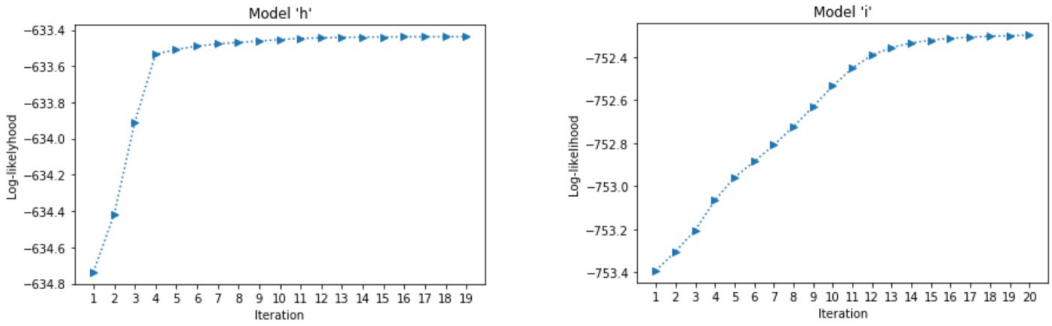
1. **What changes do you need to apply to `models\proto` so as to use a GMM as emission probability?**

   We need to add mixture weights and mean vectors and covariacne matrices for each states. If we want to have 2 GMM components, the second state in the `proto` file must look like this

```
<BeginHMM>
 <NumStates> 5
  <State> 2 <NumMixes> 2
   <Mixture> 1 0.4
    <Mean> 26
     0.0 0.0 ...  0.0
    <Variance> 26
     1.0 1.0 ...  1.0
   <Mixture> 2 0.6
    <Mean> 26
     0.0 0.0 ...  0.0
    <Variance> 26
     1.0 1.0 ...  1.0
```

2. **Run the isolated training for a low number of GMM components: is the log-likelihood still growing?**

   The log-likelihood should still growing because we are still using EM algorithm. We will plot the same plots as for the Multivariate Gaussian distribution on Figure 8.




(a) Phonem 'h' (b) Phonem 'i'

Figure 8: Log-likelihood evolution over number of iteration for HMM with 2 GMM

We can see that for HMM with 2 GMM components we have more iterations until convergence and that the speed of convergence changed, but log-likelihood is still growing.

3. **Report the accuracy performance in the test set. Discuss any differences with respect to the previous case (in isolated training).**

   Again, let us remember the results from the isolated training

   `%Corr = 79.42, %Acc = 77.43 [H = 359, D = 38, S = 55, I = 9, N = 452].`

   On the other hand, let us look at the results from the isolated training done on HMM with 2 GMM components

   `%Corr = 85.18, %Acc = 82.96 [H = 385, D = 25, S = 42, I = 10, N = 452]`

   We can see that with 2 components GMM we have better results. We had a fewer number of deletions and substitutions, and one insertion more. Which resulted in 26 more phonemes well classified in the second case.

   Let us now discuss reason for this. We want to determine the likelihood of an audio feature vector $\mathbf{X}$ given the phoneme, or $P(\mathbf{X} \mid \text{phoneme})$. With MFCC, we extract 26 features from an audio frame. Let us for now simplify the picture and assume that there is only one feature for each frame. For the first part of phoneme "a", i.e. state 1, the value of this feature can be modeled with a normal distribution. To extend the concept to 26 features, we just need a Multivariate normal distribution with 26 variables. Given different phonemes, we can calculate the corresponding probability density values and classify it as the phoneme with the highest value. So this Gaussian model is easy to learn from training data and give us a nice likelihood model. In the context of speech recognition, we can learn the Gaussian model $\mathcal{N}(\mu_l, \Sigma_l)$ for each phoneme (or in our case we represent phoneme with 3 states, so for each state of each phoneme). This serves as the likelihood probability. This also acts as the emission probability in an HMM. Unfortunately, this concept is naive when we use a multivariate Gaussian Distribution. The likelihood is more complex than a single peak bell curve.

The assumption of a Gaussian distribution at each state is very strong; in practice the acoustic feature vectors associated with a state may be strongly non-Gaussian. To address this, we switch to a Gaussian Mixture Model or GMM. This allows the distributions to be multimodal, i.e. we allow a feature to have a few possible values. This provides the flexibility of variants in speech. Given enough components, GMM can model any distribution.

4. **Run the connected training with the same number of GMM components and report the accuracy. Discuss any differences with the previous cases.**

   The results for the 2 GMM connected training are next

   `%Corr = 85.40, %Acc = 83.41 [H = 386, D = 22, S = 44, I = 9, N = 452].`

   Again, like the last time, we cannot see the improvement in the connected training in comparison to isolated one, because we do not have enough data.

5. **Produce an accuracy curve as a function of the number of GMM components. Is the accuracy systematically growing with the number of components? Provide an explanation of the phenomenon you observe.**

### 2.3.2 Full covariance matrices

1. **What kind of covariance matrix have we used until now?**

   We used diagonal matrix, or matrix with zeros everywhere except on diagonal. This is because MFCC parameters have a nice property. There are relatively independent of each other. Therefore, the non-diagonal element of $\Sigma$ can simply set to zero. But now, we will try to use full covariance matrix instead.

2. **Does HTK allow the use of other types of covariance matrices? What happens if you train the models with full covariance matrices? Hint: follow the same path as in the previous sections. Start with Gaussian emission probabilities and isolated training, then connected training. Then switch to GMM emission probabilities and isolated training, the connected training. Finally increase the number of GMM components. Report the accuracy in all your experiments.**

## 2.4 Optional additional questions

- **We have seen that there are different choices in the model (# of GMM components, full/diagonal covariance matrix). Propose a protocol to automatically choose among these different models.**
- **Provide at least three examples of extensions of hidden Markov models, together with the reference to the published article.**

As we saw, in HMM the set of observations are independent and hence the study will occure as above, the extensions of hidden Markov model is characterized by emitting a set of DEPENDENT sequence of observations, i.e, exactly as in the hidden state, where each state is depending in the neighborhooding state, the observations depend in their neighborhoods as well, this case is called **Autoregressive HMM** which is explicitly explained in [cite the reference]. Another extension for HMM, which also known in biology fields

- **Provide the motivation (from the modeling perspective) of each of these variants.**
- **Sketch the differences in the E and/or M steps of the associated EM algorithm for each of these variants.**