11

DEEP LEARNING FOR SPEECH RECOGNITION

11.1 Fundamental model of speech

11.1.1 Speech signal characterization

11.1.1.1 *Speech signal representation*

A speech signal is a sequence of numbers which denote the amplitude of the speech spoken by the speaker. When an audio signal is recorded, the signal values represent variations in air pressure around a central average value.

As we can see Figure 11.1.1, audio signal is normally non-stationary, which prevents us from directly performing Fourier transform on the whole sequence. Instead, we perform short-term Fourier transform on a short sequence of 20 to 30 milliseconds, which usually consists of about 320 to 480 observations (assuming 16kHz). We assume the short sequence signal is a stationary signal, which is justified by the fact that our vocal track physically remains still. Each sequence is called a **frame**, and the process of dividing a long sequence (1 s to 10 s) into multiple short sequences is called **framing**.

When you use the FFT to measure the frequency component of a signal, you are basing the analysis on a finite set of data. The actual FFT transform assumes that it is a finite data set, a continuous spectrum that is one period of a periodic signal. For the FFT, both the time domain and the frequency domain are circular topologies, so the two endpoints of the time waveform are interpreted as though they were connected together. When the measured signal is periodic and an integer number of periods fill the acquisition time interval, the FFT turns out fine as it matches this assumption. However, many times, the measured signal isn't an integer number of periods. Therefore, the finiteness of the measured signal may result in a truncated waveform with different characteristics from the original continuous-time signal, and the finiteness can introduce sharp transition changes into the measured signal. The sharp transitions are discontinuities.

When the number of periods in the acquisition is not an integer, the endpoints are discontinuous. These artificial discontinuities show up in the FFT as high-frequency components not present in the original signal. These frequencies can be much higher than the Nyquist frequency and are aliased between 0 and half of your sampling rate. The spectrum you get by using a FFT, therefore, is not the actual spectrum of the original signal, but a smeared version. It appears as if energy at one frequency leaks into other frequencies. This phenomenon is known as spectral leakage, which causes the fine spectral lines to spread into wider signals.

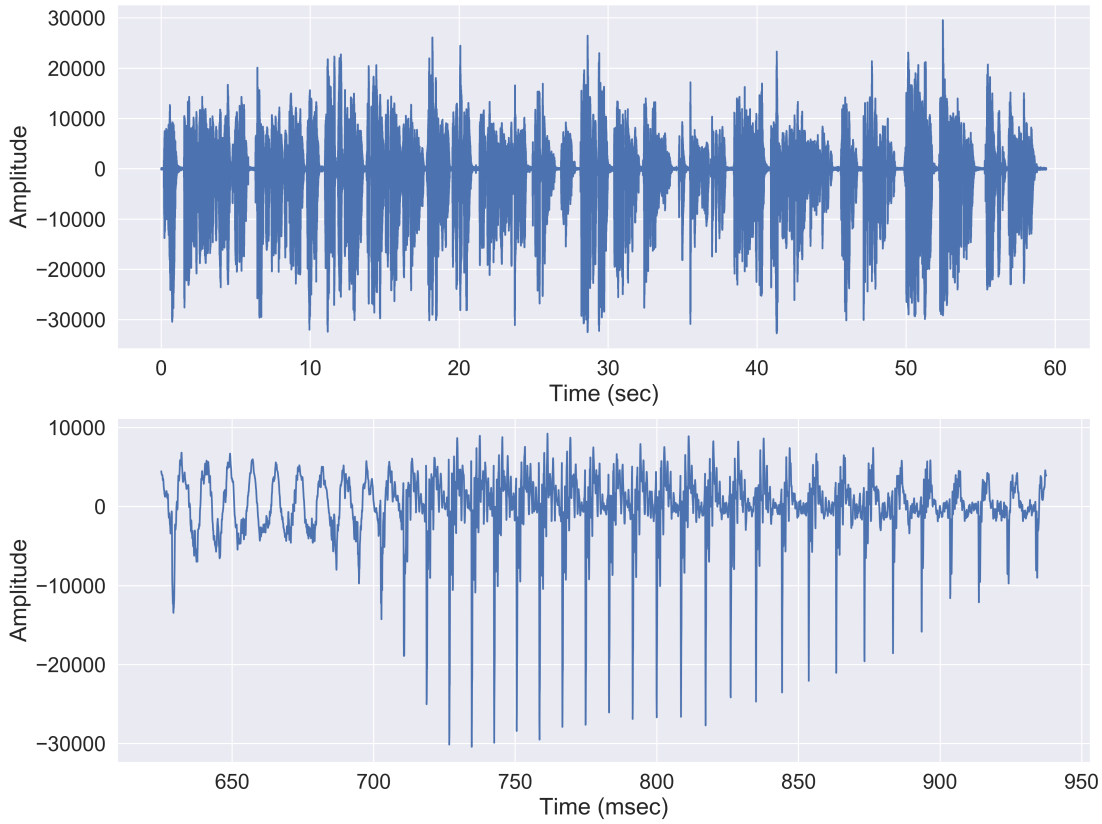


Figure 11.1.1: (a) 100 second audio signal (sampling rate 16kHz), with a sampling rate 16kHz and total samples of 950353. (b) Magnified view.

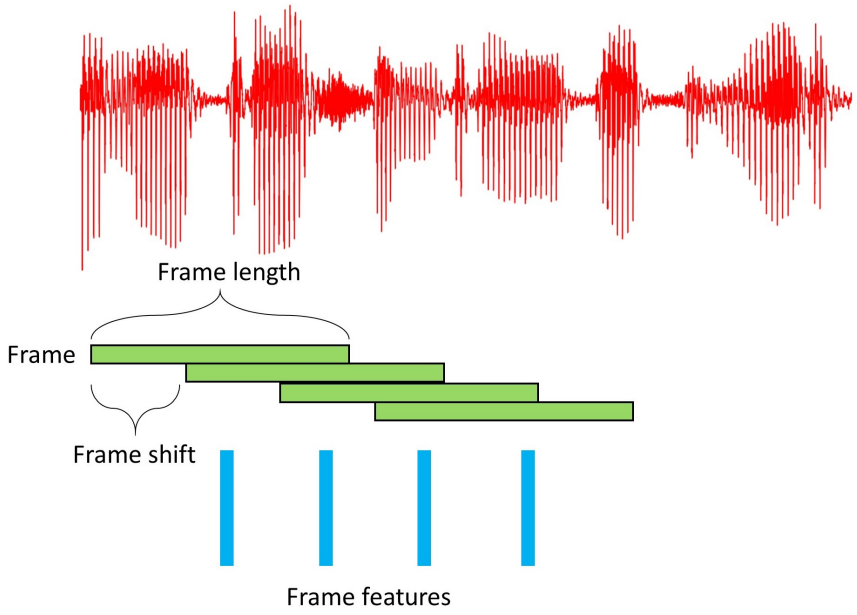


Figure 11.1.2: Speech audio signals are decomposed into overlapping frames, which are further processed into features for machine learning applications.

The purpose the window function is to minimize the end-point discontinuities in the frame. Commonly used window functions include [1, p. 89]

- Hamming window function, which has the form

$$h[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right)$$

  where $n = 0, ..., N-1$, and N is the window length.
- Hanning window function, which has the form

$$h[n] = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right)$$

  where $n = 0, ..., N-1$, and N is the window length.
- Triangular window function, which has the form

$$h[n] = \begin{cases} \frac{n}{N/2}, & \text{for } n = 0, 1, 2, \ldots, N/2 \\ 2 - \frac{n}{N/2}, & \text{for } n = N/2+1, N/2+2, \ldots, N-1 \end{cases}$$

  where $n = 0, ..., N-1$, and N is the window length.

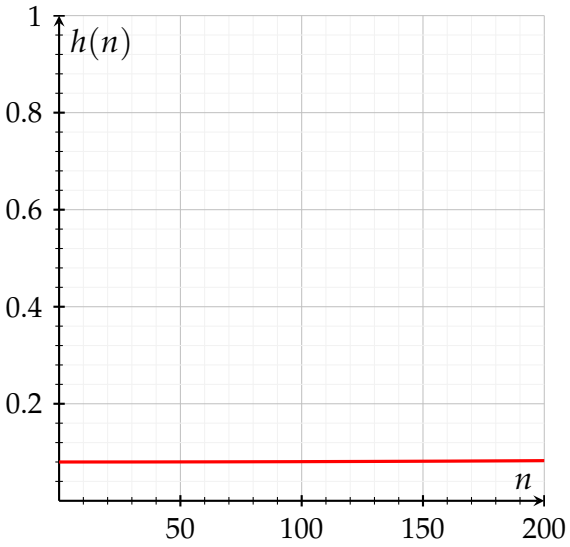


Figure 11.1.3: Hamming window function.

Due to windowing, we are actually losing the samples at the beginning and the end of the frame; this too will lead to an incorrect frequency representation. To compensate for this loss, we can take overlapping frames rather than disjoint frames. For example, the frame length is usually 25ms and the frame shift step size is usually 10ms. Therefore there is an overlap between frames.

We call our time domain signal $S(n)$. Once it is framed we have $s_i(n)$ where n ranges over $1 - 400$ (if our frames are 400 samples) and i ranges over the number of frames. $S_i(k)_{-\text{where the } i}$ denotes the frame number When we calculate the complex DFT, we get corresponding to the time-domain frame. $P_i(k)$ is then the **power spectrum** of frame i

To take the Discrete Fourier Transform of the frame, perform the following:

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-j2\pi kn/N} \quad 1 \leq k \leq K$$

where $h(n)$ is an N sample long analysis window (e.g. hamming window), and K is the length of the DFT. The **periodogram-based power spectral** estimate for the speech frame $s_i(n)$ is given by: $P_i(k) = \frac{1}{N}\left|S_i(k)\right|^2$.

11.1.1.2 *Filter bank features*

Filter bank energies is one feature widely used in automatic speech recognition. Filter bank energies is obtained by applying triangular filters, typically 40 filterson a Mel-scale to the power spectrum to extract frequency bands. The Mel scale connects perceived

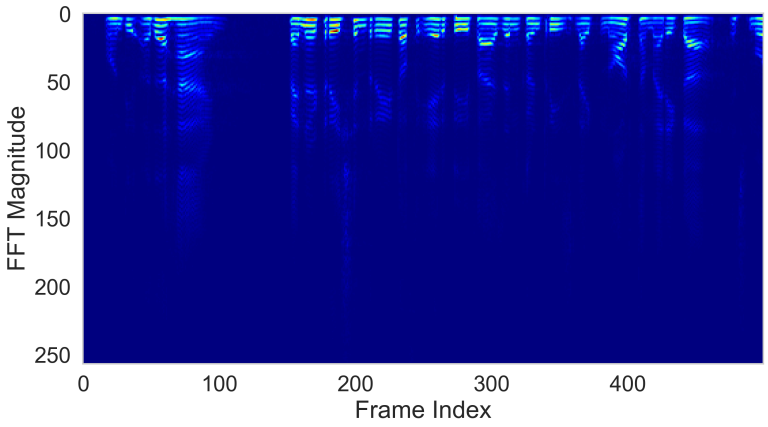


Figure 11.1.4: FFT magnitudes corresponding to 256 frequencies for each frame (horizontal axis, 500 frames in total) in the signal.

frequency, or pitch, of a signal to its actual measured physical frequency. Humans are much better at distinguishing small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear. The formula for converting from frequency to Mel scale is: $M(f) = 1125 \ln(1 + f/700)$ To go from Mels back to frequency: $M^{-1}(m) = 700(\exp(m/1125) - 1)$.

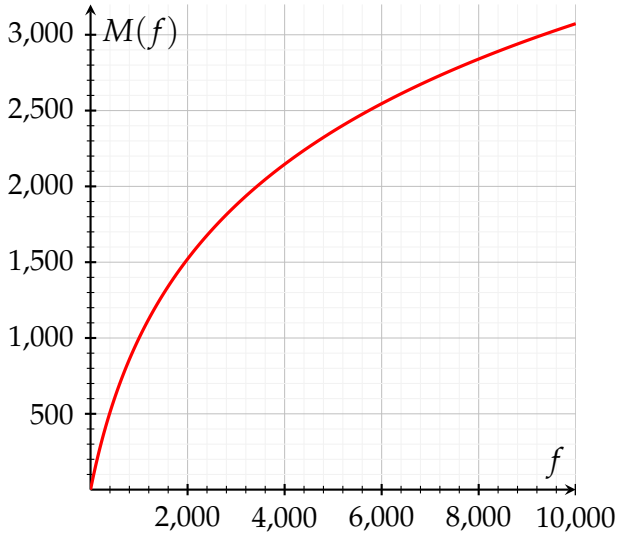
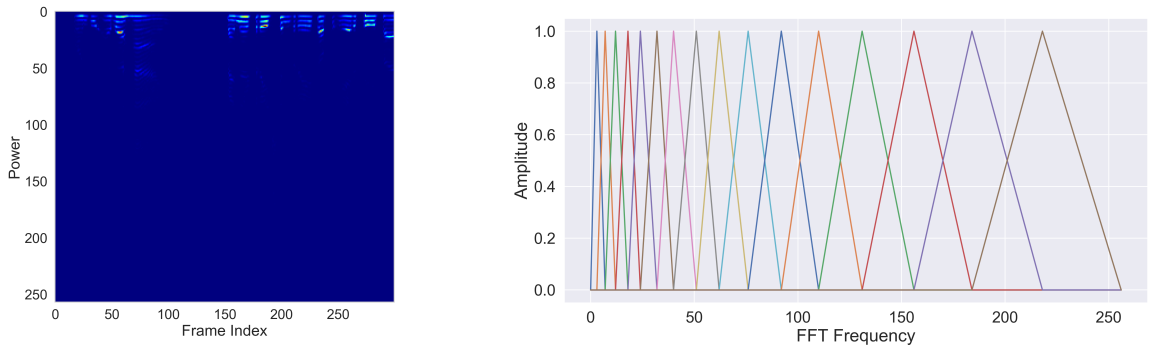


Figure 11.1.5: Mel scale transformation for physically measure frequency.



(a) Power spectrum. **(b)** Filter banks.

Figure 11.1.6: Computation of filter bank features.

11.1.1.3 *Mel Frequency Cepstral Coefficient (MFCC)*

It turns out that filter bank coefficients computed in the previous step are highly correlated, which could be problematic in some machine learning algorithms. Therefore, we can apply Discrete Cosine Transform (DCT) to decorrelate the filter bank coefficients and yield a compressed representation of the filter banks. Typically, for Automatic Speech Recognition (ASR), the resulting cepstral coefficients 2-13 are retained and the rest are discarded; The reasons for discarding the other coefficients is that they represent fast changes in the filter bank coefficients and these fine details don't contribute to Automatic Speech Recognition (ASR).

Some implementations of MFCC apply sinusoidal liftering as the final step in calculations of MFCC. It is claimed that speech recognition can be significantly improved. For instance, if MFC C_i is a cepstral coefficient, and w is a lifter, then

$$\widehat{\text{MFCC}}_i = w_i \text{MFCC}_i$$

is a lifted cepstral coefficient, where w_i for sinusoidal liftering is defined as:

$$w_i = 1 + \frac{D}{2} \sin\left(\frac{\pi i}{D}\right)$$

When I look at the equation, I understand the sinusoidal function has a shape such that its maximum is in the middle and approaches to zero at edges. Therefore, cepstral vector's first and the last coefficients are is reduced to zero while the middle one is intact.

Further, we also include first-order and second-order differences of MFCC as additional dynamic features. In total, the feature vector in a speech recognition system is usually a combination given by $X_i = (C_i, \Delta C_i, \Delta\Delta C_i)$.

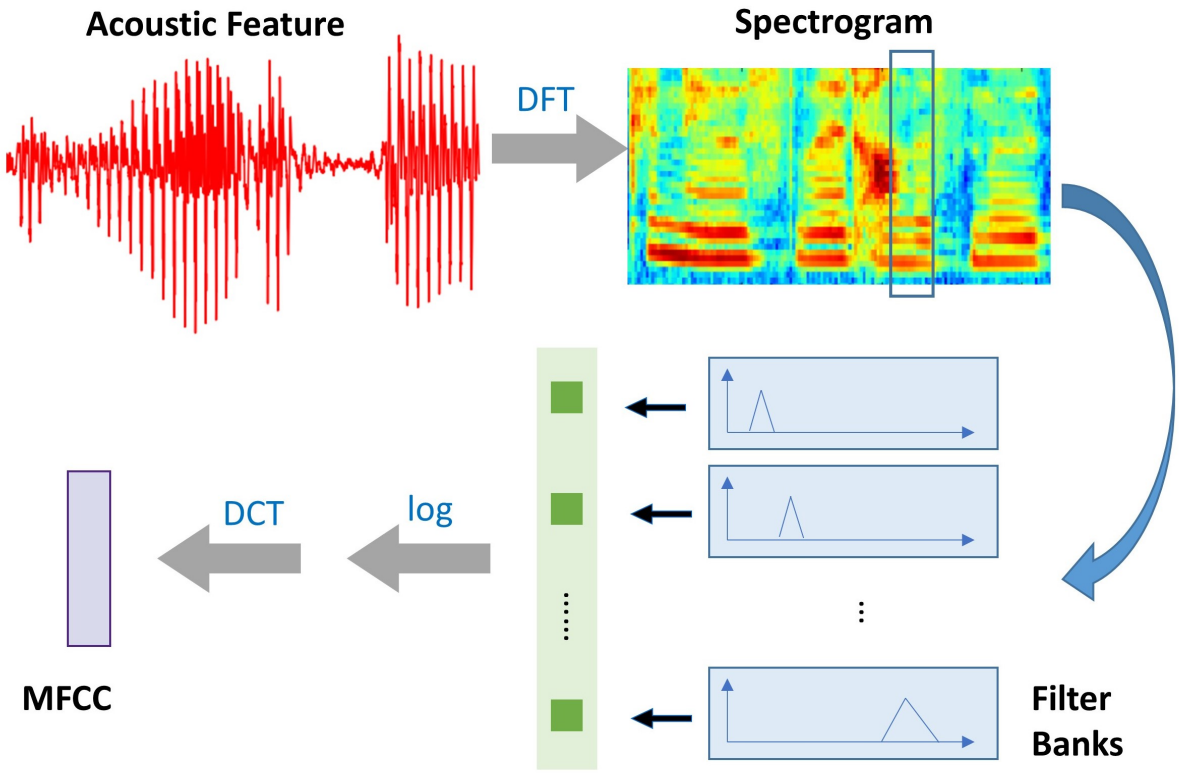


Figure 11.1.7: The computational flow for MFCC features.

11.1.2 A speech recognition systems

A traditional speech recognition system consisting of several components [Figure 11.1.8].

Suppose the input is an audio sequence $O = o_1, ..., o_T$ and output is a text sequence $W = \{w_1, ..., w_m\}$. The goal is to seek a probabilistic model (with proper model parameters) that maximizes the likelihood of N sample pairs $\{(O_i, W_i)\}$, given by

$$\prod_{n=1}^{N} P(W_n|O_n)$$

. Based on Bayes rule, the probability for a single sample pair (O, W), we have inference probability

$$P(W|O) = \frac{P(O|W)P(W)}{P(O)}.$$

In the decoding phase, we seek an optimal text sequence W^* such that

$$W^* = \arg\max_W P(O|W)P(W)$$

where $P(O|W)$ is given by an acoustic model and $P(W)$ is given by a language model.

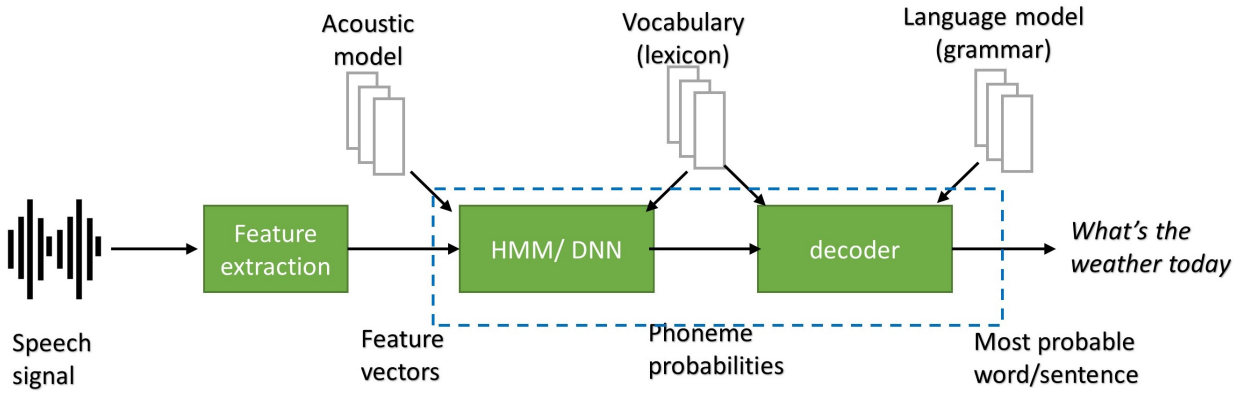


Figure 11.1.8: A traditional end-to-end speech recognition system

Directly modeling $P(O|W)$ can be intractable. Instead, we can exploit the fact that the pronunciation of each word's pronunciation is comprised of different phonemes. A phoneme is a unit of sound. Let the phoneme sequence of $Q = \{q_1, ..., q_T\}$, the probabilistic model is rewritten as

$$P(O|W) = \sum_Q P(O, Q|W) = \sum_Q P(O|Q, W)P(Q|W) \approx \sum_Q P(O|Q)P(Q|W),$$

where $P(O|Q)$ is the acoustic-phoneme model, and $P(Q|W)$ is the phoneme model, which specifies the decomposition of a word's pronunciation into a combination of different phonemes.

Usually, a phoneme model is usually given by a dictionary mapping from a word to phonemes. For example,

$$\text{cat} \rightarrow \text{K AE T}$$

$$\text{good} \rightarrow \text{G UH D}$$

$$\text{man} \rightarrow \text{M AE N}$$

$$\text{one} \rightarrow \text{W AH N}$$

$$\text{punch} \rightarrow \text{P AH N CH}$$

where $K, AE, T, G, UH, N, W, P, AH, CH$ are symbols of phonemes.

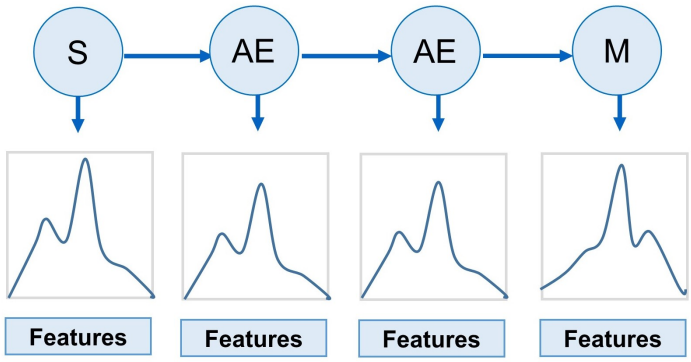


Figure 11.1.9: Scheme of GMM-HMM model. The hidden states are the phonemes and the observations are GMM features.

11.1.3 GMM-HMM

11.1.3.1 *The model*

In GMM-HMM, we have following elements

$\mathcal{S}$ state space consists of N finite hidden states.
$\mathcal{O}$ observation space consists of V finite different observations.
S_t, s_t state at time t; S_t denotes random variable; s_t denotes one realization.
O_t, o_t observation at time t; O_t denotes random variable; o_t denotes one realization. Usually $O_t = \mathbb{R}^D$.
$A = \{a_{ij}\}$ transition probability matrix A, with entry a_{ij} being the transition probability from state i to state j.
$B = \{b_i(o_t)\}$ emission probability matrix B, with entry $b_i(o_t)$ characterizing the probability of observing o_t at state i.
π initial probability distribution over state space $\mathcal{S}$.

In the GMM-HMM, we assume $b_i(o_t)$ follows a multivariate Gaussian mixture distribution [also see subsection 7.4.5]

$$b_i(o_t) = \sum_{m=1}^{M} \frac{c_{i,m}}{(2\pi)^{D/2} \left|\Sigma_{i,m}\right|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{o}_t - \mu_{i,m}\right)^\top \Sigma_{i,m}^{-1}\left(o_t - \mu_{i,m}\right)\right],$$

where $o_t \in \mathbb{R}^D$, M is the number of mixtures, $\mu_{i,m} \in \mathbb{R}^D$ is Gaussian mean vector, $\Sigma_{i,m} \in \mathbb{R}^{D\times D}$ is Gaussian covariance matrix, $w_{i,m} \geq 0, \sum_{m=1}^{M} w_{i,j} = 1$ are mixture weights.

From a data generation perspective, observation o_t is connected to hidden state i (assuming only 1 mixture) via

$$o_t = \mu_i + \Sigma_i \xi_t$$

where $\xi_t \sim MN(0, I_D)$.

A common way to reduce model parameter and improve model robustness is to share parameters among different states. For example, we can set

$$\mu_1 = \mu_{1,m} = \mu_{2,m} = \cdots = \mu_{T,m}, \Sigma_{1,m} = \Sigma_{2,m} = \cdots = \Sigma_{T,m}.$$

In modeling speech signal using GMM-HMM, we need to solve the following three fundamental problems.

- **Compute observation likelihood**: Given a GMM-HMM with model parameter θ and an observation sequence $O = \{o_1, ..., o_t\}$, compute the likelihood $P(O|\theta)$.
- **Decoder hidden state**: Given a GMM-HMM with model parameter θ and an observation sequence $O = \{o_1, ..., o_t\}$, determine the most probable hidden state sequence

$$S^* = \{s_1, ..., s_T\} = \arg\max_S P(s_1, ..., s_T, o_1, ..., o_T|\theta).$$

- **Estimate model parameter**: Given an observation sequence $O = \{o_1, ..., o_t\}$, estimate the model parameter θ.

11.1.3.2 *Compute observation likelihood via forward algorithm*

For a GMM-HMM with N hidden states and an observation sequence of T observations, the computation of $P(O)$ can be decomposed into the following steps: First, compute the joint distribution $P(O, S)$ of an observation O and a possible hidden state sequence S

$$P(O, S) = P(O|S) \times P(S) = \prod_{i=1}^{T} P\left(o_i|s_i\right) \times \prod_{i=1}^{T} P\left(q_i|q_{i-1}\right);$$

Second, compute $P(O)$ from $P(O, S)$ by marginalizing out S; that is,

$$P(O) = \sum_S P(O, S) = \sum_S P(O|S)P(S).$$

The issue of the above brute-force approach is that the number of Q can be huge. More precisely, there are totally N^T possible hidden sequences, and the computation quickly becomes intractable for large N and T.

Another efficient approach is to take average of the recursive structure in the computation and use dynamic programming. Let $\alpha_t(j)$ represents the probability of being in state j after seeing the first t observations, which can be written as:

$$\alpha_t(j) = P\left(o_1, o_2 \ldots o_t, s_t = j|\lambda\right).$$

Note that $\alpha_t(j)$ is connected to α_{t-1} via

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t),$$

where is interpreted as the probability of paths of getting to j at time t is sum of probabilities getting to i at time $t-1$, then transitioning to state j and emitting observation o_t. Finally, we can compute $P(O) = \sum_{i=1}^{N} \alpha_T(T)$. Note that we can initialize

$$\alpha_1(j) = P(o_1, s_1 = j) = P(o_1|s_1 = j)P(s_1 = j) = b_j(o_1)\pi_j$$

.

In the following, we summarize the likelihood computation procedure.

Methodology 11.1.1 (compute observation likelihood). *Given an observation sequence $O = \{o_1, ..., o_T\}$ and a GMM-HMM characterized by model parameter θ. The likelihood $P(O|\theta)$ can be computed via following steps:*

- *Initialization:*
$$\alpha_1(j) = \pi_j b_j(o_1) \, 1 \leq j \leq N$$
- *Propagation:*
$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$
- *Final computation:*
$$P(O|\theta) = \sum_{i=1}^{N} \alpha_T(i).$$

11.1.3.3 *Decoding via backward (Viterbi) algorithm*

To decode an observation sequence $O = \{o_1, ..., o_t\}$, we need to determine the most probable hidden state sequence $S^* = \{s_1, ..., s_T\}$ that maximizes $P(s_1, ..., s_T, o_1, ..., o_T|\theta)$, that is,

$$S^* = \{s_1, ..., s_T\} = \arg\max_S P(s_1, ..., s_T, o_1, ..., o_T|\theta).$$

The brute force approach that exhaustively search the desired sequence among all possible sequences is intractable since there are N^T candidates in total.

Alternatively, we can define $v_t(j)$, as the probability that the HMM is in state j after seeing the first t observations and passing through the **most probable state sequence** $s_1, \ldots, s_{t-1}$, that is,

$$v_t(j) = \max_{s_1,\ldots,s_{t-1}} P\left(s_1 \ldots s_{t-1}, o_1, o_2 \ldots o_t, s_t = j|\theta\right)$$

Note that we represent the most probable path by taking the maximum over all possible previous state sequences $\max_{s_1,\ldots,s_{t-1}}$.

Surprisingly, via dynamic programming principle, for a given state q_j at time t, the value $v_t(j)$ is computed as

$$v_t(j) = \max_i v_{t-1}(i) a_{ij} b_j\left(o_t\right).$$

In the following, we summarize the decoding procedure.

Methodology 11.1.2 (decoding via Viterbi algorithm). *Given an observation sequence $O = \{o_1, ..., o_T\}$ and a GMM-HMM characterized by model parameter θ. The model probable sequence can be obtained via following steps.*

- *Initialization:*

$$\begin{aligned} v_1(j) &= \pi_j b_j\left(o_1\right) \quad 1 \leq j \leq N \\ bt_1(j) &= 0 \qquad\qquad\quad 1 \leq j \leq N \end{aligned}.$$

- *Propagation*

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j\left(o_t\right); \quad 1 \leq j \leq N, 1 < t \leq T$$

- *Compute the most probable sequence backward:*
  - $s_T = \arg\max_i v_T(i)$
  - *Starting from $t = T - 1$ to $t = 1$,*

$$s_t = \arg\max_i v_t(i) a_{i,s_{t+1}} b_{s_{t+1}}(o_{t+1})$$

11.1.3.4 *Estimate model parameter*

11.2 Classic neural network models

11.2.1 Listen, Attend, and Spell

Listen, Attend, and Spell (LAS) is a model architecture [Figure 11.2.1] based on seq2seq framework with attention mechanism[2]. The listener is an encoder consisting of

three pyramidal Bidirectional LSTM layers. The pyramidal structure is required to reduce the length of encoder output h, which is shown to be critical for input speech signals that usually comprise of hundreds and thousands of frames. Without pyramidal structure, the encoder output would have the same large length of the input, which often leads to costly computation and the difficulty of finding relevant information in the attention module.

As a comparison, in a typical deep bidirectional LSTM architecture, the output at the ith time step, from the jth layer is given by

$$h_i^{(j)} = \text{BLSTM}\left(h_{i-1}^{(j)}, h_i^{(j-1)}\right);$$

while in the pyramidal bidirectional LSTM model, we first concatenate the outputs at consecutive steps of each layer and then feeding then to the next layer, we have

$$h_i^{(j)} = \text{pBLSTM}\left(h_{i-1}^{(j)}, \text{Concat}[h_{2i}^{(j-1)}, h_{2i+1}^{(j-1)}]\right).$$

By stacking three pyramidal layers, we actually reduce the output length by a factor of $2^3 = 8$.

In the decoder module, at every output step, an input token y^{i-1} is first projected to a state s_{i-1}, which is later combined with an attention-based context vector c_{i-1} and are fed into an RNN together. The final output is a probability distribution over the next character $y_i$conditioned on all the characters seen previously. Note that the decoder state s_i is a function of the previous state s_{i-1}, the previous input y_{i-1} and previous context c_{i-1}. Specifically,

$$\begin{aligned} c_i &= \text{Attention}\left(s_i, \mathbf{h}\right) \\ s_i &= \text{RNN}\left(s_{i-1}, y_{i-1}, c_{i-1}\right) \\ P\left(y_i | x_1, ..., x_T, y_1, ..., y_{i-1}\right) &= \text{CharacterDistribution}\left(s_i, c_i\right) \end{aligned}$$

where CharacterDistribution is a feed-forward network with Softmax outputs over characters, and RNN is a 2-layer LSTM.

At each time step, i, the Attention module generates a context vector, c_i, which extract most relevant information from $h = (h_1, ..., h_U)$ needed to generate the next character. We have following mathematical summary.

$$\begin{aligned} e_{i,u} &= \left\langle \phi\left(s_i\right), \psi\left(h_u\right) \right\rangle \\ wi, u &= \frac{\exp\left(e_{i,u}\right)}{\sum_u \exp\left(e_{i,u}\right)} \\ c_i &= \sum_{u=1}^{U} wi, uh_u \end{aligned}$$

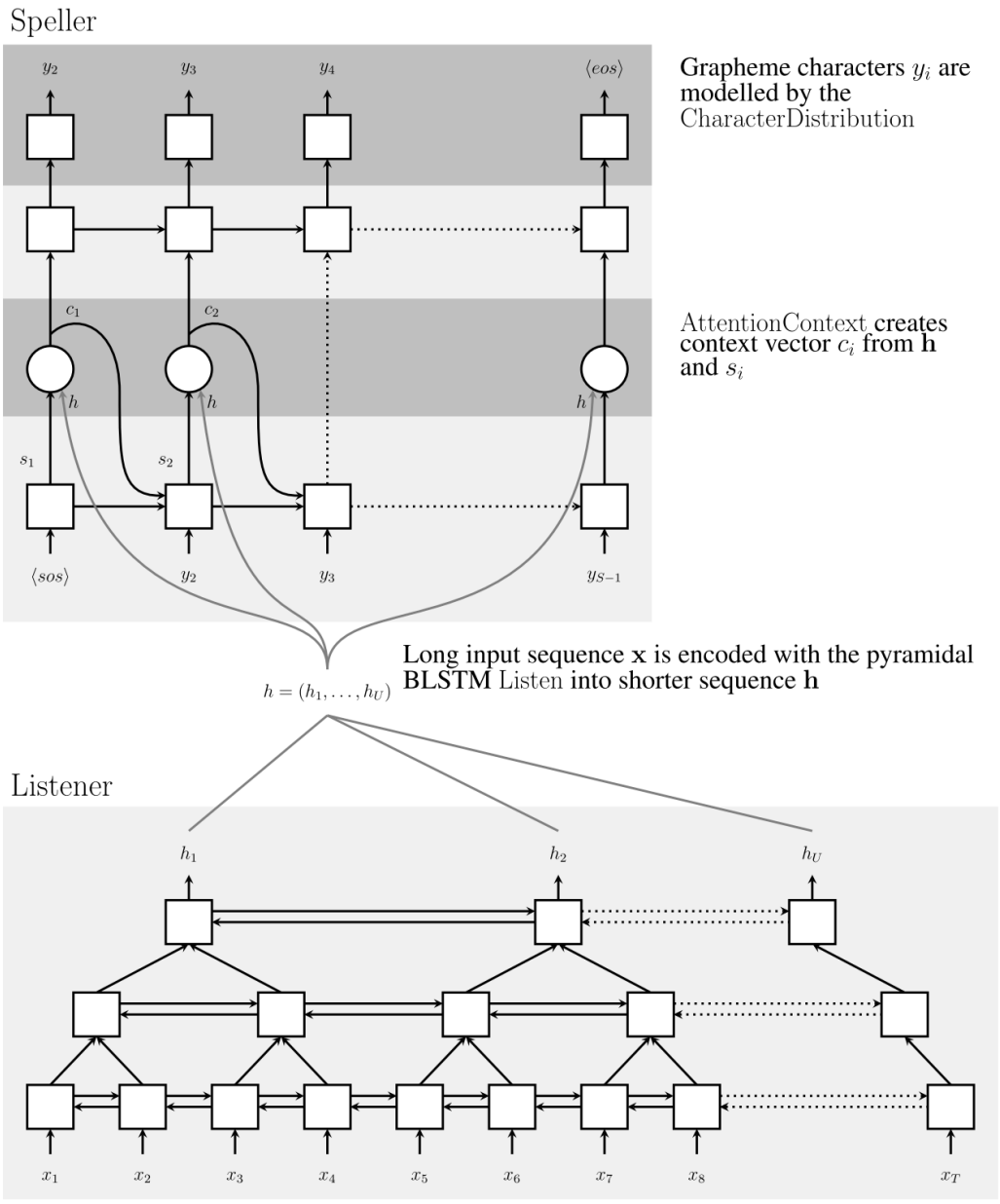


Figure 11.2.1: Listen, Attend and Spell (LAS) architecture. The listener is a pyramidal bidirectional LSTM encoding input sequence $(x_1, ..., x_T)$ into high level features $h = (h_1, ..., h_U)$. The speller is an attention-based decoder generating the y characters from h. Image from [2].

where ϕ and ψ are nonlinear transformation represented feed-forward neural networks. After convergence, the attention weight wi vector distribution is typically very sharp and localized, and focused on only a few frames of h [Figure 11.2.2]

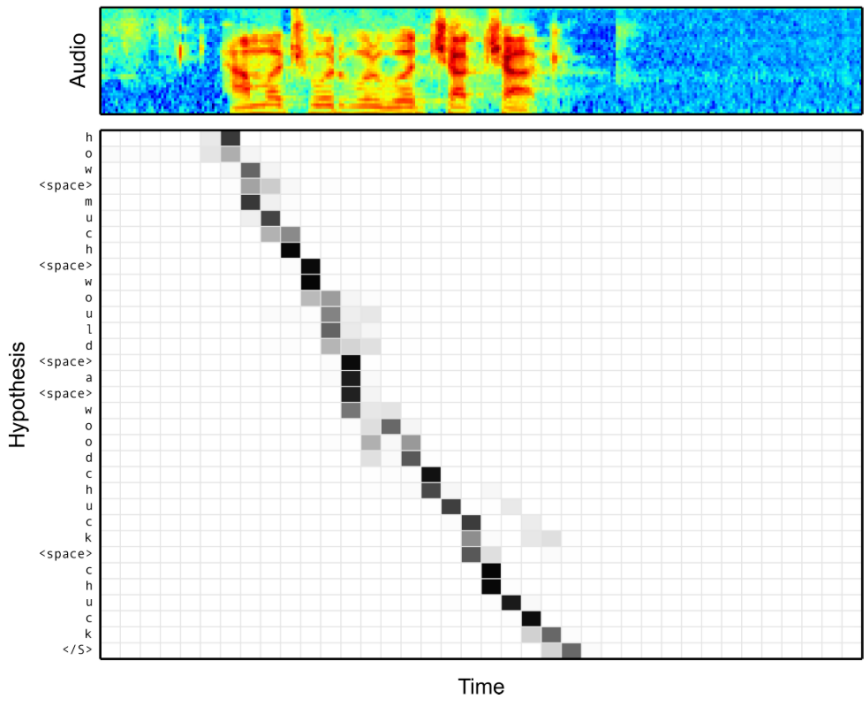


Figure 11.2.2: Alignment, as indicated by attention, between character outputs and audio signal produced by the Listen, Attend and Spell (LAS) model for the utterance "how much would a woodchuck chuck". Image from [2].

The whole model can be trained in an end-to-end fashion. The goal is to maximizes the log probability of target sequence $y_1, ..., y_N$ with teacher forcing, that is

$$\max_{\theta} \sum_{i} \log P\left(y_i | x_1, ..., x_T, y_1, ..., y_{i-1}; \theta\right)$$

where $y_1, ..., y_{i-1}$ are previous ground truth characters.

Note that during inference stage, the ground truth is missing. To enhance robustness when the previous prediction is bad, we sometimes feed previous predicted character as inputs in the next step predictions, that is

$$\begin{aligned} \tilde{y}_{i-1} \sim & \text{ CharacterDistribution } \left(s_{i-1}, c_{i-1}\right) \\ \max_{\theta} & \sum_{i} \log P\left(y_i | x_1, ..., x_T, y_1, ..., \tilde{y}_{i-1}; \theta\right). \end{aligned}$$

11.2.2 Connectionist Temporal classification

[3]

11.2.3 RNN Transducer

[4]

11.3 Speaker recognition: classic statistical approach

11.3.1 Introduction

11.3.1.1 *Background*

Automatic speaker recognition aims to recognize a speaker's identity or verify a person's claimed identity from his voice. There are three major tasks relevant to speaker recognition:

- Speaker identification, which decides who is speaking among a group of people.
- Speaker verification, which decides if a speaker is who he claims to be.
- Speaker diarization, which partitions an audio signal with multiple speakers into segments associated with each individual speaker.

Automatic speaker recognition has found many important applications [Figure 11.3.1] such as authentication, access control, personalization, and law enforcement.

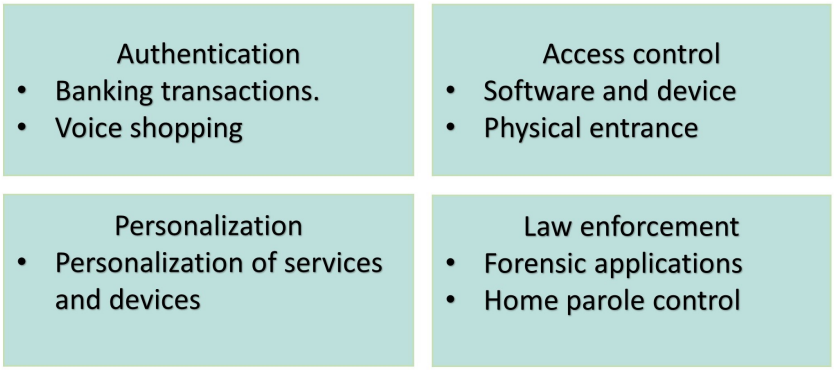


Figure 11.3.1: Speaker recognition system applications.

11.3.1.2 *Speaker identification*

The goal of speaker identification is to decide the identity of an unknown speaker based on his voice sample. Usually, the voice sample is transformed to feature vector first, and then the feature vector is compared against features from a candidate speaker set. We distinguish **open-set identification** and **closed-set identification**. In open-set identification, the unknown speaker might not lie in the candidate speaker set. On the other hand, in closed-set identification, the unknown speaker is in the candidate speaker set. Clearly, the computational cost of a speaker identification increases as the candidate set increases.

Further, based on the content of the audio signal, we also distinguish **text-dependent recognition** and **text-independent recognition**. In text-dependent recognition, recognition system knows the text the unknown speaker is going to speak and assumes the speaker is cooperative. This knowledgment can improve performance. In text-independent recognition, the system does not known what the unknown speaker is going to speak.Speech recognition system can be used to determine the text spoken by the person.

Text-dependent recognition is common for security applications where speakers are asked to speak specific words and sentences. Text-independent recognition is common for applications like voice assistant device, customer services.

Commonly used metrics for a speaker recognition system includes

- **False Rejection Rate (FRR)**: The speaker false rejection rate (FRR) is the rate that a given speaker is incorrectly rejected. In other words, the prediction is false but the ground truth is true.
- **False Acceptance Rate (FAR)**: The speaker false acceptance rate (FAR) is the rate that utterances not belonging to an enrolled speaker are incorrectly accepted as belonging to the enrolled speaker.
- **Equal Error Rate (EER)**: This is the rate used to determine the threshold value for a system when its FAR and FRR are equal.

11.3.1.3 *Speaker verification*

A speaker verification system has two stages. In the first **enrollment** stage, a speaker first enrolls his voice in the system. In the second **verification** stage, a speaker makes a claim on his identity, and the system decides the authenticity of the claim.

Because the voice sample is only compared against the model of the claimed speaker, the computational cost is independent of the speaker population. Similar to speaker identification, the verification process can also be categorized into text-dependent verification and text-independent verification.

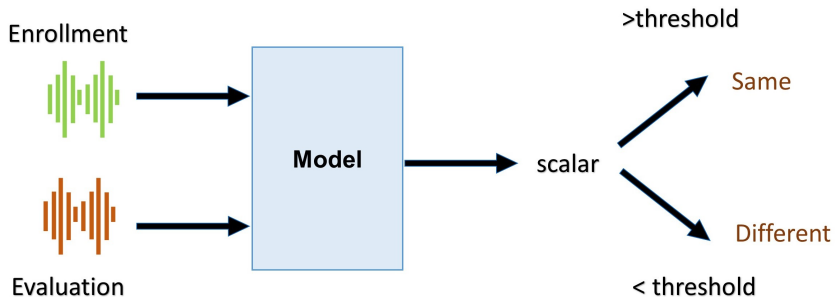


Figure 11.3.2: A generic computational flow for speaker verification

11.3.1.4 *Speaker diarization*

Another popular application of speaker recognition technology is speaker diarization, whose goal is to partition an audio stream with multiple people into homogeneous segments associated with each individual, is an important part of speech recognition systems. By solving the problem of "who spoke when", speaker diarization has applications in many important scenarios, such as understanding medical conversations, video captioning and more. In general, there are two steps in a speaker diarization task:

- Segmentation.
- Clustering (the number of speakers can be known).

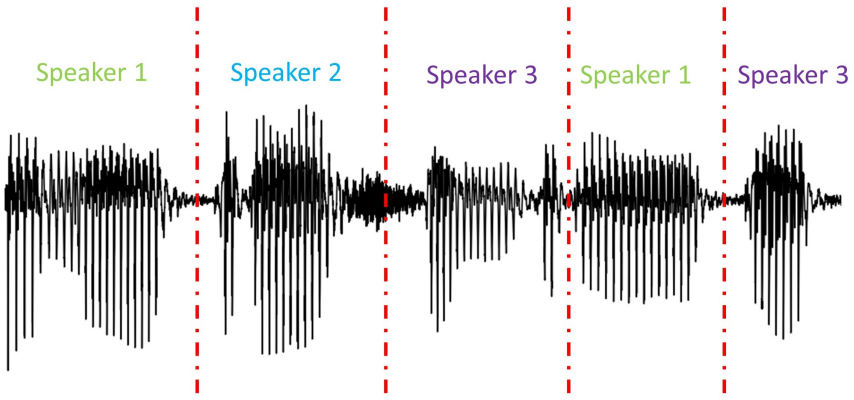


Figure 11.3.3: Speaker diarization schematics.

11.3.2 Speaker recognition via simple GMM

The GMM specifies a data generation model for a D-dimensional feature vector x. It specifies the density as the sum of Gaussian mixtures given by:

$$P(x|\lambda) = \sum_{k=1}^{M} w_k \times g\left(x|\mu_k, \Sigma_k\right)$$

where x is a D-dimensional feature vector $w_k, k = 1, 2, \ldots, M$ are the mixture weights that sum to 1, $\mu_k, \Sigma_k k = 1, 2, \ldots, M$ are the mean and the covariance of each Gaussian. Explicitly, $g\left(x|\mu_k, \Sigma_k\right)$ are the Gaussian densities given by

$$g\left(x|\mu_k, \Sigma_k\right) = \frac{1}{(2\pi)^{\frac{D}{2}} \left|\Sigma_k\right|^{\frac{1}{2}}} \exp^{-\frac{1}{2}\left(x-\mu_k\right)^T \Sigma_k^{-1}\left(x-\mu_k\right)}$$

We typically use a diagonal covariance-matrix rather than a full covariance one since it is more computationally efficient and empirically works better.

Let θ denote the model parameters $\theta = \left(w_k, \mu_k, \Sigma_k\right), k = 1, 2, 3, \ldots, M$. In the testing stage, given an audio signal of the testing speaker, we extract features from overlapping frames $X = (x_1, ..., x_T)$ and under independence assumption, compute the log-likelihood of each speaker model $i, i = 1, ..., K$, given by

$$\log P(X|\theta_i) = \frac{1}{T} \sum_{t=1}^{T} \log p(x_t|\theta_i).$$

The testing speaker is identified as speaker

$$j^* = \arg\max_i \log P(X|\theta_i).$$

Remark 11.3.1 (voice activity detection). We might now want to discard useless information in the frames we extracted features for. We do so by removing frames that do not contain speech using Voice Activity Detection (VAD).

11.3.3 Speaker recognition via GMM-UBM

The most successful early speaker recognition system is the GMM-UBM system[5][Figure 11.3.4]. There are two components in the model, one is the universal background model (UBM), which is a high-order Gaussian Mixture Model (usually 512 to 2048 mixtures with 24 dimensions) trained on a large quantity of speech samples, from a wide population. UBM learns the speaker-independent distribution of features. Another

component is individual Gaussian mixture models used to learn the feature of individual speakers. In the enrollment step, we train and adapt one GMM on the extracted features for each speaker. The speaker model is adapted from the UBM GMM using Maximum a Posteriori (MAP) Adaptation .

In MAP adaption, we simply start the expectation maximum (EM) algorithm [sub-subsection 7.4.5.1] with the parameters learned by the UBM. Normally, in practice, we only adapt the mean, and not the covariance, since updating the covariance does not improve the performance.

The mean update based on MAP is given by

$$\mu_k^{MAP} = \alpha_k \mu_k + (1 - \alpha_k)\, \mu_k^{UBM}$$

where k is iteration number, $\alpha_k = \frac{n_k}{n_k + \tau_k}$ is the mean adaptation coefficient n_k is the count for the adaptation data, τ_k is the relevance factor, between 8 and 32.

The speaker verification decision can be achieved by the likelihood ratio. Let the speech sample with speaker S be Y. The verification task can be formulated as following hypothesis testing: $H_0 : Y$ is from speaker $SH_1 : Y$ is not from speaker S.

The decision to accept H_0 or not is the Likelihood Ratio (LR):

$$LR = \frac{p\left(Y|H_0\right)}{p\left(Y|H_1\right)}$$

where H_0 would use the speaker GMM and H_1 would use the UBM. If the Likelihood ratio is greater than the threshold θ, we accept H_0, otherwise we accept H_1.

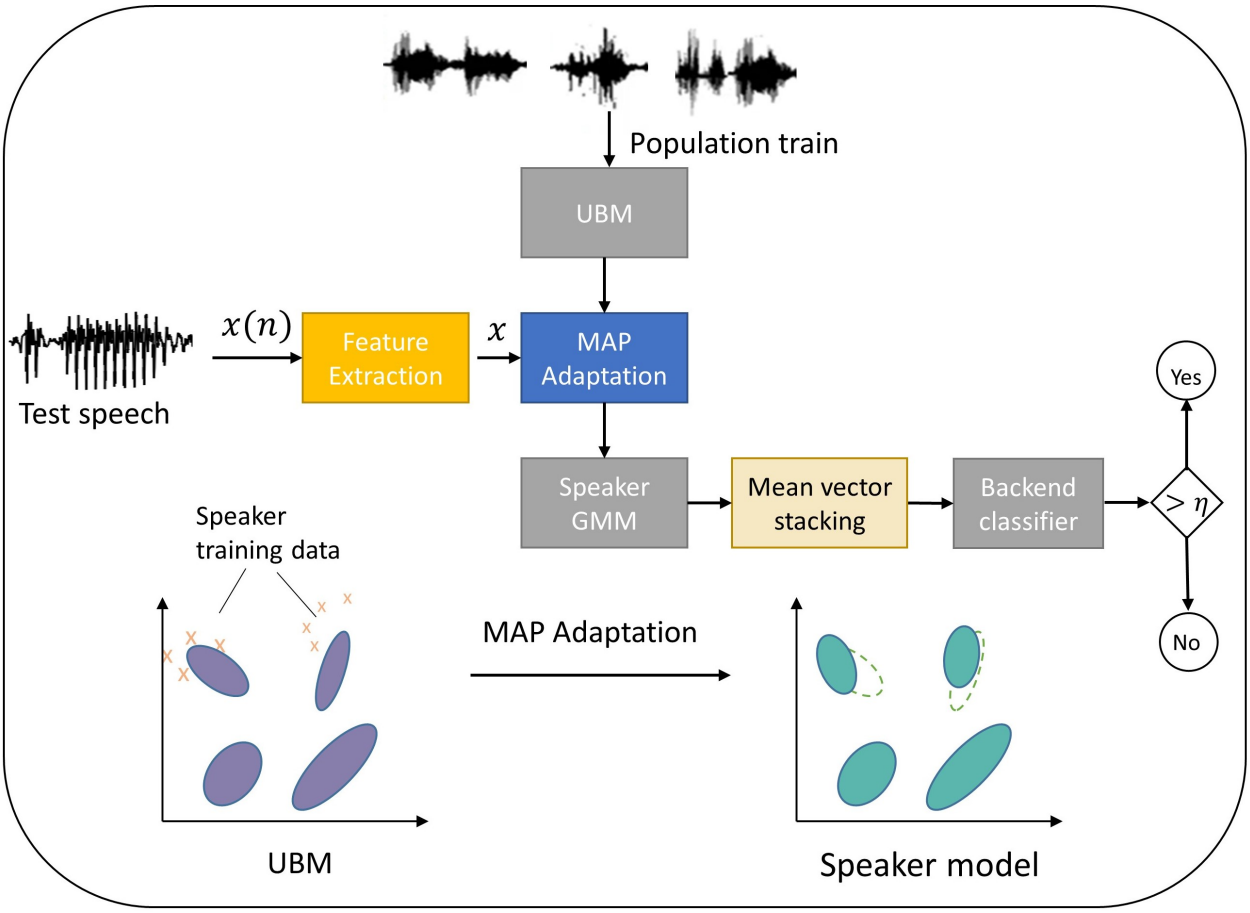


Figure 11.3.4: GMM-UBM scheme for speaker identification.

11.3.4 i-vector method

11.3.4.1 *i-vector construction*

Let s be the supervector consisting of stacked mean vectors from a GMM speaker model, m be the GMM-UBM supervector. In the i-vector method [Figure 11.3.5], we assume the factorization

$$s = m + Tw,$$

where T is the total variability matrix, and w is a set of low-dimensional total variability factors. w known as the **i-vectors.**

The w factor are hidden variable and requires factor analysis to estimate. w can be used as feature vectors and fed into backend classifiers for classification.

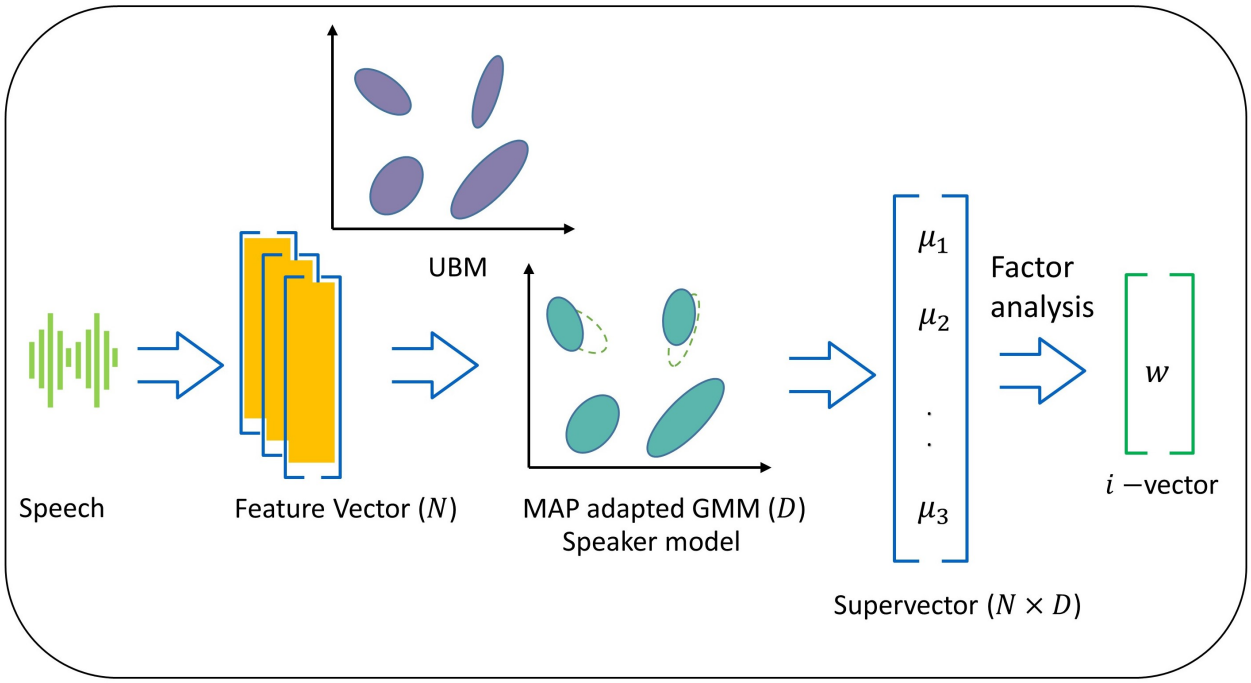


Figure 11.3.5: i-vector computational flow.

11.3.4.2 *Backend PLDA classifier*

The computed i-vector for each speaker can be fed into backend classifiers to recognize the identities of speaker. One of most successful classifier is based on probabilistic linear discriminant analysis (PLDA)[6, 7], which is also widely used in facial recognition.

The core idea of PLDA is to model the feature (e.g., MFCC) x of a speaker coming from a hierarchical generative model given by

$$
\begin{aligned}
x &= m + Au \\
u &\sim MN(\cdot \mid v, I) \\
v &\sim MN(\cdot \mid 0, \Psi).
\end{aligned}
$$

where u is the latent class variable and v is the latent variable generating u. Both u, v follow multivariate Gaussian distribution. $x \in \mathbb{R}^D$, $m \in \mathbb{R}^D$ is a constant mean vector. Ψ is a diagonal matrix.

Suppose we have already calibrated A, Ψ. In a speaker recognition task, we are given a testing example x^p and dataset $\left(x^1 \dots x^M\right)$ containing one example from each of M classes.

The goal is to determine which class x^p comes from. We perform the classification in the latent space of class variable u. For each observation x, its associated latent value can be applied via transformation

$$u = A^{-1}(x - m).$$

By performing the inference on the class variable via multivariate Gaussian Bayesian estimation [Theorem 3.4.2], we have

$$P(v \mid u) = MN\left(v \mid \frac{\Psi}{\Psi + I}u, \frac{\Psi}{\Psi + I}\right)$$

Since u^p and u^g are conditionally independent given v, we have

$$\begin{aligned} & P\left(u^p \mid u^g\right) \\ =& \int_v P\left(u^p \mid v\right) P(v \mid u^g) dv \\ =& MN\left(u^p \mid \frac{\Psi}{\Psi + I}u^g, I + \frac{\Psi}{\Psi + I}\right). \end{aligned}$$

To classify a testing example, we compute $P\left(u^p \mid u^g\right)$ for $g = 1 \ldots M$, and pick the maximum.

If we have n independent examples of a class, denoted by $u^g_{1:n}$, we can get that

$$P\left(u^p \mid u^g_{1\ldots n}\right) = MN\left(u^p \mid \frac{n\Psi}{n\Psi + I}\overline{u}^g, I + \frac{\Psi}{n\Psi + I}\right)$$

where $\overline{u}^g = \frac{1}{n}\left(u^g_1 + \cdots + u^g_n\right)$.

Remark 11.3.2 (estimation of matrix A and Ψ). Define $y = m + Av$, which can be interpreted as the mean of each class. Given feature samples from different speakers (i.e., different classes), we can compute within-class covariance matrix Φ_W and between-class covariance matrix [also see LDA in section 6.3]. We can show that

$$\begin{aligned} A\Psi A^T &= \Phi_b \\ AA^T &= \Phi_w. \end{aligned}$$

Solving A and Ψ is known as generalized eigenvalue problem and it is addressed in Theorem 1.8.13.

11.4 Deep feature extractor for speaker recognition

11.4.1 DNN d-vector method

Google proposes a DNN (deep neural network) d-vector method [8], whose core idea is to use DNN as a speaker feature extractor. The DNN is first trained using supervised learning. For each frame of the audio signal, which usually lasts 25 ms and consists of 400 data points (assuming sampling rate of 16kHz), we compute the 40 log filter bank energy. The input to the DNN is a combined feature by stacking current frame with its context (30 frames before and 10 frames after). The output layer is a softmax classification layer with N classes, where N is the number of speakers in the training set. The hidden layers contains 256 nodes (ReLU activation) and maxout pooling.

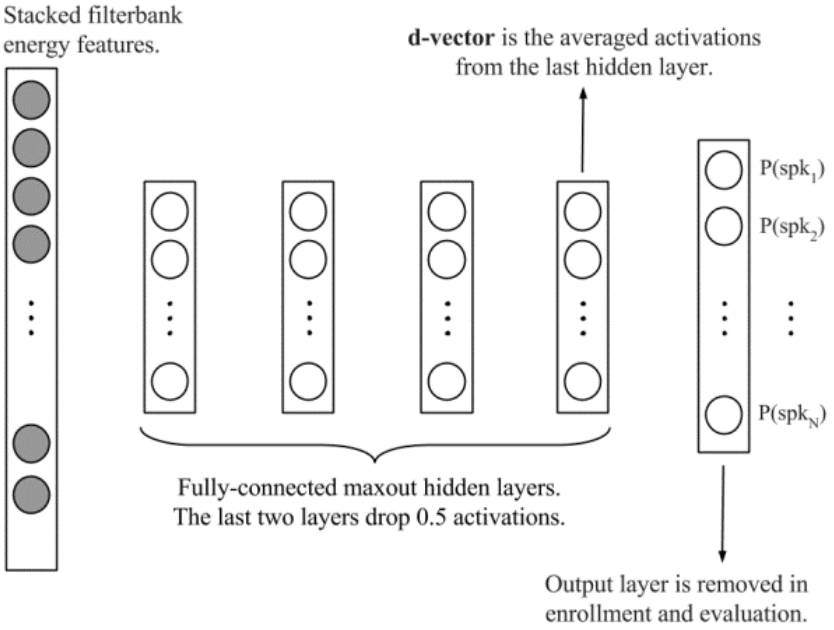


Figure 11.4.1: The background DNN model for speaker verification[8].

Once the DNN has been trained successfully, we can extract the feature vector, d-vector, in the following way: For every frame of a given utterance, we compute the activations of last hidden layer. We then average the L2 normalized activations and get the d vector.

In the enrollment process, we are given a set of utterances $X_s = \{O_{s_1}, O_{s_2}, \ldots, O_{s_n}\}$ from a speaker s, with observations at the frame level $O_{s_i} = \{o_1, o_2, \ldots, o_m\}, i = 1, ..., n$.

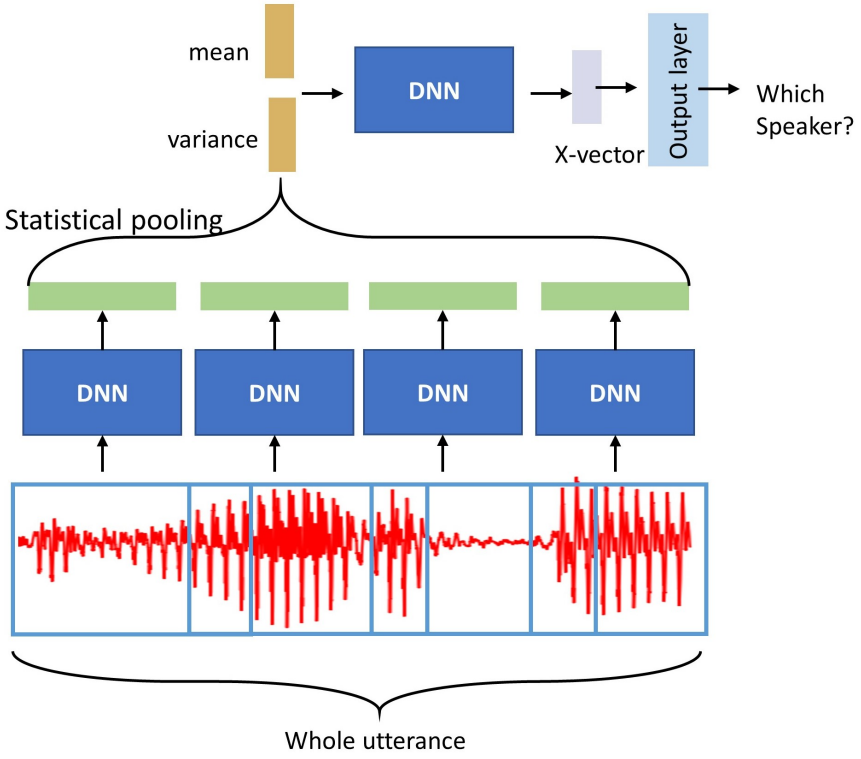


Figure 11.4.2: X-vector architecture.

First, we use the trained DNN to take every observation o_j in utterance O_{s_i}, together with its context, as the input to get the activations. We then accumulated activations from all the observations o_j in O_{s_i}, and get the d_{s_i} vector of utterance s_i. The d_s vector associated with speaker s is computed by

$$d_s = \frac{1}{|X_s|} \sum_i d_{s_i}.$$

In the evaluation phase, we use the trained DNN to extract the d vector from the test utterance. Then we compute the cosine distance between the test d -vector and the claimed speaker's d -vector. By comparing the distance value to a threshold, we either accept or reject the identity.

11.4.2 X-vector

[9]

11.4.3 Using attentions

[10] [11]

11.4.4 SincNet

DNN usually takes either manually derived features like MFCC and filter bank energies or raw speech waveform as the network input. When the raw waveform is used as input, we use CNN layers in DNN to perform feature extraction and hope that better features can be automatically derived for speech-related tasks.

The CNN layers learn filters to perform convolution with time series. Oftentimes, the learned filters are not easy to interpret. Recently, [12] invented SincNet, which can be viewed as parameterized CNN layers and can help the CNNs discover more meaningful filters at the front end. SincNet is based on parametrized sinc functions, which implement band-pass filters. In contrast to standard CNNs, that learn all elements of each filter, only low and high cutoff frequencies are directly learned from data in the SincNet. This offers a very compact and efficient way to derive a customized filter bank specifically tuned for the desired application.

In the frequency domain, the magnitude of a generic band pass filter, with low and high frequency cutoff f_L and f_H is given by $G(f; f_L, f_H) = \text{rect}(\frac{f}{2f_H}) - \text{rect}(\frac{f}{2f_L})$ where the rectangular function $\text{rect}(f/2B)$ with cutoff frequency B is defined by

$$\text{rect}\left(\frac{f}{2B}\right) = \begin{cases} 0, & \text{if } |f| > B \\ \frac{1}{2}, & \text{if } |f| = B \\ 1, & \text{if } |f| < B \end{cases}$$

The inverse Fourier transform of the rectangular function $\text{rect}(f/2B)$ is given by $2B\,\text{sinc}(2Bt)$, where a sinc function is defined by $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$.

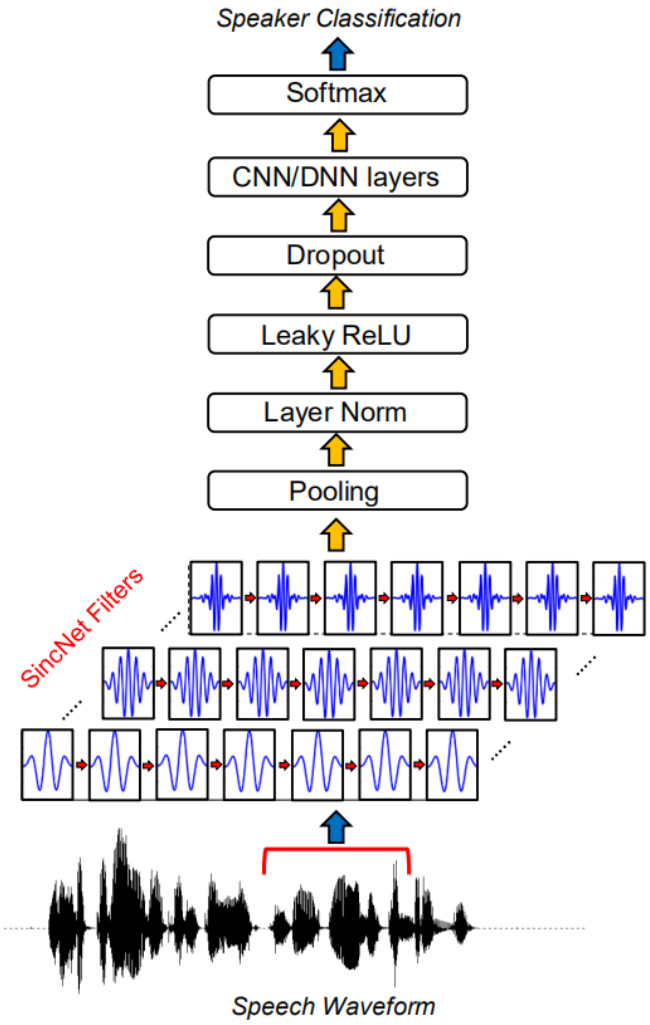


Figure 11.4.3: The SincNet architecture which takes raw speech waveform as the input and outputs classification results. The intermediate layers act as feature extractor. Image from [12].

11.5 Speaker recognition end-to-end systems

11.5.1 Google end-to-end speaker verfication

11.5.1.1 *Tuple-based end-to-end system (TE2E)*

Google first proposed a tuple-based end-toend (TE2E) model [8], which simulates the two-stage process of runtime enrollment and verification during training. In our experiments, the TE2E model combined with LSTM [14] achieved the best performance at the time. Let x denote the feature vector (e.g., the log-Mel filterbank energies) from a fixed-length audio sample. For each training step, a tuple of one evaluation utterance $x_{j\sim}$ from speaker j and M enrollment utterances $x_{km}, m = 1, ..., M$ from speaker k is fed into our LSTM network. Note that j and k might not be the same speaker.

The tuple $\left\{x_{j\sim}, (x_{k1}, \ldots, x_{kM})\right\}$, is call a positive tuple if $j = k$ and negative tuple, otherwise.

For each tuple example, we use following steps to compute the loss

- L_2 normalize the tuple and get $\left\{e_{j\sim}, (e_{k1}, \ldots, e_{kM})\right\}$.
- Compute the centroid of $(e_{k1}, \ldots, e_{kM})$ via

$$c_k = \sum_{m=1}^{M} e_{km}.$$

- Compute the cosine similarity score between $e_{j\sim}$ and c_k via

$$s = w \cos(e_{j\sim}, c_k) + b,$$

  where $w, b \in \mathbb{R}$ are learnable parameters, and $\cos(a, b) = \frac{a \cdot b}{\|a\|\|b\|}$.
- Final TE2E loss function is given by

$$L\left(e_{j\sim}, c_k\right) = \delta_{jk}\sigma(s) + (1 - \delta_{jk})(1 - \sigma(s))$$

  where δ_{ij} is the Kronecker delta function, and $\sigma(\cdot)$ is the Sigmoid function.

If $j = k$, the final loss function increases when similarity score increases and if $j \neq k$, the final loss function increases when similarity score decreases.The TE2E loss function encourages a larger value of s when $k = j$, and a smaller value of s when $k \neq j$.

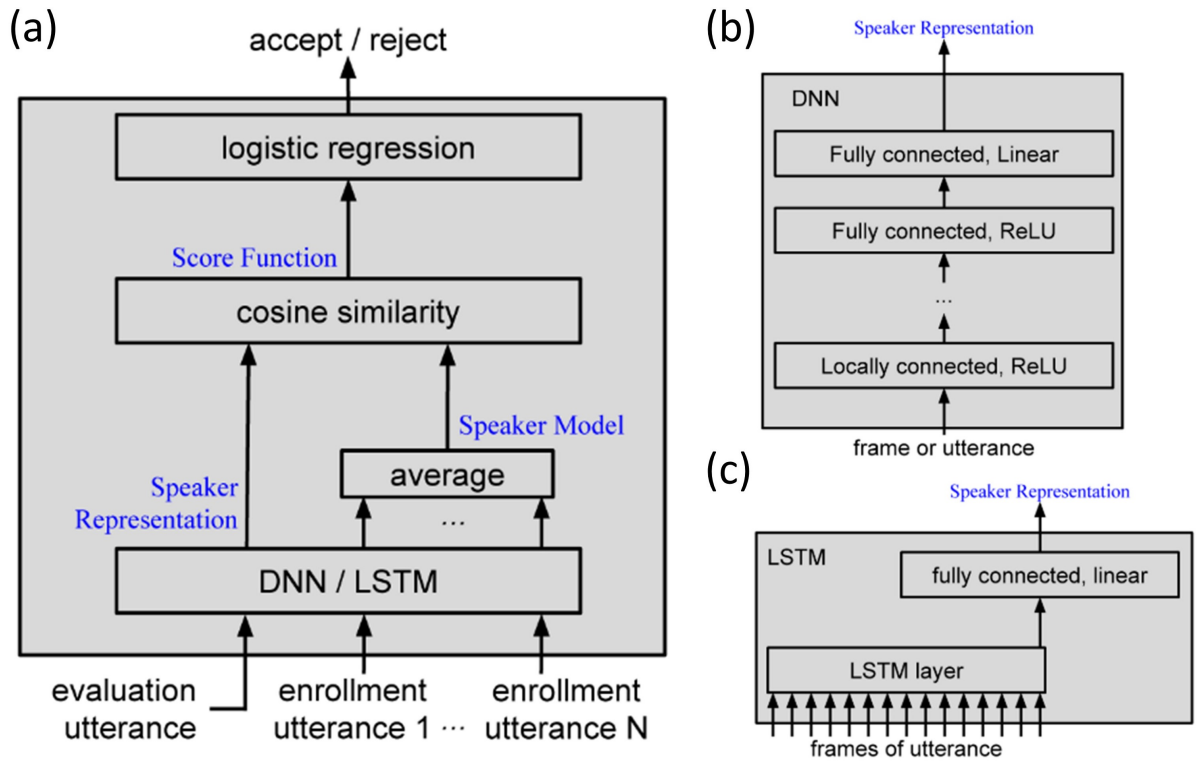


Figure 11.5.1: (a) End-to-end architecture, combining training of speaker representations (DNN/LSTM box), estimation of a speaker model based on up to N "enrollment" utterances (in blue), and verification (cosine similarity/logistic regression boxes). (b) Deep neural network (DNN) with a locally-connected layer followed by fully-connected layers. (c) Long short-term memory recurrent neural network (LSTM) with a single output. Modified from [8].

11.5.1.2 *Generalized end-to-end system*

To help the model learn more discriminating features, one improvement is fetch more diverse sample (i.e., more samples from different classes) and train the network in a way to cluster embeddings from the same class and place cluster centers of different classes away from each other.

Specifically, in a batch, we fetch $N \times M$ utterances from N different speakers and each speaker has M utterances. Each feature vector $x_{ji}(1 \leq j \leq N \text{ and } 1 \leq i \leq M)$ represents the features extracted from speaker j utterance i.

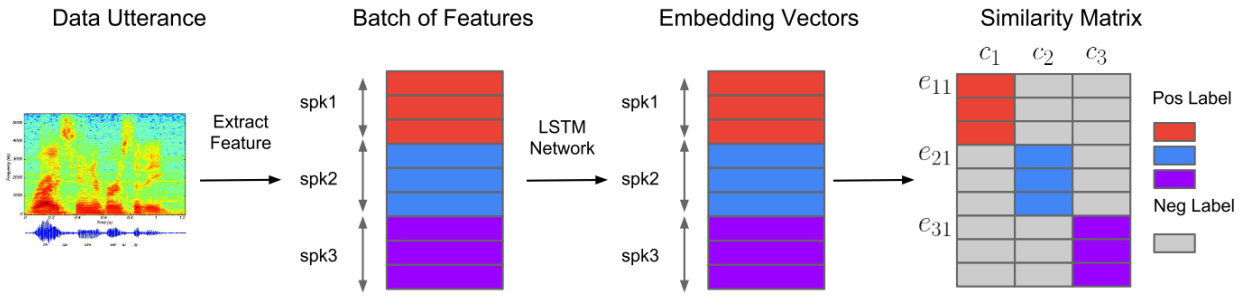


Figure 11.5.2: System overview. Different colors indicate utterances/embeddings from different speakers. Image from [13].

The features extracted from each utterance x_{ji} are fed into a backend classifier consisting of an LSTM network and a linear layer. We denote the output of the entire neural network as $f\left(x_{ji};\theta\right)$ where θ represents all parameters of the neural network. The embedding vector (d-vector) is defined by

$$e_{ji} = \frac{f\left(x_{ji};w\right)}{\left\|f\left(x_{ji};w\right)\right\|_2}$$

where e_{ji} represents the embedding vector of the j th speaker's ith utterance.

Among embedding vectors from different speakers, we can construct a similarity matrix $S_{ji,k}$ consisting of scaled cosine similarities between each embedding vector e_{ji} to all centroids c_k ($1 \leq j,k \leq N$, and $1 \leq i \leq M$)

$$S_{ji,k} = w \cdot \cos\left(e_{ji}, c_k\right) + b$$

where w and b are learnable parameters. We also constrain the weight to be positive $w > 0$ such that the similarity to be larger when cosine similarity is larger.

We consider two types of loss functions to achieve discriminating embeddings. First type of loss is softmax loss, where we put a softmax on $S_{ji,k}$ for $k = 1,\ldots,N$ that makes the output equal to 1 iff $k = j$, otherwise makes the output equal to 0. Thus, the loss on each embedding vector e_{si} could be defined as:

$$L\left(e_{ji}\right) = -S_{ji,j} + \log\sum_{k=1}^{N}\exp\left(S_{ji,k}\right)$$

Minimizing the softmax loss function will push each embedding vector close to its centroid and pull it away from all other centroids.

Another type of loss is the contrast loss, which is defined on positive pairs and most aggressive negative pairs, as:

$$L\left(e_{ji}\right) = 1 - \sigma\left(S_{ji,j}\right) + \max_{1 \le k \le N} \sigma\left(S_{ji,k}\right)$$

where $\sigma(x) = 1/\left(1 + e^{-x}\right)$ is the sigmoid function.

Thus, contrast loss allows us to focus on difficult pairs of embedding vector and negative centroid.

By minimizing the contrast loss function, we achieve two goals:

- A speaker's embedding is pushed towards its true speaker's voice print centroid.
- A speaker's embedding is pushed away from the highest similar false speaker's voice print centroid.

In experiments, we found both loss functions are useful.

11.6 Pretrained speech models

11.6.1 Overview

Recently, representation learning from unlabeled data has achieved remarkable success[14] in the field of natural language processing. The idea of representation learning can be applied to speech recognition given the abundant availability of unlabeled audio data.

Audio speech signals possess rich information, including phonemes, words, semantic meanings, tone, emotion, and speaker characteristics. By learning from large scale unlabeled data, representation learning for speech data aims to extract high-level structure that can be easily adapted for downstream tasks.

11.6.2 Autoregressive predictive coding

High-level features can be obtained by performing autoregressive prediction[15]. Given a sequence of N acoustic features $(x_1, x_2, \ldots, x_T)$, we can construct a RNN to predict the future based on historical observations.

Because acoustic features often exhibit local smoothness, we need to ask the model to predict frames n steps ahead to learn more global structures rather than the local information in the signals. The model parameter is learned by minimizing L_1 loss given by

$$\sum_{i=1}^{T-n} \left| x_{i+n} - y_i \right|$$

where $y_1, ...y_T$ is the model predicted output. To enable deep RNN structure, we can also introduce residual connection to facilitate gradient flow and speed up training.

The feature used to downstream tasks can be extracted from outputs of different RNN layers.

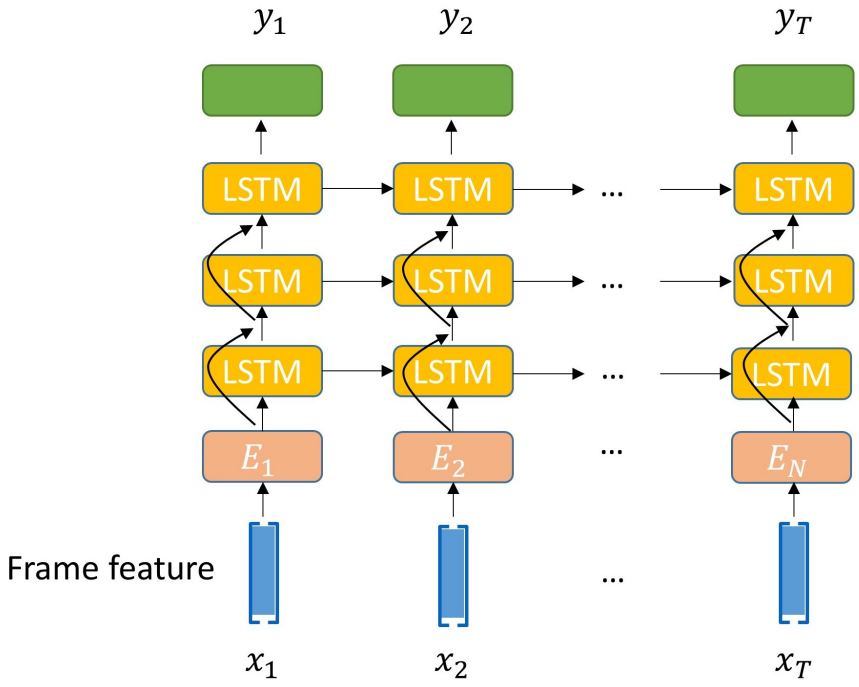


Figure 11.6.1: Pretraining model architecture with Autoregressive Predictive Coding (APC).

11.6.3 BERT-style pre-trained model

11.6.3.1 *Mockingjay*

Mockingjay is a speech encoder [16] inspired by BERT, which utilizes multi-layer transformer encoders and multi-head self-attention mechanism to achieve context-aware encoding.

The pretraining task is to predicte masked acoustic features from past and future signal with L1 loss. During training, masked frames (will not be used in self-attention) are given, and the model learns to reconstruct and predict the original frames. In addition, to cope with long speech signals, down-sampling on input features is used. Similar to the original transformer, sinusoidal positional encoding is used to encode order information. Note that we need to transform acoustic features first before we add positional encoding together.

To enhance model robustness, the masking strategy is dynamic (similar to [17]), where we randomly sample masking patterns every-time and apply them to sequences before they are fed into the model.

Different from language embedding features, acoustic features are quite similar to nearby frames (resulting from the physics of sound generation). To avoid the model exploiting this local smoothness, marks are applied up to C_{num} consecutive frames (say $C_{num} = 7$).

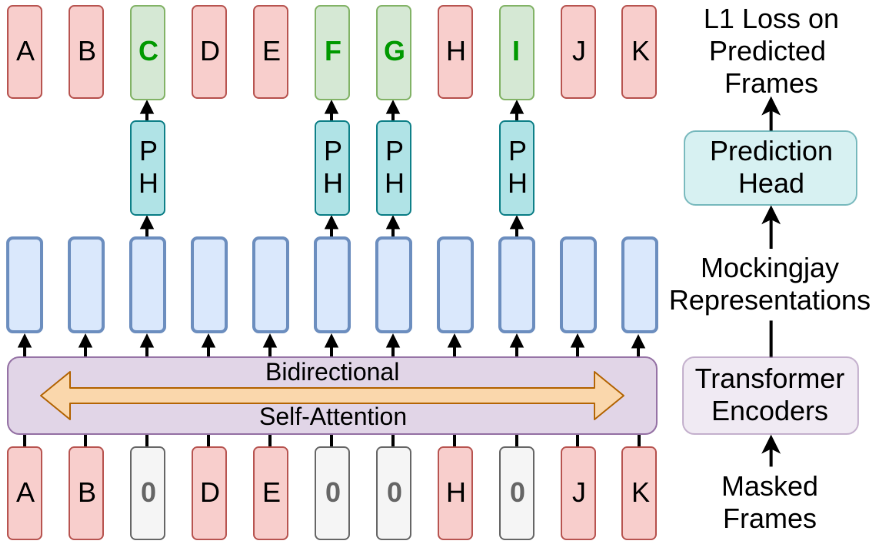


Figure 11.6.2: Masked acoustic training model for Mockingjay architecture. Image from [16].

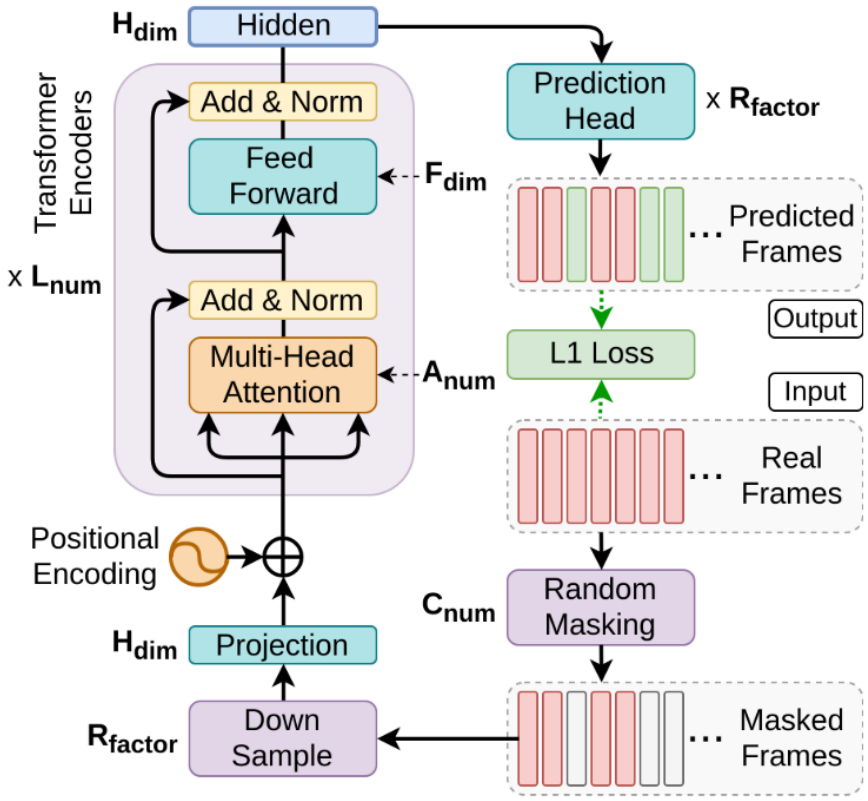


Figure 11.6.3: Mockingjay pretraining framework. Image from [16].

11.6.3.2 *TERA*

Another pretraining model TERA [18], which stands for Transformer Encoder Representation from Alternation, employs similar BERT architecture but pretrains on multiple novel self-supervised tasks.

The auxiliary pretraining tasks consist of reconstructing features from altered counterparts. The alternation applies to three different dimensions:

- **Time alteration**, where a certain percentage of input frames are masked or corrupted during training, and the model attempts to reconstruct them from neighboring frames.
- **Channel or frequency alteration**, where features for randomly selected consecutive frequencies are set to zero
- **Magnitude alteration**, where sampled Gaussian noise are applied to augment the magnitude of input sequences with certain probability.

By reconstructing the corrupted features from neighboring frames, the model learns representative features from unlabeled data.

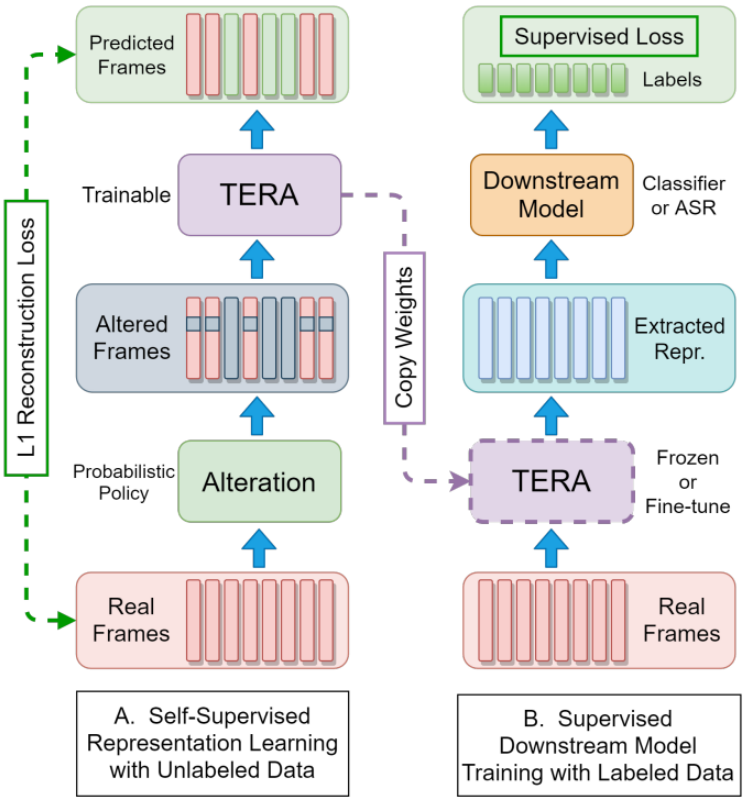


Figure 11.6.4: The self-supervised TERA architecture for audio representation learning.Image from [18].

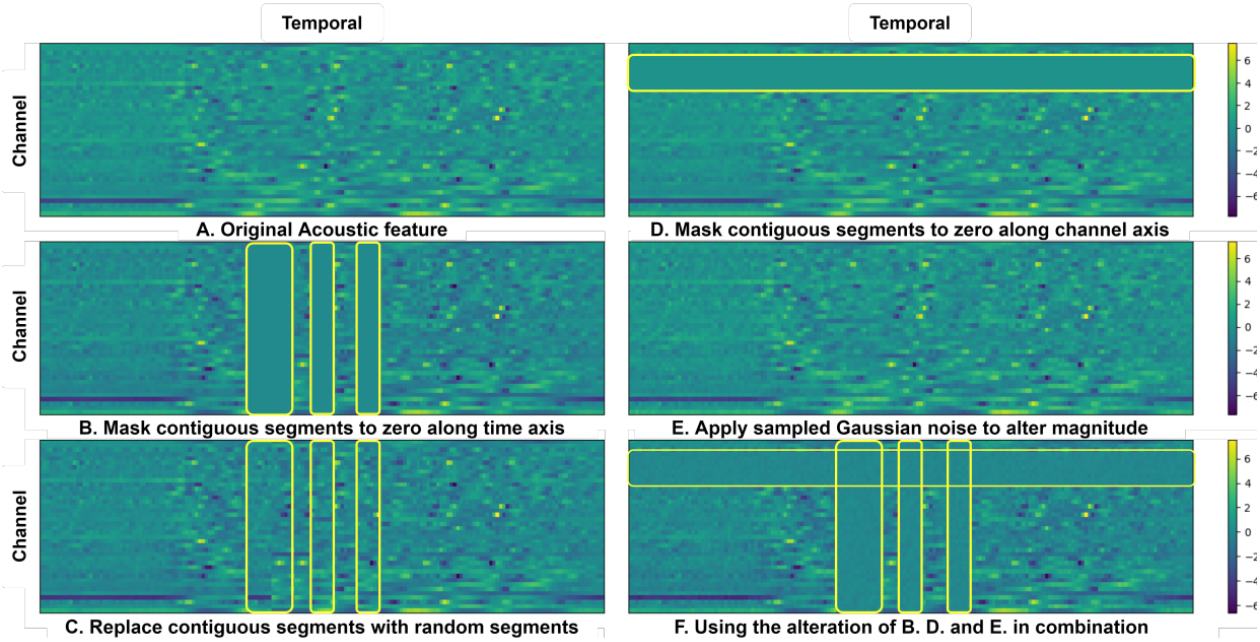


Figure 11.6.5: Different inputs with various alteration applied for the proposed auxiliary objective. Image from [18].

11.7 Notes on Bibliography

General books on speech recognition include [19][20][21][22].

Relevant software include PyTorch-Kaldi [23], Kaldi, S3PRL[24].

BIBLIOGRAPHY

1. Lyons, R. *Understanding Digital Signal Processing: Unders Digita Signal Proces_3* ISBN: 9780137028528 (Pearson Education, 2010).
2. Chan, W., Jaitly, N., Le, Q. V. & Vinyals, O. *Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition* in *ICASSP* (2016).
3. Graves, A., Fernández, S., Gomez, F. & Schmidhuber, J. *Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks* in *Proceedings of the 23rd international conference on Machine learning* (2006), 369–376.
4. Graves, A. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711* (2012).
5. Reynolds, D. A., Quatieri, T. F. & Dunn, R. B. Speaker verification using adapted Gaussian mixture models. *Digital signal processing* **10,** 19–41 (2000).
6. Prince, S. J. & Elder, J. H. *Probabilistic linear discriminant analysis for inferences about identity* in *2007 IEEE 11th International Conference on Computer Vision* (2007), 1–8.
7. Ioffe, S. *Probabilistic linear discriminant analysis* in *European Conference on Computer Vision* (2006), 531–542.
8. Heigold, G., Moreno, I., Bengio, S. & Shazeer, N. *End-to-end text-dependent speaker verification* in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), 5115–5119.
9. Snyder, D., Garcia-Romero, D., Sell, G., Povey, D. & Khudanpur, S. *X-vectors: Robust dnn embeddings for speaker recognition* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), 5329–5333.
10. Zhu, Y., Ko, T., Snyder, D., Mak, B. & Povey, D. *Self-Attentive Speaker Embeddings for Text-Independent Speaker Verification.* in *Interspeech* (2018), 3573–3577.
11. Okabe, K., Koshinaka, T. & Shinoda, K. Attentive statistics pooling for deep speaker embedding. *arXiv preprint arXiv:1803.10963* (2018).
12. Ravanelli, M. & Bengio, Y. *Speaker recognition from raw waveform with sincnet* in *2018 IEEE Spoken Language Technology Workshop (SLT)* (2018), 1021–1028.
13. Wan, L., Wang, Q., Papir, A. & Moreno, I. L. *Generalized end-to-end loss for speaker verification* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), 4879–4883.

14. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

15. Chung, Y. A., Hsu, W. N., Tang, H. & Glass, J. An unsupervised autoregressive model for speech representation learning. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* **2019-Septe,** 146–150. ISSN: 19909772. arXiv: 1904.03240 (2019).

16. Liu, A. T., Yang, S.-w., Chi, P.-H., Hsu, P.-c. & Lee, H.-y. *Mockingjay: Unsupervised speech representation learning with deep bidirectional transformer encoders* in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2020), 6419–6423.

17. Liu, Y. *et al.* Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

18. Liu, A. T., Li, S.-W. & Lee, H.-y. TERA: Self-Supervised Learning of Transformer Encoder Representation for Speech. *arXiv preprint arXiv:2007.06028* (2020).

19. Rabiner, L., Rabiner, L. & Juang, B. *Fundamentals of Speech Recognition* ISBN: 9780130151575 (PTR Prentice Hall, 1993).

20. Huang, X., Acero, A., Hon, H.-W. & Reddy, R. *Spoken language processing: A guide to theory, algorithm, and system development* (Prentice hall PTR, 2001).

21. Yu, D. & Deng, L. *Automatic Speech Recognition: A Deep Learning Approach* ISBN: 9781447157793 (Springer London, 2014).

22. Li, J., Deng, L., Haeb-Umbach, R. & Gong, Y. *Robust Automatic Speech Recognition: A Bridge to Practical Applications* ISBN: 9780128026168 (Elsevier Science, 2015).

23. Ravanelli, M., Parcollet, T. & Bengio, Y. *The pytorch-kaldi speech recognition toolkit* in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), 6465–6469.

24. Liu, A. T. & Shu-wen, Y. *S3PRL: The Self-Supervised Speech Pre-training and Representation Learning Toolkit* 2020.