HMM for Gesture Recognition

Hongji Guo
August 19, 2020

Abstract
This Document is a summary of HMM for gesture recognition process.

1 Data Preprocessing

1.1 What data we have
In this work, we use the 3D skeleton data. They are the coordinates of joints in the camera frame. Assume we have $N$ sequence data:

$$\{x_c(n), y(n)\}_{n=1}^N$$

(1)

where $x_c(n)$ has the dimension $D \times T$

$x_c$ means the coordinates are in the camera frame.

1.2 Data normalization

1.2.1 Body-centered frame
Define the body-centered as we have 25 body joints. The direction of the $x,y,z$ axis of the body-centered frame is defined as below:

$X$: from right hip to the left hip
$Y$: from base to neck
$Z$: from center of right hip and left hip to the front

1.2.2 How to do the normalization
To make the data invariant to translation and rotation and different people. We need to transform the coordinates from the camera to the body-centered frame.

$$X = RX_c + T$$

(2)

where $X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ is the coordinates of joints in the body-centered frame,

and $X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$ is the coordinates of joints in the camera frame.
Our major task is to reconstruct the rotation matrix $R$ and the transformation matrix $T$ using the data (coordinates of joints in the camera frame).

Assume the center of the body spine base in the camera is $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$, the coordinate of left hip in the camera frame is $\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$, the coordinate of the neck in the camera frame is $\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$. From the translation geometry, we have relationships below:

The relationship of coordinates of the spine base in the body-centered frame and camera frame can be captured as:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = R \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + T \quad (3)$$

The relationship of coordinates of the left hip in the body-centered frame and camera frame can be captured as:

$$\begin{bmatrix} \text{dist}_1 \\ 0 \\ 0 \end{bmatrix} = R \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + T \quad (4)$$

where $\text{dist}_1 = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$, it is the distance from the left hip to the spine base.

And similarly, we have the relationship of coordinates of the neck in the body-centered
frame and camera frame can be captured as:

\[
\begin{bmatrix}
0 \\
\text{dist}_2 \\
0
\end{bmatrix} = R \begin{bmatrix} x_2 \\
y_2 \\
z_2 \end{bmatrix} + T
\]  \hspace{1cm} \text{(5)}

where \(\text{dist}_2 = \sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2 + (z_2 - z_0)^2}\), it is the distance between the neck and the spine base.

Subtract equation (4) by equation (3), we have:

\[
\begin{bmatrix}
\text{dist}_1 \\
0 \\
0
\end{bmatrix} = R \begin{bmatrix} x_1 - x_0 \\
y_1 - y_0 \\
z_1 - z_0 \end{bmatrix}
\]  \hspace{1cm} \text{(6)}

Then multiply both sides by \(R^T\), as \(R\) is an orthogonal matrix \((R^T R = I)\), we have:

\[
R^T \begin{bmatrix}
\text{dist}_1 \\
0 \\
0
\end{bmatrix} = \begin{bmatrix} x_1 - x_0 \\
y_1 - y_0 \\
z_1 - z_0 \end{bmatrix}
\]  \hspace{1cm} \text{(7)}

We can see;

\[
\begin{bmatrix}
r_{11} & r_{21} & r_{31} \\
r_{12} & r_{22} & r_{32} \\
r_{13} & r_{23} & r_{33}
\end{bmatrix} \begin{bmatrix}
\text{dist}_1 \\
0 \\
0
\end{bmatrix} = \begin{bmatrix} x_1 - x_0 \\
y_1 - y_0 \\
z_1 - z_0 \end{bmatrix}
\]  \hspace{1cm} \text{(8)}

We use \(r_1, r_2\) and \(r_3\) to represent the first, second and third row of \(R\)

So we can get the first row of \(R\) in this way:

\[
r_1^T = \begin{bmatrix} r_{11} \\
r_{12} \\
r_{13} \end{bmatrix} = \frac{\begin{bmatrix} x_1 - x_0 \\
y_1 - y_0 \\
z_1 - z_0 \end{bmatrix}}{\text{dist}_1}
\]  \hspace{1cm} \text{(9)}

Similarly, we can reconstruct the second row of \(R\) as below:

\[
r_2^T = \begin{bmatrix} r_{21} \\
r_{22} \\
r_{23} \end{bmatrix} = \frac{\begin{bmatrix} x_2 - x_0 \\
y_2 - y_0 \\
z_2 - z_0 \end{bmatrix}}{\text{dist}_2}
\]  \hspace{1cm} \text{(10)}

Finally, we can compute the third row of \(R\) by the cross product the first row and second
row of R:

\[ r_3^T = r_1^T \times r_2^T \]  \hspace{1cm} (11)

We got \( R = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} \)

After we got \( R \), from equations (3), we can compute the transform matrix \( T \):

\[ T = -R \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \]  \hspace{1cm} (12)

Using \( R \) and \( T \), we can get the normalized data \( \{X(n), y(n)\}_{n=1}^{N} \)

### 1.3 Feature Extraction

Here we use both position and motion as the feature.

#### 1.3.1 Compute the Motion

We use the position at time \( t \) subtract the position at time \( t-1 \) as the motion for time \( t \):

\[ M_t = X_t - X_{t-1} \]  \hspace{1cm} (13)

where \( M_t \) represents the Motion of joint at time \( t \) and \( X_t \) represents the position at time \( t \).

In this way, if the length of the sequence is \( T \), we can only compute \( T - 1 \) motion because at time 1 we don’t have the previous position to subtract. So the feature data have the dimension of \( D \times T - 1 \)

#### 1.3.2 Construct the Feature

Combine the position and the motion from \( t = 2 \) to \( t = T \) we have the feature:

\[ F_t = \begin{bmatrix} x_t \\ y_t \\ z_t \\ m_{x,t} \\ m_{y,t} \\ m_{z,t} \end{bmatrix} \]  \hspace{1cm} (14)

where \( x_t, y_t, z_t \) are normalized position coordinates

So, we got out featured data: \( \{x_f(n), y(n)\}_{n=1}^{N} \). in the experiment here(kinect), we have totally 25 joints, so \( x_f(n) \) has a dimension of \( 150 \times (T - 1) \), where \( T \) is the length of the
2 Training the HMM for Each Class

2.1 Divide the Dataset

First divide the dataset into two parts: training data and testing data. Then assign the labels to each of these sequences. Simultaneously, we need to divide them by their class. Here is the whole process.

2.1.1 Collecting Raw Data

Here, use the Kinect v2 to collect the data, there are totally 25 joints. I can select the sampling frequency and time length. What we get is the 3D coordinates of these 25 joints in the camera frame ($\{x_c(n)\}_{n=1}^N$). These data contains some frames which are not relevant to the gesture. Then these frames should be processed or they may influence the result of the training.

For each data sequence $x_c(n)$, it has the dimension of $3 \times T(n)$, where $T(n)$ is the length of the nth sequence. The are stored in bvh format (biovision hierarchy animation), csv or txt format. After having their labels, construct a structure format to further more store the data.

2.1.2 Incorporate Variance in the Data

To make the model more generalized, we need to use data with different kinds of variances for one class of gesture to train the model. Here I introduce several types of variances:

(1) Distance: doing the gestures in different at differences from the camera.

(2) Rotation: doing the gestures in different directions to the camera. For example, facing directly to the camera, facing back to the camera, facing left to the camera, facing right to the camera. And other special directions.

(3) Different People: different may doing the same gesture in different patterns, so having different people doing gestures to train the model can improve the generalization ability of the model.

2.1.3 Segment A Gesture in A Sequence

This is the process to select a gesture in a sequence. Here I use the Brekel Pro Body to collect the data. The data is the 3D coordinates of the joints. These coordinates are estimated by the Depth images and RGB images from the Kinect.
Figure 2: Distance Variance

Figure 3: Rotation Variance

Figure 4: People Variance
The length (number of frames) of sequences varies from 57 to 110. Apart from the starting frames and ending frames. Select frames from the central of the sequences. For example, the length of the sequence collected is $T(n)$. I cut the sequence from the gesture begin to the gesture end.

### 2.2 Initialization of Parameters

An HMM model contains three parts:

1. $\pi$: the initial distribution of hidden states;
2. $A$: transition matrix;
3. $B$: observation matrix (for discrete observation), and if the observation is continuous, $B$ becomes a distribution (which is the case we will use here). As the observation is continuous, here we use Gaussian Mixture Model (GMM) to model the emission probability. We use $\lambda = (\pi, A, B)$ to represent the model (parameters).

In this document, use $x_t$ as the observation at time $t$ and $y_t$ as the hidden state at time $t$.

#### 2.2.1 Decide the hidden states number

As hidden states are unknown, we need to first decide how many hidden states we will use. Here we assume there are $Q$ possible hidden states.

$$\text{Hidden States Number} = Q \quad (15)$$

#### 2.2.2 Do the clustering to initialize the hidden state

In order to initialize the prior probability, we need to initialize the hidden state. Assume the number of class is: $K$

These classes are: $c_1, c_2, ..., c_K$

The number of sequences for each class is: $N^k$, $k = 1, 2, \ldots K$

The $i$th sequence (a matrix) in class $k$: $X^k_i = X^k_i(j)_{j=1}^{N^k_i}$

where $j$ represents the jth column vector of the sequence (a matrix) and $N^k_i$ represents the number of vectors (columns) of the sequence (matrix).

So, the total number of these vectors we have for these $K$ classes is: $\sum_{k=1}^{K} \sum_{j=1}^{N^k_i} \sum_{i=1}^{N^k_i} 1$. We do the clustering on these vectors.

Assume the number of hidden states is $Q$, after clustering, each of these vectors belongs to one of these $Q$ clusters. In another word, each of these vectors has a label from 1 to $Q$. Then we can use these hidden states to do the initialization for the prior distribution and the transition probability.
Here we use k-means to do the clustering and assign the hidden states labels to these states. Each of these $Q_1$ samples has the dimension of 150. After we do the clustering for these hidden states, each of these states has a class from 1 to $Q$.

### 2.2.3 Initialize the prior probability $\pi$

After finishing the clustering, we need to initialize the parameters for each class.

For class $k$ ($k$ from 1 to $K$):

We initialize the prior probability as:

$$\pi^k_i = \frac{N^k_i}{N^k}, \quad i = 1, 2...Q$$  \hspace{1cm} (16)

where $N^k_i$ is the number of sequences which belongs to class $k$. And $N^k_i$ is the number of sequence whose first frame is $i$.

### 2.2.4 Initialize the Transition matrix $A$

We already got the initial hidden states. We need to find the initial distribution of the $Q \times Q$ matrix $A$. For each class $k$, we can initialize $A$ by just counting the transition between states:

$$A^k_{ij} = \frac{N^k_{ij}}{\sum_{j=1}^{Q} N^k_{ij}}$$  \hspace{1cm} (17)

### 2.2.5 Initialize the Emission Matrix $'B'$

In this part, we use Gaussian Mixture Model to model the emission probability. Assume we have $M$ Gaussian Model to constitute the Gaussian Mixture Model.

We denote the $B$ as below:

$$B(j, t) = \sum_{m} M(j, m)N(x(t), \mu(j, m), \Sigma(j, m))$$  \hspace{1cm} (18)

where

- $R(j, m)$: represents the weight(probability) of the $m$th Gaussian of hidden state $j$
- $\mu(j, m)^{D \times 1}$: represents the mean of the $m$th Gaussian of hidden state $j$
- $\Sigma(j, m)^{D \times D}$ represents the covariance matrix of the $m$th Gaussian of the hidden state $j$

Here we create three matrix to store them:

- $R : Q \times M$
- $\mu : Q \times M \times D$
- $\Sigma : Q \times M \times D \times D$
We have the data for each class. Here we use k-means to decide the mean of each Gaussian model. Then we calculate the covariance of each Gaussian model based on the clustering result.
Assume the number of Gaussian mixture model is M
For each class, we have their means and covariances:
\[ \mu_1, \mu_2, \ldots, \mu_M \]
\[ \Sigma_1, \Sigma_2, \ldots, \Sigma_M \]
Similarly as initializing the A matrix,
The number of sequences for each class is: \( N^k \), \( k = 1, 2, \ldots, K \)
The ith sequence(a matrix) in class k: \( X^k_i = X^k_i(j)_{j=1}^{N^k} \)
where j represents the jth column vector of the sequence(a matrix) and \( N^k_i \) represents the number of vectors(columns) of the sequence(matrix)
So, the total number of vectors we used to do clustering for class k is: \( \sum_{i=1}^{N^k} \sum_{i=1}^{N^k} 1 \)
We do the clustering on these vectors, assign the the center of each clustering to the mean of each Gaussian model, then calculate the covariance matrix of each Gaussian model based on each clusters.

2.3 Using EM algorithm to train the model

2.3.1 General idea of HMM training

The HMM training is the optimization of parameters to increase the likelihood, or log-likelihood of the model. In another word, we have the training data 
\( D = \{x_1, x_2, \ldots, x_N \} \), we want to maximize \( \log P(D|\lambda) \), where \( \lambda = \{\pi, A, B\} \)

\[
\log P(D|\lambda) = \log \prod_{i=1}^{N} P(x_i|\lambda) \\
= \sum_{i=1}^{N} \log P(x_i|\lambda) \\
= \sum_{i=1}^{N} \log \sum_{y_i} P(x_i, y_i|\lambda) \\
= \sum_{i=1}^{N} \log \sum_{y_i} P(x_i, y_i|\pi, A, B)
\]

where \( y_i \) is the ith hidden state sequence i.e. \( y_i = y_i1, y_i2, \ldots, y_{it} \)
To maximize it, there is no closed-form solution for the parameters.

2.3.2 Why not using the maximum likelihood estimation?

We know the usual way to train a probabilistic model is to use maximum likelihood estimation. We first set up a likelihood function, and then maximize it with respect to the parameters of the model. Instead of maximizing the likelihood, we usually maximize the
log-likelihood which is usually easier (log function is a monotonically increasing function).

We use EM algorithm because we want to do the maximum likelihood estimation but cannot. Because we have to sum over latent variables but we don’t know them:

\[
P(x) = \sum_{y_i} P(y_i, x)
\]

(20)

\[
\log P(x) = \log \sum_{y_i} P(y_i, x)
\]

(21)

where \( y \) is the hidden state and \( x \) is the observation.

2.3.3 Idea of EM algorithm

Here, we use EM algorithm to train the model.

Generally speaking, instead of maximizing \( P(x) \), we find another function which is a lower bound of \( P(x) \).

In the E-step, we create functions for the expectation of the log-likelihood evaluated using the current estimate for the parameters. We calculate the posterior probability:

\[
q(y) = P(y|x)
\]

(22)

Then, we get the lower bound of the log-likelihood:

\[
\log P(x) = \log \sum_{y} P(x, y)
\]

\[
= \log \sum_{y} P(x, y) \frac{q(y)}{q(y)}
\]

\[
= \log \sum_{y} q(y) \frac{P(x, y)}{q(y)}
\]

\[
= \log \mathbb{E}_{q(y)}[\frac{P(x, y)}{q(y)}]
\]

\[
\geq \sum_{y} q(y) \log \frac{P(x, y)}{q(y)} \quad (\text{Jensen's inequality})
\]

\[
= \sum_{y} q(y) \log P(x, y) - \sum_{y} q(y) \log q(y)
\]

Note the second term is the entropy. So

\[
\max \log P(x)
\]

(24)

is equivalent to

\[
\max \sum_{y} q(y) \log P(x, y) = \mathbb{E}_{q(y)}[\log P(x, y)]
\]

(25)
this is the expected log-likelihood.
In the M-step, we compute parameters maximizing the expected log-likelihood found on
the E-step. These parameter-estimates are then used to determine the distribution of
latent variables in the next E step. This can be expressed as below:

$$
\pi, A, B = \arg \max_{\pi, A, B} \sum_y q(y) \log \frac{P(y|x)}{q(y)} \\
\text{s.t. } \sum_{i=1}^{Q} \pi_i = 1 \\
\text{s.t. } \sum_{j=1}^{Q} A_{ij} = 1 \quad \forall i = 1, 2, ... Q \\
\text{s.t. } \sum_{k=1}^{K} B_{jk} = 1 \quad \forall j = 1, 2, ... Q
$$

(26)

Then I simplify the objective function:

$$
\pi, A, B = \arg \max_{\pi, A, B} \sum_y q(y) \{ \log P(y, x) - \log q(y) \}
$$

(27)

$$
q(y) \text{ can be removed from the objective function because it does not depend on model parameters:}
$$

$$
\pi, A, B = \arg \max_{\pi, A, B} \sum_y q(y) \{ \log P(y, x) \}
$$

(28)

Then, we plug in the probabilities of the HMM:

$$
\pi, A, B = \arg \max_{\pi, A, B} \sum_y q(y) \log P(y, x)
$$

(29)

Then we plug in the parameters and use Lagrange multiplier method to get the optimal parameters.

Prior probability:

$$
\pi_i = \frac{q(y_1)1(y_1 = i)}{\sum_{i=1}^{Q} q(y_1)1(y_1 = i)}
$$

(30)

where 1() is the indicator function.

Transition matrix:

$$
A(i, j) = \frac{\sum_y q(y) \sum_{t=2}^{T} 1(y_t = j, y_{t-1} = i)}{\sum_y q(y) \sum_{t=2}^{T} 1(z_{t-1} = i)}
$$

(31)
Emission matrix:

\[
B(j, k) = \frac{\sum_y q(y) \sum_{t=1}^{T} 1(y_t = j, x_t = k)}{\sum_y q(y) \sum_{t=1}^{T} 1(y_t = j)}
\] (32)

Then we have to deal with \(q(y)\), here is when we need to use the Baum-Welch algorithm, here I solve matrix \(A\).

The nominator:

\[
\sum_y q(y) \sum_{t=2}^{T} 1(y_t = j, y_{t-1} = i)
\]

\[
= \sum_y q(y) \sum_{t=2}^{T} 1(y_t = j, y_{t-1} = i)
\]

\[
= \sum_{t=2}^{T} \sum_y 1(y_t = j, y_{t-1} = i) q(y)
\]

\[
= \sum_{t=2}^{T} \sum_y \ldots \sum_{y_T} P(y_1, y_2, \ldots, y_T | x)
\]

\[
= \sum_{t=2}^{T} P(y_t = j | x)
\] (33)

The denominator:

\[
\sum_y q(y) \sum_{t=2}^{T} 1(y_{t-1} = i)
\]

\[
= \sum_y q(y) \sum_{t=2}^{T} 1(y_{t-1} = i)
\]

\[
= \sum_{t=2}^{T} P(y_{t-1} = i | x)
\] (34)

We do the same thing \(\pi, A, B\), then these probabilities we will define and use in the Baum-Welch algorithm.

This is for the discrete situation. When the observation is continuous, \(B\) becomes a distribution.

We do E-step and M-step iteratively until convergence.

Here, we use the EM algorithm (a special kind of EM algorithm) during the training.

The method below is for only one class \(k\):

To compute the posteriori probability which is used in the algorithm. Define \(T' = T - 1\)

Define forward probability \(\alpha\) and backward probability \(\beta\) as below:

\[
\alpha^k(t, j) = P(x_1^k, x_2^k, \ldots, x_t^k, y_t^k = j | \lambda_k)
\] (35)

\[
\beta^k(t, j) = P(x_{t+1}^k, x_{t+2}^k, \ldots, x_{T'}^k | y_t^k = j, \lambda_k)
\] (36)

We compute them as follow:
For $\alpha^k$, we first initialize them as:

$$\alpha^k(1, j) = \pi_j^k B^k(j, 1), 1 \leq j \leq Q$$  \hspace{1cm} (37)

then do

$$\alpha^k(t, j) = \sum_{i=1}^{Q} \alpha^k(t-1, i) A^k(i, j) B^k(j, t), 1 \leq j \leq Q, 1 < t < T'$$  \hspace{1cm} (38)

For $\beta$, first initialize $\beta$ in the last time point:

$$\beta^k(T', j) = 1, 1 \leq j \leq Q$$  \hspace{1cm} (39)

Then do the backward:

$$\beta^k(t, i) = \sum_{j=1}^{Q} A^k(i, j) B^k(j, t+1) \beta^k(j, t+1), 1 \leq j \leq Q, 1 \leq t \leq T'$$  \hspace{1cm} (40)

2.3.4 E-step

The goal of the E step is to compute the posteriori probability which will be used in the M-step. Here we use the forward-backward to compute the posteriori probability.

First we define:

$$\xi_t(i, j) = P(y_t = i, y_{t+1} = j|x)$$  \hspace{1cm} (41)

This is the posteriori probability of the transition. In the E-step, we compute $\xi$ as below:

$$\xi_t(i, j) = \frac{\alpha(t, i) A(i, j) B(j, x_{t+1}) \beta(t+1, j)}{\sum_{i=1}^{Q} \sum_{j=1}^{Q} \alpha(t, i) A(i, j) B(j, x_{t+1}) \beta(t+1, j)}$$  \hspace{1cm} (42)

$$\gamma(j, m, t) = \frac{\alpha(t, j) \beta(t, j) \cdot R(j, m) N(x(t), \mu(j, m), \Sigma(j, k))}{\sum_{j'=1}^{Q} \alpha(t, j') \beta(t, j') \cdot \sum_{m'=1}^{M} R(j, m') N(x(t), \mu(j, m'), \Sigma(j, m'))}$$  \hspace{1cm} (43)

This is to update the emission distribution ‘B’.

2.3.5 M-step

The goal of of M step is to use the posteriori computed in the E-step to update the parameters to maximum the expection(increasing the log-likelihood). Assume we have $N^k$ observation sequence(data) for class k training. Update the parameters:

$$\pi_i^k = \frac{1}{N^k} \sum_{n=1}^{N^k} \frac{\alpha_n(1, i) \beta_n(1, i)}{p(n)}$$  \hspace{1cm} (44)

where $p(n)$ is the probability of the $n$th training data. It is just the likelihood or the posteriori computed from the forward algorithm.
The likelihood ($p(n)$) is the cost function we have, so we sum over all the probability to use as as the cost function:

$$cost = \sum_{n=1}^{N} \log(p(n))$$  \hspace{1cm} (46)

Because some of the observation you would like to model is very improbable. Some of the observation we model is very improbable, then we give updates based on that observation more weight.

We update the emission(parameters of Gaussian Mixture Model):

$$R^k(j, m) = \frac{\sum_{t=1}^{T'} \gamma(j, m, t)}{\sum_{t=1}^{T'} \sum_{m'=1}^{M} \gamma(j, m', t)}$$  \hspace{1cm} (48)

$$\mu^k(j, m) = \frac{\sum_{t=1}^{T'} \gamma(j, m, t)x_t}{\sum_{t=1}^{T'} \gamma(j, m, t)}$$  \hspace{1cm} (49)

$$\Sigma^k(j, m) = \frac{\sum_{t=1}^{T'} \gamma(j, m, t)(x_t - \mu(j, m))(x_t - \mu(j, m))^T}{\sum_{t=1}^{T'} \gamma(j, m, t)}$$  \hspace{1cm} (50)

After we train the model, we have the parameter of the model: $\pi$, $A$, $B$. Then we can do the gesture recognition based on the new data.

3 Gesture recognition

This work is actually computing the likelihood of each class and pick the maximum likelihood class as the gesture class. In another word, we compute the

$$k^* = \arg\max logp(x|\lambda_i), \hspace{0.5cm} 1 \leq i \leq Q$$  \hspace{1cm} (51)

To compute the likelihood, we can use the forward algorithm. We compute the likelihood for each sample, and then sum over all the likelihood of these samples for each class to make the prediction.

First, compute the emission probability:

$$B(j, t) = \sum_{m=1}^{M} R(j, m)N(x_t, \mu(j, m), \Sigma(j, m))$$  \hspace{1cm} (52)
Then initialize $\alpha$:
\[
\alpha(1, j) = \pi_j B(j, 1), 1 \leq j \leq Q
\]  
(53)

then do
\[
\alpha(t, j) = \sum_{i=1}^{Q} \alpha(t-1, i) A(i, j) B(j, t), 1 \leq j \leq Q, 1 < t < T'
\]  
(54)

So
\[
P(x_1, ...x_T | \lambda) = \sum_{i=1}^{Q} \alpha(T, i)
\]  
(55)

We calculate for each sample, then sum them together of each $\lambda_i$, $1 \leq i \leq Q$, we got $P_i, 1 \leq i \leq Q$. If $P_i$ is the largest one, then we predict the sample as class $i$. After we make the prediction we can get the accuracy on the test set. Note at first we set the number of hidden state and number of Mixture Gaussian. We can change the number of them and retrain the model to evaluate and pick the best number of them.

\[
Q^* = \arg \max_Q P(x_1, ..., x_t | \lambda, Q)
\]  
(56)