Lecture 8: Hidden Markov Models (I)

Kai Yu

SpeechLab
Department of Computer Science & Engineering
Shanghai Jiao Tong University

Autumn 2014
Table of Contents

- Isolated word recognition
  - Deterministic approach: Dynamic Time Warping
  - Probabilistic approach: Stochastic process
- Markov chains
- Hidden Markov Models
  - Evaluation: Forward algorithm
  - Decoding: Viterbi algorithm
Isolated Word Recognition

Recap

(a) Training

Training Examples

one  
1. 
2. 
3. 

two  

three  

(b) Recognition

Unknown $O = \begin{array}{cccc}
\vspace*{-5pt}
\end{array}$

$P(O|M_1)$  
$P(O|M_2)$  
$P(O|M_3)$

Choose Max
Deterministic approach

All process is based on extracted feature $\mathbf{O} = [o_1, \cdots, o_T]$ from waveform

1. Extract feature of the unknown word
2. Compute the distance between the unknown feature vector and each reference vector $D(\mathbf{O}, \mathbf{V}_k)$
3. Find the word with minimum distance

$$\hat{k} = \min_k D(\mathbf{O}, \mathbf{V}_k)$$

**Difficulty:** How to calculate distance between invariable length sequences?
Dynamic Time Warping
Dynamic Programming using a \textit{real-valued} cost function

To compute DTW between sequence $X = [x_1, \ldots, x_M]$ and $Y = [y_1, \ldots, y_N]$, define a $M \times N$ distance matrix $D$, such that

$$D(i, j) = \min \begin{cases} 
D(i - 1, j) + C_I & \text{insertion} \\
D(i, j - 1) + C_D & \text{deletion} \\
D(i - 1, j - 1) + d(x, y) & \text{substitution}
\end{cases}$$

where $C_I$ and $C_D$ are the costs for insertion and deletion and $d(x, y) = \sqrt{(x - y)^\top (x - y)}$ is the substitution cost.
Dynamic Time Warping for Speech Recognition

Pros and Cons

DTW finds the best alignment between two data sequences and compute the distance between these sequences. It is used as a simple template matching for speech recognition:

1. **Training**: record an example speech for each word as a template
2. **Decoding**: record a testing speech and perform DTW against the template

**Pros**: Simple to train, just one example

**Cons**: Unreliable. Highly dependent on the template, hard to generalize.

**Q**: How to deal with multiple examples effectively?
Probabilistic Approach for Isolated Word Recognition

Statistical speech recognition:

\[
\hat{W} = \arg \max_{W} P(W|O) = \arg \max_{W} p(O|W)P(W)
\]

**Isolated word recognition**: assume all words are equally likely, the problem is just to pick the word with highest acoustic likelihood.

\[
k = \arg \max_{i} p(O|w_i)
\]

where \(p(O|w_i)\) performs feature vector length normalization.

**Hidden Markov Model (HMM)** is commonly used as \(p(O|w_i)\)
A Markov chain is a stochastic process that allows to generate a random sequence that is discrete in both time and value. The output of the process is a state sequence of length $T$ given by

$$\mathbf{x} = [x_1, x_2, \cdots, x_T]$$

where state $x \in \{s_1, \cdots, s_M\}$,

$$P(\mathbf{x}) = P(x_1) \prod_{t=2}^{T} P(x_t|x_1, \cdots, x_{t-1})$$

First-order Markov approximation: $x_t$ only depends on $x_{t-1}$

$$P(\mathbf{x}) = P(x_1) \prod_{t=2}^{T} P(x_t|x_{t-1})$$
Parameter Set of Markov Chain

The alphabet defines the states and the random sequence is generated by moving from one state to another state. The selection of the state is governed by the transition probabilities.

- **Initial state probabilities**

  \[ \pi_i = P(s_i) \sum_i P(s_i) = 1 \quad 1 \leq i \leq M \]

- **State transition probabilities**

  \[ a_{ij} = P(x_t = s_j | x_{t-1} = s_i) \quad 1 \leq i \leq M, \quad 1 \leq j \leq M \]

  \[ \sum_j P(x_t = s_j | x_{t-1} = s_i) = 1 \quad 1 \leq j \leq M \]

**Stationary Markov chain:** The parameters are constant (not a function of time)
Example of Markov Chain

A Markov chain can be visualised, e.g. a 3 state Markov chain with the alphabet $S = \{1, 2, 3\}$.

Q: How to draw the transition of higher order Markov chain?
Example of Markov Chain
Take Balls Out of Barrels

Randomly select a barrel and take a ball from the barrel.

A Markov chain can be used to model the ball sequence.
Randomly select a barrel and take a ball from the barrel.

A hidden Markov model (HMM) can be used to model the ball sequence.
Hidden Markov Model (HMM)

- **Markov chain**: Output at each state is *deterministic*
  \[ o = s \]

- **Hidden Markov model**: Output at each state is random and governed by a probability distribution
  \[ b(o) = P(o|s) \]

HMM is a *finite state transducer* to which takes in a sequence of feature vectors (representing the speech waveform) and transducing them into a state sequence representing the phones, syllables or words.
Example of HMM

Depending on the nature of **observation**, HMM can be:

- **Discrete**: \( b(o) = P(o|s) \), e.g.: Finance
- **Continuous**: \( b(o) = p(o|s) \), e.g.: Speech

Two types of **sequences** in HMM

- **State sequence**: hidden, underlying clue
- **Observation sequence**: actual features observed
Example of HMM Alignment
How variable length time series are modelled by HMM

Consider a sequence of discrete symbol \([a, b, c]\) to be modelled by a 3-state left-to-right HMM

Note: Seq 4 has a low likelihood as the trend is \(c \rightarrow b \rightarrow a\), which is reverse to \(a \rightarrow b \rightarrow c\) as described in the model.
HMM Topology Examples

- Left-to-right

- With skips

- Fully connected
HMM for Modelling Speech

Speech Signals

- consists of quasi-stationary segments
- have finite length
- can be reasonably well described by concatenation of small sound units ("phones")

Left-to-right HMMs can be used to model these properties:

Note that the initial and final state of the HMM are non-emitting.
HMM for Pattern Matching
Comparison to DTW

- Probabilistic model as the *template*
- Cost function is likelihood
- States are hidden and multiple alignments allowed

Train HMM From Multiple Examples

Align test waveform with trained HMM
HMM Formulation and Model Assumption

**HMM Formulation:**
- **Hidden states:** \( Q = \{q_i, \ 1 \leq i \leq S\} \)
- **Transition probabilities:** \( A = \{a_{ij}, \ 1 \leq i \leq S\} \)
- **State output distribution:** \( B(o) = \{b_j(o), \ 1 \leq j \leq S\} \)

\[
a_{ij} = P(q_t = j | q_{t-1} = i) \quad b_j(o_t) = p(o_t | q_t = j)
\]

**Model Assumption:**
- **Markov assumption:** Instantaneous transition
  Given current state \( q_t \), transition to the next state \( q_{t+1} \) is independent of historical states
- **Conditional independence assumption:**
  Given current state \( q_t \), observation \( o_t \) is independent of historical states and observations.
Key Problems in HMM for Isolated Word Recognition

The use of HMM for speech recognition requires answers to the following questions:

- **Evaluation**: How can we compute the overall likelihood of a given observation sequence \( O = [o_1, \cdots, o_T] \)?

\[
    p(O|\theta) = ?
\]

- **Decoding**: How to find the most likely state sequence given the observation sequence?

\[
    \hat{Q} = \max_Q p(O, Q|\theta)
\]

- **Estimation**: How to find optimal model parameters?

\[
    \hat{\theta} = \max_\theta \mathcal{L}(\theta)
\]

where \( \mathcal{L}(\theta) \) is a specific criterion.
State alignment is known:

\[ p(O|q, \theta) = \prod_{t=1}^{T} p(o_t|q_t, \theta) \]

State alignment is unknown:

\[ p(O|\theta) = \sum_{q} p(O|q, \theta) P(q|\theta) = \sum_{q} \left( \prod_{t=1}^{T} p(o_t|q_t, \theta) P(q_t|q_{t-1}) \right) \]

where \( P(q|\theta) = \prod_{t=1}^{T} P(q_t|q_{t-1}) \) is the prior for each state sequence and let \( P(q_0) = 1 \).
Example of HMM Evaluation

For observation \( o = [1, 1, 2, 3] \), the possible state sequences are

\[
q_1 = [1, 2, 2, 3, 4, 5], \quad q_2 = [1, 2, 3, 3, 4, 5], \quad q_3 = [1, 2, 3, 4, 4, 5], \\
q_4 = [1, 3, 3, 3, 4, 5], \quad q_5 = [1, 3, 3, 4, 4, 5], \quad q_6 = [1, 3, 4, 4, 4, 5].
\]

The total likelihood of this observation sequence is given by

\[
p(O|\theta) = \sum_{i=1}^{6} p(O|q_i, \theta) P(q_i|\theta)
\]
Forward algorithm
Efficient HMM Likelihood Calculation

**Likelihood** of observation sequence \( O^T_1 = [o_1, \ldots, o_T] \) is

\[
p(O^T_1 | \theta) = \sum_{q} p(O^T_1, q, q_0 = 1, q_{T+1} = N | \theta)
\]

where start and end *non-emitting* states are \( q_0 = 1 \) and \( q_{T+1} = N \) respectively. The above likelihood can be computed efficiently using **forward recursion** with the below forward probability

\[
\alpha_j(t) = p(O^t_1, q_t = j | \theta) = \sum_{i=1}^{N-1} p(O^t_1, q_t = j, q_{t-1} = i | \theta)
\]

\[
= p(o_t | q_t = j, \theta) \sum_{i=1}^{N-1} P(q_t = j | q_{t-1} = i) p(O^{t-1}_1, q_{t-1} = i | \theta)
\]

\[
= b_j(o_t) \sum_{i=1}^{N-1} a_{ij} \alpha_i(t - 1)
\]
Forward algorithm

Boundary condition and final output

Forward probability $\alpha_j(t)$ is defined for $1 \leq t \leq T$ and $1 \leq j \leq N$.

**Boundary conditions:**

- Initial state is fixed as $q_0 = 1$

$$\alpha_j(0) = \begin{cases} 1 & j = 1 \\ 0 & \text{otherwise} \end{cases}$$

- Final state must be the non-emitting state $q_{T+1} = N$

$$p(O_T^T | \theta) = p(O_T^T, q_{T+1} = N | \theta)$$

$$= \alpha_N(T + 1) = \sum_{i=1}^{N-1} a_{iN} \alpha_i(T)$$
float forward(float a[][], float b[][]) {
    int N = a.length, T = b.length;
    float alpha[][] = new float[N][T+1];
    for (int j=0; j<N; j++) alpha[j][0] = (j==0) ? 1 : 0;
    for (int j=0; j<N; j++)
        for (int t=1; t<=T; t++) {
            alpha[j][t] = 0;
            for (int i=0; i<N; i++) alpha[j][t] += a[i][j]*alpha[i][t-1];
            alpha[j][t] *= b[t][j];
        }
    float likelihood = 0;
    for (int j=0; j<N; j++) likelihood += alpha[j][T];
    return likelihood;
}
Forward algorithm can be directly used for isolated word recognition.
Decoding Using HMM

Evaluate the most likely state sequence

**Evaluation:**
- **Goal:** Expected likelihood over all possible state alignments

\[
p(O|\theta) = \sum_q p(O, q|\theta)
\]

- Word (sequence) for the feature sequence is normally known
- Computationally expensive for unknown word (sequence)

**Decoding:**
- Approximate overall likelihood using the best state alignment

\[
p(O|\theta) \approx \max_q p(O, q|\theta)
\]

- Highly efficient and allow Dynamic Programming to be used
- **Goal:** Find the most likely state sequence

\[
\hat{q} = \arg \max_q p(O, q, q_0 = 1, q_{T+1} = N|\theta)
\]
Viterbi Algorithm
Dynamic programming for decoding HMM

Given \( O_T^T = [o_1, \cdots, o_T] \) and \( q_T^T = [q_1, \cdots, q_T] \),

Viterbi algorithm is to recursively find

\[
\hat{q}_T^T = \arg \max_{q_T^T} p(O_T^T, q_T^T, q_0 = 1, q_{T+1} = N | \theta)
\]

Similar to \( \alpha_j(t) \) in Forward algorithm, at time \( t \), we define

\[
\phi_j(t) = \max_{q_t^T} p(O_t^t, q_0 = 1, q_{t-1}^{t-1}, q_t = j | \theta)
\]

\[
= \max_{1 \leq i \leq N} \max_{q_t^T} p(O_t^t, q_0 = 1, q_{t-2}^{t-2}, q_{t-1} = i, q_t = j | \theta)
\]

\[
= p(o_t | q_t = j, \theta) \max_{1 \leq i \leq N} P(q_t = j | q_{t-1} = i)
\]

\[
\max_{q_{t-1}^{t-1}} p(O_{t-1}^{t-1}, q_0 = 1, q_{t-2}^{t-2}, q_{t-1} = i | \theta)
\]

\[
= b_j(o_t) \max_{1 \leq i \leq N} a_{ij} \phi_i(t - 1)
\]
Viterbi Algorithm

Boundary condition and final output

Viterbi probability $\phi_j(t)$ is defined for $1 \leq t \leq T$ and $1 \leq j \leq N$.

**Boundary conditions:**

- Initial state is fixed as $q_0 = 1$

  \[
  \phi_j(0) = \begin{cases} 
  1 & j = 1 \\
  0 & \text{otherwise}
  \end{cases}
  \]

- Necessary to record the previous best state giving the best partial likelihood

  \[
  q_j^{\text{max}}(t) = \arg \max_{1 \leq i \leq N} a_{ij} \phi_i(t - 1)
  \]

- Final state must be the non-emitting state $q_{T+1} = N$

  \[
  \phi_N(T+1) = \max_{1 \leq i \leq N} a_{iN} \phi_i(T), \quad q_N^{\text{max}}(T+1) = \arg \max_{1 \leq i \leq N} a_{iN} \phi_i(T)
  \]
**Viterbi Algorithm**

**Trace back**

**Trace back**: find the best state sequence after Viterbi recursion.

- Start from the final state $\hat{q}_{T+1} = N$.
- The preceding state that generates this is $\hat{q}_T = q_{\hat{q}_{T+1}}^{\text{max}}(T)$.
- Similarly, the best state at $t-1$ that generates the best likelihood score at $t$ is
  
  $$\hat{q}_{t-1} = q_{\hat{q}_t}^{\text{max}}(t)$$

- The process can be repeated until $\hat{q}_0 = 1$ is obtained.
Viterbi Algorithm

Procedure

\[ \phi_j(t) = \max_{q_t} p(O_t, q_0 = 1, q_{t-1}, q_t = j|\theta) \]

1. Initialization

\[ \phi_j(0) = \begin{cases} 1.0 & j = 1 \\ 0 & 1 \leq j \leq N \end{cases} \quad \phi_1(t) = 0, \quad 1 \leq t \leq T \]

2. Recursion

for \( t = 1, 2, ..., T \)
for \( j = 2, 3, ..., N - 1 \)

Compute: \( \phi_j(t) = b_j(o_t) \max_{1 \leq i < N} (\phi_i(t - 1) a_{ij}) \)

Store: \( q_j^{\text{max}}(t) = \arg \max_{1 \leq i < N} (\phi_i(t - 1) a_{ij}) \)

3. Termination

\[ p(O, \hat{q}|\theta) = \max_{1 < k < N} (\phi_k(T) a_{kN}) \]

The most likely path can be recovered by tracing back using the predecessor information stored at each state \( q_j^{\text{max}}(t) \).
float viterbi(float a[][], float b[][], float qbest[]) {
    int N = a.length, T = b.length;
    float v[][] = new float[N][T+1];
    int qmax[][] = new int[N][T+1];
    for (int j=0; j<N; j++) alpha[j][0] = (j==0) ? 1 : 0;
    for (int j=0; j<N; j++)
        for (int t=1; t<=T; t++) {
            v[j][t] = b[t][j]*FindMax(a, v, j, t);
            qmax[j][t] = FindMaxArg(a, v, j, t);
        }
    qbest[T] = FindMaxArg(a, v, N, T);
    for (int t=T-1; t>=0; t++) qbest[t] = qmax[qbest[t+1]][t+1];
    return FindMax(a, v, N, T+1);
}
Given the above HMM and observed sequence $O = [1, 1, 2, 3]$:
Practical Issue in Viterbi Search
Log likelihood and pruning

**Log likelihood** is commonly used to avoid arithmetic underflow:

\[
\phi_j(t) = \max_i \{ \log \phi_i(t-1) + \log a_{ij} + \log b_j(o_t) \}
\]

**Pruning** is to cut paths which have relatively low likelihood so that the search space is kept small

- **Beam pruning:**
  1. At time \( t \), find log likelihood of the *best* path
     \[
     \phi^*(t) = \max_j \phi_j(t)
     \]
  2. Disable the paths with lower likelihood \( \phi_j(t) < \phi^*(t) - \tau \)
     where \( \tau \) is a predefined constant called **beam width**

- **Histogram pruning:**
  1. At time \( t \), find the histogram of the log likelihoods of all paths
  2. Select the top \( N \) paths and find the beam width \( \tau \) according to the \( N^{th} \) path’s likelihood
  3. Disable the paths with lower likelihood \( \phi_j(t) < \phi^*(t) - \tau \)
Viterbi algorithm only makes **local** decision - Efficient!

Paths merge and divide at roughly the same rate - No growth of search space

With pruning, time for the search is linear in the length of the observed data

Much more powerful extension to Dynamic Time Warping