# Digital Speech Processing
# Homework #1

## Implementing Discrete Hidden Markov Model

林子權 (modified from 馮子軒 2021DSP ver.)

October 05, 2022

*Due on 23:59, November 04, 2022*

# Outline

# HMM in Speech Recognition

# Speech Recognition 1/2

In acoustic model

- each word consists of syllables
- each syllable consists of phonemes
- each phoneme consists of some (hypothetical) states

"語音" →"語 (ㄩˇ) 音 (ㄧㄣ)" →"ㄩ" →$\{s_1, s_2, \ldots\}$

Each phoneme can be described by a HMM (acoustic model). Given a sequence of observation (MFCC vectors), each of them can be mapped to a corresponding state.

# Speech Recognition 2/2

Hence, there are

- state transition probabilities ($a_{ij}$) and
- observation distribution ($b_j[o_t]$)

in each phoneme acoustic model (HMM).

Usually in speech recognition we restrict the HMM to be a *left-to-right model*, and the observation distribution is assumed to be a continuous Gaussian mixture model.

# Review

- Left-to-right
- The observation distribution is a continuous Gaussian mixture model.

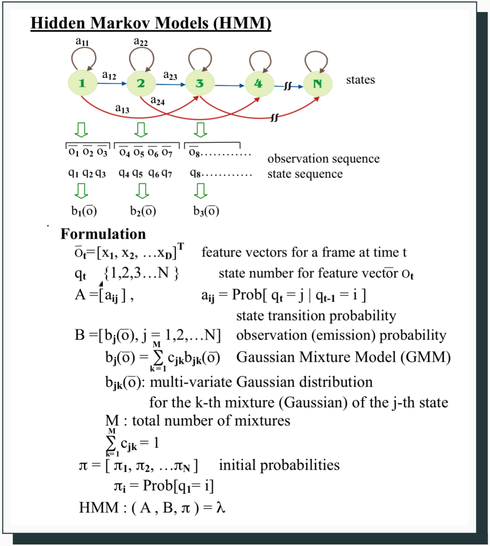



**Figure 1:** HMM from lecture 2.0

# General Discrete HMM

$$a_{ij} = P(q_{t+1} = j \mid q_t = i), \forall\, t, i, j \tag{1}$$

$$b_j(A) = P(o_t = A \mid q_t = j), \forall\, t, A, j \tag{2}$$

Given $q_t$, the probability distributions of $q_{t+1}$ and $o_t$ are completely determined. (independent of other states or observation)

# HW1 v.s. Speech Recognition

| | *Homework* | *Speech Recognition* |
|---|---|---|
| $\lambda$ **set** | 5 models | initial-final |
| $\lambda$ | model_01-05 | “ㄐ” |
| $\{\mathbf{o_t}\}$ | A, B, C, D, E, F | 39-dim MFCC |
| **unit** | an alphabet | a time frame |
| **observation** | sequence | voice wave |

# Homework of HMM

# Flowchart

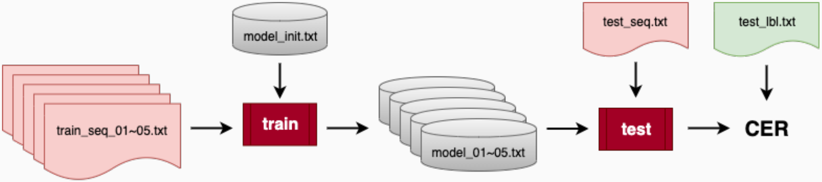



**Figure 2:** Training and testing models

# Problems of HMM

*Training*

- Basic problem 3 in lecture 4.0
  - Given observations $O$ and an initial model $\lambda = (A, B, \pi)$, adjust $\lambda$ to maximize $P(O \mid \lambda)$.

$$A_{ij} = a_{ij},\ B_{jt} = b_j[o_t],\ \pi_i = P(q_1 = i)$$

- Baum-Welch algorithm

*Testing*

- Basic problem 2 in lecture 4.0
  - Given $\lambda$ and $O$, find the best state sequences to maximize $P(O \mid \lambda, q)$.
- Viterbi algorithm

# Homework of HMM

## Training

# Training

- Basic problem 3
- *Baum-Welch algorithm*: A generalized expectation-maximization (EM) algorithm[1]
  - Calculate $\alpha$ (forward probabilities) and $\beta$ (backward probabilities) given the observations
  - Find temporary variables $\epsilon$ and $\gamma$ from $\alpha$ and $\beta$
  - Update model parameters $\lambda' = (A', B', \pi')$

[1] *http://en.wikipedia.org/wiki/Baum-Welch_algorithm*

*Forward algorithm*: define a forward variable $\alpha_t(i)$

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t,\ q_t = i \mid \lambda) \tag{3}$$

$$= \text{Prob}[\text{ observing } o_1, o_2, \ldots, o_t,\ \text{state } i \text{ at time } t \mid \lambda\ ] \tag{4}$$

Initialization

$$\alpha_1(i) = \pi_i b_i(o_1),\ 1 \le i \le N \tag{5}$$

Induction

$$\alpha_{t+1}(j) = \left[ \sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] \cdot b_j(o_{t+1}),$$

$$1 \le t \le T-1,\ 1 \le j \le N \tag{6}$$

Termination

$$P\left(\bar{O} \mid \lambda\right) = \sum_{i=1}^{N} \alpha_T(i) \tag{7}$$

# Backward Procedure

*Backward algorithm*: define a backward variable $\beta_t(i)$

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T \mid q_t = i, \lambda) \tag{8}$$
$$= \text{Prob}[\text{ observing } o_{t+1}, o_{t+2}, \ldots, o_T \mid \text{state } i \text{ at time } t,\ \lambda\ ] \tag{9}$$

Initialization

$$\beta_T(i) = 1,\ 1 \leq i \leq N \tag{10}$$

Induction

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\ b_j(o_{t+1})\ \beta_{t+1}(j),$$
$$t = \{T-1, T-2, \ldots, 1\},\ 1 \leq i \leq N \tag{11}$$

# Calculate $\gamma$

Define a temporary variable $\gamma_t(i) = P\left(q_t = i \mid \bar{O}, \lambda\right)$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} = \frac{P\left(\bar{O}, q_t = i \mid \lambda\right)}{P\left(\bar{O} \mid \lambda\right)} \tag{12}$$

**It should be a $T \times N$ matrix!**

# Calculate ε

The probability of transition from state *i* to state *j* given observation and model.

$$\epsilon_t(i,j) = P\left(q_t = i, q_{t+1} = j \mid \bar{O}, \lambda\right) \tag{13}$$

$$= \frac{\alpha_t(i)\ a_{ij}\ b_j(o_{t+1})\ \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)\ a_{ij}\ b_j(o_{t+1})\ \beta_{t+1}(j)} \tag{14}$$

$$= \frac{\text{Prob}\left[\bar{O},\ q_t = i,\ q_{t+1} = j \mid \lambda\right]}{P\left(\bar{O} \mid \lambda\right)} \tag{15}$$

**In total $T-1$ matrices (each $N \times N$)**

# Accumulate ε and γ

Recall $\gamma_t(i) = P\left(q_t = i \mid \bar{O}, \lambda\right)$

$$\begin{aligned}\sum_{t=1}^{T-1} \gamma_t(i) &= \text{expected number of times that state } i \\ &\quad \text{is visited in } \bar{O} \text{ from } t = 1 \text{ to } t = T-1 \qquad (16)\\ \sum_{t=1}^{T-1} \epsilon_t(i,j) &= \text{expected number of transitions from} \\ &\quad \text{state } i \text{ to state } j \text{ in } \bar{O} \qquad (17)\end{aligned}$$

# Re-estimate Model Parameters

Now we could update the parameters with εand γ

$$\lambda' = (A', B', \pi') \tag{18}$$

$$\pi_i' = \gamma_1(i) = p(\text{the first state is i}) \tag{19}$$

$$a_{ij}' = \frac{\sum_{t=1}^{T-1} \epsilon_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} = \frac{\mathbb{E}\text{ [ Number of transition from } i \text{ to } j\text{ ]}}{\mathbb{E}\text{ [ Number of visiting state } i\text{ ]}} \tag{20}$$

$$b_i'(k) = \frac{\sum_{O_t=k} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\mathbb{E}\text{ [ Number of observation } O = k \text{ in state } i\text{ ]}}{\mathbb{E}\text{ [ Number of visiting state } i\text{ ]}} \tag{21}$$

# Re-estimate Model Parameters with Multiple Sequences

With N training sequences, we should update parameters from all those sequences

$$\pi_i' = \frac{\sum_{n=1}^{N} \gamma_{n,1}(i)}{N} \tag{22}$$

$$a_{ij}' = \frac{\sum_{n=1}^{N} \sum_{t=1}^{T-1} \epsilon_{n,t}(i,j)}{\sum_{n=1}^{N} \sum_{t=1}^{T-1} \gamma_{n,t}(i)} \tag{23}$$

$$b_i'(k) = \frac{\sum_{n=1}^{N} \sum_{O_{n,t}=k} \gamma_{n,t}(i)}{\sum_{n=1}^{N} \sum_{t=1}^{T} \gamma_{n,t}(i)} \tag{24}$$

# Homework of HMM

## Testing

# Testing

- Basic problem 2
  - Given $\lambda$ and $O$, find the best state sequences to maximize $P(O \mid \lambda, q)$.
- Calculate $P(O \mid \lambda) \approx \max P(O \mid \lambda, q)$ for each of the five models
- The model with the highest probability for the most probable path usually also has the highest probability for all possible paths.

# Viterbi Algorithm

Complete procedure for Viterbi algorithm[2]

Initialization

$$\delta_1(i) = \pi_i b_i(o_1),\ 1 \leq i \leq N \tag{25}$$

Recursion

$$\delta_t(j) = \max_{1 \leq i \leq N} [\ \delta_{t-1}(i)\ a_{ij}\ ] \cdot b_j(o_t),\ 2 \leq t \leq T,\ 1 \leq j \leq N \tag{26}$$

Termination

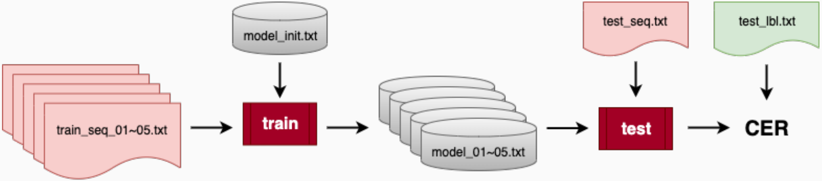
$$P^* = \max_{1 \leq i \leq N} [\ \delta_T(i)\ ] \tag{27}$$

$$\delta_t(i) = \max_{q_1, \ldots, q_{t-1}} P[\ q_1, q_2, \ldots, q_{t-1}, q_t = i, o_1, o_2, \ldots, o_t \mid \lambda\ ] \tag{28}$$

$$= \text{the highest probability along a certain single path ending at state } i \text{ at time } t \text{ for the first } t \text{ observations, given } \lambda \tag{29}$$

[2] `http://en.wikipedia.org/wiki/Viterbi_algorithm`

# Requirements


model_init.txt
test_seq.txt
test_lbl.txt
train_seq_01~05.txt
train
model_01~05.txt
test
CER

# Provided Files

*data/train_seq_0X.txt*

- Training data (10K observation sequences)

*data/test_lbl.txt*

- Testing labels

*data/test_seq.txt*

- Testing data (2.5K observation sequences)

*inc/hmm.h*

- Provided by TA, please work with it!
- You can load/dump models with functions within.

*model_init.txt*

- Initial model parameters

*modellist.txt*

- Paths to model files

*src/test_hmm.c*

- A showcase of the usage of *hmm.h*

```
dsp_hw1
├── data
│   ├── test_lbl.txt
│   ├── test_seq.txt
│   ├── train_seq_01.txt
│   ├── train_seq_02.txt
│   ├── train_seq_03.txt
│   ├── train_seq_04.txt
│   └── train_seq_05.txt
├── inc
│   └── hmm.h
├── Makefile
├── model_init.txt
├── modellist.txt
└── src
    └── test_hmm.c
```

# Input and Output of Your Program

*Training*

**Input**
1. number of iterations
2. initial model (***`model_init.txt`***)
3. observation sequences (***`train_seq_01~05.txt`***)

**Output** Five files of parameters for 5 models, each contains $\lambda = (A, B, \pi)$ (e.g. ***`model_01~05.txt`***)

*Testing*

**Input**
1. a file of paths to the models trained in the previous step (***`modellist.txt`***)
2. observation sequences (***`test_seq.txt`***)

**Output** best answer labels and $P(O \mid \lambda)$ (e.g. ***`result.txt`***)

# Training Details

```
./train <iter> <model_init_path> <seq_path> <output_model_path>
```

| | |
|---|---|
| *iter* | # of iterations |
| *model_init_path* | path to the initial model params |
| *seq_path* | path to sequence data |
| *output_model_path* | path to dump trained models |

# Testing Details

```
./test <models_list_path> <seq_path> <output_result_path>
```

| | |
|---|---|
| *models_list_path* | path to the model list file |
| *seq_path* | path to sequence data |
| *output_result_path* | path to output testing result |

# Program Execution Example

*Compiling*

```
make  # type this in the root directory of the project
```

*Training*

```
./train 100 model_init.txt data/train_seq_01.txt model_01.txt
```

*Testing*

```
./test modellist.txt data/test_seq.txt result.txt
```

**Notice!**

Command-line arguments are not fixed, read them during runtime. (e.g. Use `argv` in main function to pass the arguments.)

Also the paths in arguments need to be variable path.

# Requirements

## File Format

# Observation Sequence Format

The given `data/train_seq_01~05.txt` and `data/test_seq.txt` look like this.

```
ACCDDDDFFCCCCBCFFFCCCCCEDADCCAEFCCCACDDFFCCDDFFCCD
CABACCAFCCFFCCCDFFCCCCCDFFCDDDDFCDDCCFCCCEFFCCCCBC
ABACCCDDCCCDDDDFBCCCCCDDAACFBCCBCCCCCCCFFFCCCCCDBF
AAABBBCCFFBDCDDFFACDCDFCDDFFFFFCDFFFCCCDCFFFFCCCCD
AACCDCCCCCCCDCEDCBFFFCDCDCDAFBCDCFFCCDCCCEACDBAFFF
...
```

Each of the former has 10000 sequences and the latter has 2500 sequences.

# Model Format 1/2

Please make sure the format is the same as *`model_init.txt`*, namely, do not swap the row and column.

*`initial: 6`*

$$\pi = [\, \pi_1,\ \pi_2,\ \pi_3,\ \pi_4,\ \pi_5,\ \pi_6 \,]$$

*`transition: 6`*[3]

$$A = \begin{bmatrix} a_{11} & \dots & a_{16} \\ \vdots & \ddots & \vdots \\ a_{61} & \dots & a_{66} \end{bmatrix}$$

*`observation: 6`*[4]

$$B = \begin{bmatrix} b_1(o_1) & \dots & b_6(o_1) \\ \vdots & \ddots & \vdots \\ b_1(o_6) & \dots & b_6(o_6) \end{bmatrix}$$

[3] The sum of row is 1 here.

[4] The sum of column is 1 here.

# Model Format 2/2

A model file (e.g. `model_0X.txt`) should look like this and be able to be loaded by ‘loadHMM‘ in `inc/hmm.h`.

```
initial: 6
0.2     0.1     0.2     0.2     0.2     0.1

transition: 6
0.3     0.3     0.1     0.1     0.1     0.1
0.1     0.3     0.3     0.1     0.1     0.1
0.1     0.1     0.3     0.3     0.1     0.1
0.1     0.1     0.1     0.3     0.3     0.1
0.1     0.1     0.1     0.1     0.3     0.3
0.3     0.1     0.1     0.1     0.1     0.3

observation: 6
0.2     0.2     0.1     0.1     0.1     0.1
0.2     0.2     0.2     0.2     0.1     0.1
0.2     0.2     0.2     0.2     0.2     0.2
0.2     0.2     0.2     0.2     0.2     0.2
0.1     0.1     0.2     0.2     0.2     0.2
0.1     0.1     0.1     0.1     0.2     0.2
```

# Model List Format

The given *modellist.txt* looks like this.

```
model_01.txt
model_02.txt
model_03.txt
model_04.txt
model_05.txt
```

Your testing program should be able to read a list like this and load models from the specified paths for testing. (Don't worry! If you use *hmm.h*, all of these are done by calling function *load_models()*. For more details please refer to *hmm.h*.)

# Output Format

Your testing program should output these to the specific path (e.g. *result.txt*) given as a command-line argument while executing the program.

```
model_01.txt 7.822367e-34
model_05.txt 1.094896e-40
model_01.txt 7.928724e-33
model_02.txt 4.262100e-37
model_02.txt 5.914689e-42
...
```

Each line consists of the hypothesis model and its likelihood. They should be separated by a space.

# Label File Format

The first few lines of the given *data/test_lbl.txt* looks like this.

```
model_01.txt
model_05.txt
model_01.txt
model_02.txt
model_02.txt
...
```

# Makefile Format

The Makefile you submit should be capable to compile your program using *make*. The provided one can compile `train.c` and `test.c` in directory `src` into two executables `train` and `test`.

```makefile
.PHONY: all clean run
CC=gcc
CFLAGS=-std=c99 -O2
LDFLAGS=-lm
TARGET=train test

all: $(TARGET)

train: src/train.c
	$(CC) -o $@ $^ $(CFLAGS) $(LDFLAGS) -Iinc

test: src/test.c
	$(CC) -o $@ $^ $(CFLAGS) $(LDFLAGS) -Iinc

clean:
	rm -f $(TARGET)
```

# Report Format

Please write a **one-page** report in **PDF** format, name it *`report.pdf`* and submit with your source code.

State your name (the name on NTU cool, we have no way know your English name or any nickname), student ID and any challenges you encounter or attempts you try.

If your report contains your name and student ID, you will get all points of report. A good report may grant you bonus of extra 5%.

# File Structure

**All of your source code files must be placed under `inc/` and `src/`.**

Let's say you only have two implementation files and use the functions provided in *hmm.h*. You should put your source code under *src/* and leave *hmm.h* in *inc/*.

```
.
├── inc
│   ├── hmm.h
│   └── [other *.h or *.hpp]
├── Makefile
├── model_init.txt
├── report.pdf
└── src
    ├── test.c
    ├── train.c
    └── [other *.c, *.cc or *.cpp]
```

# Requirements

## Submission Requirement

# Submission Requirement 1/2

1. Create a directory named *hw1_[STUDENT_ID]*[5].
2. Put
   - *inc/*
   - *Makefile*
   - *model_init.txt*
   - *report.pdf*
   - *src/*

   into the directory. [6]
3. Compress the directory into a ZIP file named *hw1_[STUDENT_ID].zip*.
4. Upload this ZIP file to NTUCOOL.

[5] lowercase

[6] Put every source code file in *inc/* and *src/*.

# Submission Requirement 2/2

Let's say your student ID is *r01234567*.

```
hw1_r01234567.zip
└── hw1_r01234567
    ├── inc
    │   └── [*.h or *.hpp]
    ├── Makefile
    ├── model_init.txt
    ├── report.pdf
    └── src
        └── [*.c, *.cc or *.cpp]
```

# Grading

# Grading Method

Your training and testing program will be tested respectively. We will specify *`100`* as the number of iterations while testing your training program. And each of your program is allowed to run for *`1 min`*.

Here's TA's environment.

| | |
|---|---|
| Kernel | Linux 5.11.0-46-generic |
| Processors | AMD Ryzen 5 3600 6-Core Processor |
| RAM | 32 GB |
| GCC Version | 9.3.0 |

# Grading Policy

**File Format** 20%

- ZIP file name
- directory name
- separated header and implementation files
- `Makefile`
- `model_init.txt`

**Program** 20%

- compiled and executed without error (including the correctness of test)
- output files generated after execution

**Report** 10%

and bonus of extra 5% for the impressive ones

**Accuracy** 50%

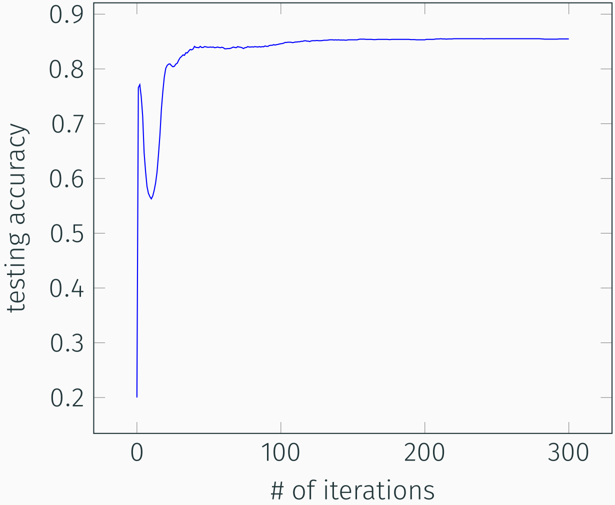
- only for the training part, not for the testing part
- 30% for provided test set, 20% for private test set[7]

[7] details are provided at the next page

# Training Accuracy vs Credit

| Accuracy | Credit |
|---|---|
| $80\% \leq \text{accuracy}$ | 100% |
| $60\% \leq \text{accuracy} < 80\%$ | 75% |
| $40\% \leq \text{accuracy} < 60\%$ | 50% |
| $\text{accuracy} < 40\%$ | 0% |

# Test Accuracy v.s. # of Training Iterations




*This figure is only for demo, and is not from the given public data.*

# Late Submission

*Due on 23:59, November 04, 2022*

You are still allowed to submit after the due date. The penalty for late submission is an exponential decay with decay rate 1.5%[8] of the maximum grade applicable for the assignment, for each hour that the assignment is late.

An assignment submitted more than 3 days after the deadline will have a grade of zero recorded for that assignment.

$$\mathrm{SCORE}_{final}(hr) = \begin{cases} \mathrm{SCORE}_{original} \times 0.985^{hr} & , \ hr \leq 72 \\ 0 & , \ hr > 72 \end{cases}$$

[8] less than 70% after 24 hrs, 48% for 48 hrs and 33% for 72 hrs

# Please Note...

*File Format*

- All of your source code files should be placed under ***`inc/`*** and ***`src/`***.
- None of your object file, executable files should present.
- ***`model_init.txt`*** must be submitted, even if it's not needed for your program.

*Program*

- Only *`C/C++`* is allowed.
- Make sure your program can be compiled with the ***`Makefile`*** you submit.
- The paths in command-line arguments have to support both absolute and relative path.
- Each of your program is allowed to run for 1 min. (each model have 1 min to train)

*Accuracy*

- Make sure your training program saves models within time limit.
- Public and private test set may sample from ***`different`*** distributions.

If you have any questions, please read the FAQ[9] first.

[9] http://speech.ee.ntu.edu.tw/DSP2022Autumn/hw1/FAQ.html

# DO NOT CHEAT

Any form of cheating, lying, or plagiarism will not be tolerated.

- *Discussion is encouraged, while sharing your code is not*
- *A public Github repo is not equal to an open source code*
- *Even is open source, it still couldn't be your homework*
- *The simplest guideline: do homework by yourself*

Please refer to this file for more detail about plagiarism.[10].

[10] https://speech.ee.ntu.edu.tw/DSP2022Autumn/Pdfs/Plagiarism.pdf

Questions?

# Contact TAs

If you have any question or need help,

- send email to *ntudsp2022ta@gmail.com*
- and use "**`[HW1]`**" as the subject line prefix

Or come to my online TA hour, and don't forget to inform me by email, thanks!

| | | |
|---|---|---|
| 林子權 | Mon. | 10:00 - 12:00 |

Office hours