Topics not covered

- VQ
- N-Grams

Chapter 1

Speech Recognition in a Medical Environment

1.1 Background

Automatic speech recognition (ASR) has been a topic that interested many from an early age. Many consider it to have started in the 1950s with Bell Labs' *Audrey* [Moskvitch, 2017], which was able to do a single-speaker digit recognition. Seventy years later, the ASR technologies have grown so that they are present within our personal homes, with Amazon Alexa and Google Home devices [Fowler, 2018], as they have become more affordable. As the speech technologies advance, they approach the concept of ubiquitous computing (often also known as ambient intelligence), which is highly desirable for many industries. Although most of the ASR systems available for commerce are trained in normal language and not medical language, one industry that could greatly benefit from ASR technologies is the health care system, given that the ASR system is able to understand medical language.

Health care is a system which has a very high demand and the services are required to be as detailed as possible due to various reasons, one of which is the concept that it is paramount for a hospital to have a clear and rich track of a patient's medical history. Currently, in order to maintain a patient's medical history, a physician sees multiple people during his/her working hours, and only after their shift is over, does (s)he sit down to write the medical notes. The lag in between seeing a patient and taking notes may sometimes go up to 8-10 hours, and then those notes are often inaccurate. Having an ASR system in a medical environment could greatly help in keeping track of a patient's medical history, where a physician could easily dictate the notes. In a hospital environment, however, such as in an intensive care unit (ICU), a physician who is trying to dictate notes may find him or herself in trouble, as in the ICU, there are multiple background sounds from machines, multiple people speaking, and a great amount of white noise.

Current companies have been creating ASR systems that can understand medical language and allows physicians to dictate their medical notes. One major company that has been "dominating" a lot of the market is Nuance, with their Dragon Medical system. Unfortunately, their dictation system is not yet capable of inferring punctuation marks and markup language onto the text, thus, in order to dictate a segment such as: "37-year-old female presents complainint of urinary frequency, urgency and dysuria along with hematuria and low-grade fever." needs to be dictated as:

"37-year-old female presents complainint of urinary frequency **comma** urgency and dysuria along with hematuria and low-grade fever **period**"

Although the system has very high accuracy results, the system works best when one is in a quiet environment, which is not always a realistic scenario. The goal of this project is to create a transcription engine that can recognize medical language in a noisy environment.

1.2 Preliminary Data

1.2.1 Comparison of Different Commercial Engines

Compare commercial (+ CMU) Text-¿Voice-¿ Text MIMIC II + 20kLeagues

1.2.2 Grade Reading Level of Various Texts

MIMIC II + 20kLeagues

1.3 Automatic Speech Recognition Schematic for a Medical Setting

In order to have a speech recognition system to be able to understand medical language, it is important to obtain a big picture of the project.

[INSERT FIGURE]

As one can see from the diagram above, the sound signals would come to a microphone or an array of microphones, which would then be passed through a system that performs blind source separation on the sound signal. It is very possible that, in a hospital environment, there may be multiple sources, such as the heart rate monitor, the ventilator, the healthcare providers talking, and others. A blind source separation algorithm would ensure to separate the audio channel to multiple sources. Once the sources have been separated, there would be a classification methodology to identify the physician from the patient, and ...

1.3.1 AI-Complete and ASR-Complete

Although there have been many advancements in the speech technologies, especially through black boxes such as neural networks, and there currently exists very accurate engines to do voice recognition, it still remains to be an unsolved problem. The speech recognition problem is ASR-complete, which falls under the umbrella of AI-complete, which, by analogy of NP-completeness, means that the problem is hypothesized to deal with multiple parts of the AI world, and it cannot be solved by a single algorithm. Current technologies are not able to solve AI-complete problems, and they often require human computation. Since speech recognition is an AI-complete problem, there is still a lot of room for improvement in the fundamental problems that have not yet been solved, and although deep learning has recently shown promising results, deep learning neural networks (DNN) do not necessarily solve fundamental problems, as no one has a truthful understanding of how they work and behave. Therefore, there are a lot of possibilities for the formulation design of the framework (i.e. pipeline).

1.4 Speech Enhancement

Speech enhancement is one of fundamental problems of speech that has not yet been fully solved yet, as noise is subjective, and there isn't necessarily one specific way of performing a fully accurate noise reduction for any type of noise. As previously mentioned before, a ventilator and heart rate monitor are examples of background noise that do not necessarily have a Gaussian distribution (i.e. it is not a white noise), yet, for ASR, they are both irrelevant. There are a number of methodologies that have been created to attempt to solve the problem of speech enhancement, and they may potentially be useful to be implemented onto the pipeline.

1.4.1 Recurrent Neural Networks for Noise Suppression

The work on recurrent neural networks (RNN) was originally shown by [Jordan, 1986] and [Barak, 1988], and it finally came to be coined by [Cleeremans et al., 1989]. They have shown to work very efficiently with time-series data as their cyclic nature allows them to adapt to incoming inputs in a sequential form, working often times better than convolutional neural networks (CNN), when dealing with time-series data. Other variants of RNNs have been created, which allow them to store "memory" for future cases, with Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], which were then further expanded with gates for Gated Recurrent Units (GRU) [Gers et al., 2000] specifically to allow such units to forget information, and a visual representation of LSTMs and GRUs are shown on Figure 1.1

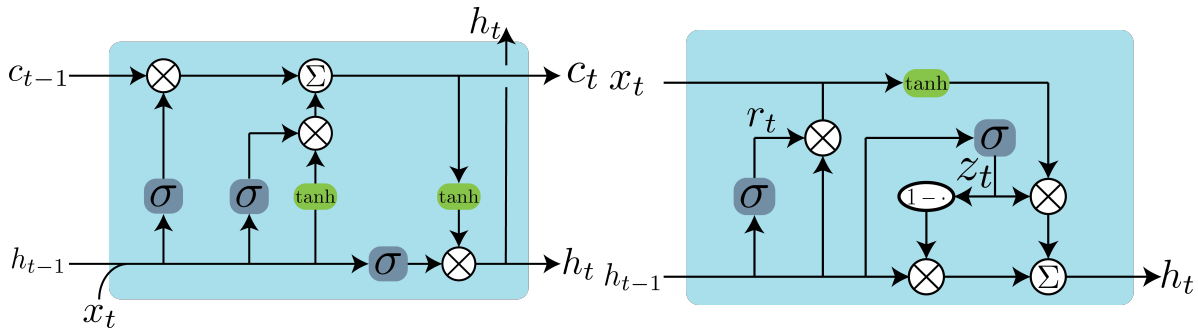

Figure 1.1: A visual representation of the LSTM cell (Left) and the GRU cell (Right). The LSTM contains three representations of its states: the cell state/memory c_t, the hidden states h_t, and the input state x_t, all of which occurs at time t. On the GRU, there are the input state x_t, the hidden state h_t, the update gate z_t and the reset gate r_t. The σ block is used to represent a sigmoid function, and the tanh block is used to represent a hyperbolic tangent for both images. Please see A.1 for the mathematical expressions of the cells above

Because of the nature of the network being able to receive continuous sequential inputs, it becomes a very attractive model to potentially behave as an adaptive filter. Hence, members from the Xiph.Org Foundation, a non-profit organization, along with Mozilla have created a RNN architecture that is based on GRUs called RRNoise [Jean-Marc et al., 2017], which, after trained, is able to reduce a lot of the background noise of audio files, thus enhancing the signal. Below on Figure 1.2 is the topology of the RRNoise architecture, where it outputs voice activity detection (VAD) as well as the gains from the input features.

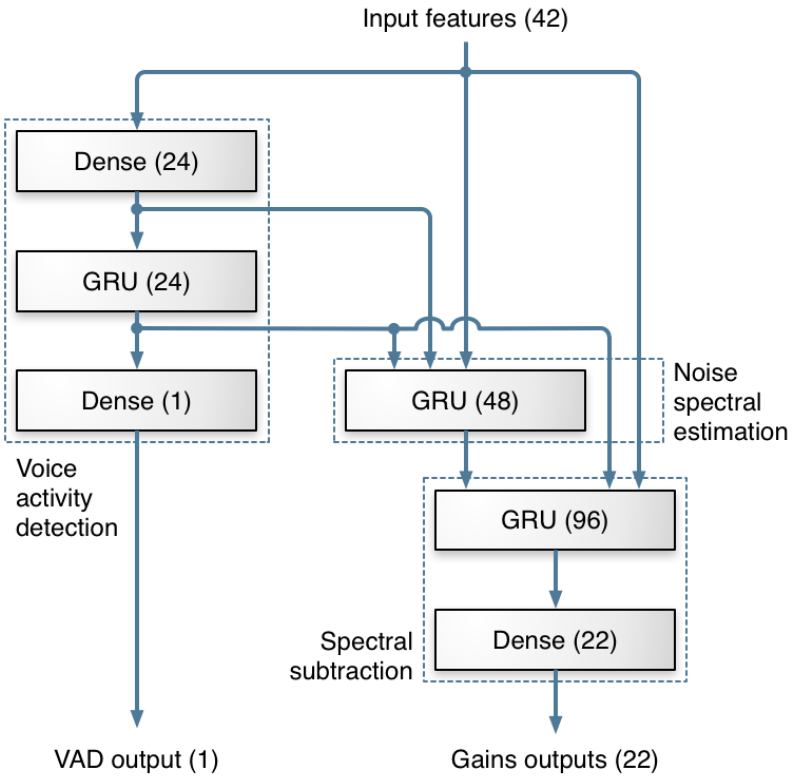

Figure 1.2: The topology of the RRNoise network used to perform noise reduction. This figure is taken from [Jean-Marc et al., 2017]

1.4.2 Non-Negative Matrix Factorization

Another very neat way of performing speech enhancement is via non-negative matrix factorization. Please skip to the Blind Source Separation's subsection on Non-Negative Matrix Factorization [Ng et al., 2001]

1.5 Blind Source Separation

Given that a recording seldom contains one and only one speaker speaking at a time, an ASR may find itself troubled in detecting who is the speaker and who should be taken as the background. This is namely known as the Cocktail Party Effect. Its name originates from the idea that, in a cocktail party, there are multiple people speaking to one another. While a human is able to have a conversation with another, focusing on his/her friend while having all of the other speakers (i.e. sources) as background, an ASR system is not able to do so. Hence it is important to be able to separate a mixed signal to then be able to focus on a single individual and this is called blind source separation (BSS). There are two methodologies that are proposed to perform BSS: independent component analysis, and non-negative matrix factorization.

1.5.1 Independent Component Analysis

1.5.2 Non-Negative Matrix Factorization (NMF) and Sparse NMF (SNMF)

Background

One of the major drawbacks from any methodology in ICA is that it requires N or more observations (i.e. microphone recordings) for N sources (e.g. speakers, noise), which, depending on the circumstance, may be limiting. The method of non-negative matrix factorization (NMF), however, is able to perform BSS with very high accuracies, while using a single-channel recording. NMF originated in the field of chemometrics, where chemists were trying to identify contents of solutions via positive matrix factorization [Paatero and Tapper, 1994] [Anttila et al., 1995]. It was later further developed in [Lee and Seung, 1999] and algorithms were then better formulated in [Lee and Seung, 2000].

The concept behind NMF is that a mixed signal should be able to be decomposed into its unmixed signals, under the assumption that all of the superpositioned signals are non-negatively added to one another. In other words, they are positively added, but their value may also be zero. Conceptually, this makes absolute sense. If one see the work in [Lee and Seung, 1999], one will see that many of the components added are interpretations of existing physical features, such as eyebrows and mustache. It does not make sense to add a negative eyebrow to reconstruct one's face. Hence, in retrospect, it makes complete sense why this methodology was being used in chemometrics, as it does not make sense to add a negative sulfate ion to a solution.

One may ask how is this factorization different from other types of factorizations, such as principal component analysis (PCA). First, every factorization has different purposes; for instance, a Cholesky decomposition was specifically made to obtain lower computation complexity in order to invert matrices, where as PCA was made to obtain information about its principal orthogonal components. Second, when looking at the comparison between a signal reconstruction or separation from an NMF and from PCA, PCA poses a lot of problems. PCA only allows for the feature space to represent vectors that are orthogonal, which is not always the case. In the sense of speech, it is almost impossible for one speech signal to be orthogonal to another one. PCA also assumes that the probability distribution of the data follows a multivariate Gaussian, which again is not always the case. A NNF is very promising as it performs a matrix decomposition, which is very desirable for computational reconstruction in the last step of the BSS, and it is able to attain reconstructions just as detailed as some of the other reconstruction techniques, plus one is allowed to determine how many components to use.

Here, the notation follows the notations shown in [Schmidt and Olsson, 2006].

Consider the magnitude of a spectrogram of a mixed speech signal $\boldsymbol{Y} \in \mathbb{R}^{M \times N}$, which is a summation of R source signals (i.e. $\boldsymbol{Y} = \sum_{i=1}^{R} \boldsymbol{Y}_i$). The spectrogram can be sparsely represented in an overcomplete basis as

$$\boldsymbol{Y} = \boldsymbol{D}\boldsymbol{H}$$

where $\boldsymbol{D} \in \mathbb{R}^{M \times R}$ represents a compendium of dictionaries with R columns, which can be chosen by the user, and $\boldsymbol{H} \in \mathbb{R}^{R \times N}$ represents a code matrix, which is sparse and it contains code matrices associated with the

dictionaries in $\boldsymbol{D}$. The matrix $\boldsymbol{D}$ and $\boldsymbol{H}$ can be represented as

$$\boldsymbol{D} = \begin{bmatrix} | & | & & | \\ | & | & & | \\ | & | & & | \\ \boldsymbol{D}_1 & \boldsymbol{D}_2 & \cdots & \boldsymbol{D}_R \\ | & | & & | \\ | & | & & | \\ | & | & & | \end{bmatrix}$$

$$\boldsymbol{H} = \begin{bmatrix} - & - & - & - & \boldsymbol{H}_1 & - & - & - & - & - & - \\ - & - & - & - & \boldsymbol{H}_1 & - & - & - & - & - & - \\ & & & & \vdots & & & & & & \\ - & - & - & - & \boldsymbol{H}_R & - & - & - & - & - & - \end{bmatrix}$$

A simple version of how a NMF would work is shown in Figure 1.3, where the figure below was inspired on the works from [Schmidt and Olsson, 2006].

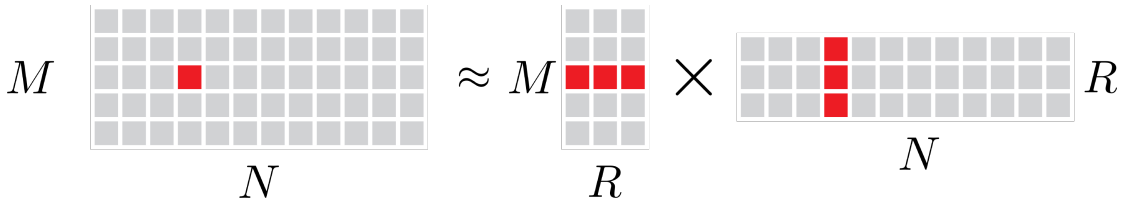

Figure 1.3: A very simple schematic of a non-negative matrix factorization, where the matrix on the left represents the mixed signal, the matrix on the center represents the dictionary matrix, and the matrix on the right represents the coding matrix

There are two different ways to obtain the desired decomposition, and that is based on what optimization approach one wishes to take. One approach is to minimize the squared Euclidean-based Frobenius norm, $\|\boldsymbol{Y} - \boldsymbol{DH}\|_F^2$, and another approach is to minimize the divergence, $\mathcal{D}(\boldsymbol{Y}\|\boldsymbol{DH})$, both cases with respect to $\boldsymbol{W}$ and $\boldsymbol{H}$, subject to the contraints that $\boldsymbol{D}, \boldsymbol{H} \geq 0$ [Lee and Seung, 2000]. Both of their definitions are shown below, where the subscript i represents the i^{th} row, and the j represents the j^{th} column. Notice that the divergence measurement below is not the Kullback-Leibler (KL) divergence, but it does converge to the KL divergence if $\sum_{ij} A_{ij} = \sum_{ij} B_{ij} = 1$s

$$\begin{cases} \text{Norm:} & \|A - B\|_F^2 = \sum_{i,j} (A_{ij} - B_{ij})^2 \\ \text{Divergence:} & \mathcal{D}(A\|B) = \sum_{i,j} \left(A_{ij} \log \left(\frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij} \right) \right) \end{cases}$$

Because this is an iterative algorithm, if one follows the norm minimization, the update rules are:

NMF Algorithm for Norm Minimization

1.Initialize $\boldsymbol{D}, \boldsymbol{H} \geq 0$

2.$\boldsymbol{H}_{ij} \leftarrow \boldsymbol{H}_{ij} \frac{(\boldsymbol{D}^T \boldsymbol{Y})_{ij}}{(\boldsymbol{D}^T \boldsymbol{DH})_{ij}}$

3.$\boldsymbol{D}_{ij} \leftarrow \boldsymbol{D}_{ij} \frac{(\boldsymbol{Y}\boldsymbol{H}^T)_{ij}}{(\boldsymbol{DHH}^T)_{ij}}$

4.Repeat 2,3 until stability

If one follows the divergence minimization, the update rules are

NMF Algorithm for Divergence Minimization

1.Initialize $\boldsymbol{D}, \boldsymbol{H} \geq 0$

2.$\boldsymbol{H}_{ij} \leftarrow \boldsymbol{H}_{ij} \frac{\sum_k \boldsymbol{D}_{ki} \boldsymbol{Y}_{kj} / (\boldsymbol{DH})_{kj}}{\sum_l \boldsymbol{D}_{li}}$

3.$\boldsymbol{D}_{ij} \leftarrow \boldsymbol{D}_{ij} \frac{\sum_k \boldsymbol{H}_{jk} \boldsymbol{Y}_{ik} / (\boldsymbol{DH})_{ik}}{\sum_l \boldsymbol{H}_{jl}}$

4.Repeat 2,3 until stability

If one performs an NMF, a visual representation is shown on Figure 1.4, where the figure below was inspired on the works from [Schmidt and Olsson, 2006].

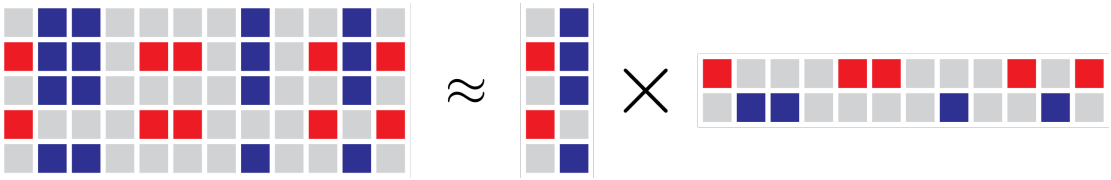

Figure 1.4: A visual example of an NMF result, with two sources being separated.

The algorithms for NMF were further improved by [Eggert and Korner, 2004], where the authors created a Sparse NMF. Their work was inspired by the basic NMF problems that, when one had overcomplete dictionaries, there was no well defined solution such that you would obtain sparse solutions. Thus, they wanted to make sure to obtain sparse solutions, especially on the coding matrix. To tune the sparsity measurement, the authors simply applied a L1 regularization based on the coding matrix coefficients on the cost functional of the norm, and called it a Sparse NMF. In other words, the cost (or energy) functional that originally was $\mathcal{E} = \|Y - \bar{\boldsymbol{D}}\boldsymbol{H}\|_F^2$ became

$$\mathcal{E} = \|\boldsymbol{Y} - \bar{\boldsymbol{D}}\boldsymbol{H}\|_F^2 + \underbrace{\lambda \sum_{i,j} H_{ij}}_{L1 \text{ regularization}} \qquad s.t. \quad \boldsymbol{D}, \boldsymbol{H} \geq 0$$

where $\bar{\boldsymbol{D}}$ still follows the notation of [Schmidt and Olsson, 2006], representing a column-wise normalized dictionary matrix, and λ is the parameter to control the degree of sparsity. To solve this optimization problem, if one was to set $\boldsymbol{R} = \boldsymbol{DH}$, then one would obtain the following algorithm. Note, for this one case, that the dot sign $\cdot$ and and division operator $\frac{a}{b}$ represent point-wise multiplication and division

Sparse NMF Algorithm for Frobenius Norm L1 Regularized Minimization

1.Initialize $\boldsymbol{D}, \boldsymbol{H} \geq 0$

2.$\boldsymbol{H}_{ij} \leftarrow \boldsymbol{H}_{ij} \cdot \frac{\boldsymbol{Y}_i^T \bar{\boldsymbol{D}}_j}{\boldsymbol{R}_i^T \bar{\boldsymbol{D}}_j + \lambda} \in \mathbb{R}^1 \cdot \frac{\mathbb{R}^{1\times M}\mathbb{R}^{M\times 1}}{\mathbb{R}^{1\times M}\mathbb{R}^{M\times 1} + \mathbb{R}^1}$

3.$\boldsymbol{D}_{ij} \leftarrow \boldsymbol{D}_{ij} \cdot \frac{\sum_i \boldsymbol{H}_{ij}[\boldsymbol{Y}_i + (\boldsymbol{R}_i^T \bar{\boldsymbol{D}}_j)\bar{\boldsymbol{D}}_j]}{\sum_i \boldsymbol{H}_i j[\boldsymbol{R}_i + (\boldsymbol{Y}_i \bar{\boldsymbol{D}}_j)\bar{\boldsymbol{D}}_j]} \in \mathbb{R}^{M\times 1} \cdot \frac{\sum_i \mathbb{R}^1 + (\mathbb{R}^{1\times M}\mathbb{R}^{M\times 1})\mathbb{R}^{M\times 1}}{\sum_i \mathbb{R}^1[\mathbb{R}^{M\times 1} + (\mathbb{R}^{1\times M}\mathbb{R}^{M\times 1})\mathbb{R}^{M\times 1}]}$

4.Repeat 2,3 until stability

All of the algorithms shown above are the steps necessary to train the algorithm, where $\boldsymbol{Y}$ contains all of the training data. In order to actually perform a speech reconstruction, one should simply perform the algorithms above, but to keep the dictionary matrix $\boldsymbol{D}$ fixed and update only the code matrix $\boldsymbol{H}$. The speech is then separated by conputing the reconstruction of parts of the sparse decomposition. In other words:

1. Apply NMF or Sparse NMF to learn the dictionaries of individual speakers

2. To separate the mixtures, keep $\boldsymbol{D}$ fixed, and only update $\boldsymbol{H}$
3. Reconstruct the signal by using the desired parts of the decomposition

Aproaches to Learn Dictionaries

As shown in [Schmidt and Olsson, 2006], there are multiple approaches to perform BSS with NMF or SNMF: unsupervised approach and segmenting the training data and solving multiple NMF problems

In the unsupervised approach, one would compute the SNMF over a very large dataset matrix $\boldsymbol{Y}$ of a single speaker to obtain a single dictionary. This, however, may take a lot of computation time. The second approach, used in [Schmidt and Olsson, 2006] involved segmenting the training data according to phoneme labels obtained by a speech recognition software with hidden-Markov models (HMM) for a single speaker, and creating a sparse dictionary for each phoneme, and finally , a final dictionary would be constructed by the concatenation of individual phoneme dictionaries. In order to create the dictionary of each phoneme, each type of phoneme would be concatenated together, to create that phoneme's dictionary, and then, the phonemes dictionaries would be concatenated together to create a dictionary that is representative of the speaker. Figure 1.5 shows an illustration on how to create the dictionary.

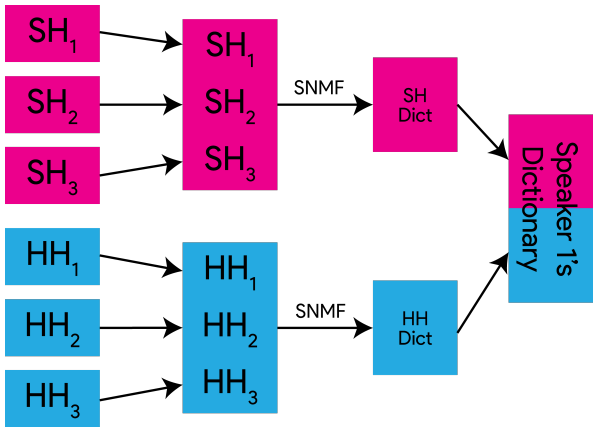

Figure 1.5: An illustration on how to create a speaker's dictionary $\boldsymbol{D}_i$

NMF for Speech Enhancement

The works from [Li and Xiao,] have shown that it is also possible to perform speech enhancement on a speech signal via non-negative matrix factorization. An illustration on Figure 1.6, inspired by the images on [Li and Xiao,], shows the procedure. It is possible to generate the dictionary matrix $\boldsymbol{D}$ and the coding matrix $\boldsymbol{H}$ from a noisy signal, however one would not know which of the subdictionaries $\boldsymbol{D}_i$ or the code submatrices $\boldsymbol{H}_i$ represent the ones associated with the speech and with the noise. Through spectral clustering (for technical details, see A.3), it is possible to group the subparts of the dictionary matrix and the code matrix so that it is differentiated between speech and noise. Given that the reconstruction process is based on linear algebra rules, it is then possible to simply multiply the dictionary matrix related to speech $\boldsymbol{D}_{speech}$ and the code matrix related to speech $\boldsymbol{H}_{speech}$ to obtain the cleaned speech.

Of course this solution is not guaranteed to remove every single noise source due to the possible stochasticity that is propagated to the eigenvectors of the Laplacian, and the fact that the solution is not unique, which is based on the initialization of the centroids of the k-means algorithm, but it could be a good fundamental method to implement.

1.6 Speech Recognition

1.6.1 Hidden-Markov Models and Gaussian Mixture Models

A well documented work that I've found on Gaussian mixture models was [Figueiredo and Jain, 2002], though works on the Expectation-Maximization (EM) algorithm have been shown as early as in [Duda and Hart, 1973].

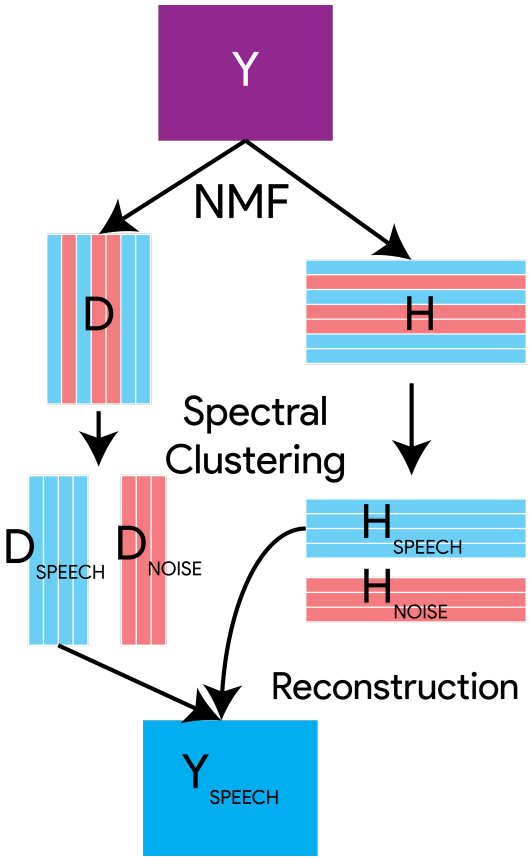

Figure 1.6: An illustration on how to use non-negative matrix factorization to denoise a signal, where $\boldsymbol{Y}$ represents the noisy signal, $\boldsymbol{D}$ represents the dictionary matrix obtained through NMF, and $\boldsymbol{H}$ represents the code matrix, also attained through the NMF.

Such models are often used for unsupervised clustering classification problems, where one assumes the probability distribution of different classes to follow a Gaussian. As mentioned before, it is critical that the data has compactness in order to make the assumption that the data distributions follow Gaussians, and if not, spectral clustering (see A.3) may be ussed for clustering problems. Nonetheless, for speech, depending on the features that one may be using, it is very safe to assume that they follow Gaussian distributions, which has led a lot of work in speech to be dependent on derivatives of Gaussian mixture models. For more detailed information on GMMs, please see A.2

One of the most important works for speech are the hidden Markov models (HMM). They were originally introduced by [Rabiner and Juang, 1986] (which happens to be yet one of the best tutorials on HMMs), and shown to work greatly on speech [Rabiner, 1989]. The Viterbi algorithm, which was first proposed in [VITERBI, 1967], and later formalized in [Forney, 1973], is a great way to the solve HMM problems, via dynamic programming. Many consider HMMs to be dynamic forms of GMMs, where a single point in time of a HMMs is essentially a GMM (assuming a Gaussian distribution between the classifications), but this fails if the prior probabilities are not Gaussian. Although, most of the people in the machine learning field use GMMs for clustering classification, it was primarily created in order to create models that, when brought together, created a better representation of an unknown distribution, in the same way that boosting is used to combine different ML methods that complement each other to create a better regressor or classifier. Nonetheless, it became very common to use HMMs for speech recognition. For instance, the work from [Schmidt and Olsson, 2006] could not be completed if the authors did not have an HMM module to output phonemes to segment their signals in order to create dictionaries for SNMF.

As an example, Figure 1.7 shows an example of a HMM, where the states are the phonemes of the word "shoot". As it can be seen, there are multiple entry points to the model, where they can start at SH, UW or T, and as time progresses, which is demonstrated as the repetition of the three states to the right, it is possible that the SH will progress to SH, or any of the other phonemes, and this propagates throughout the entire signal.

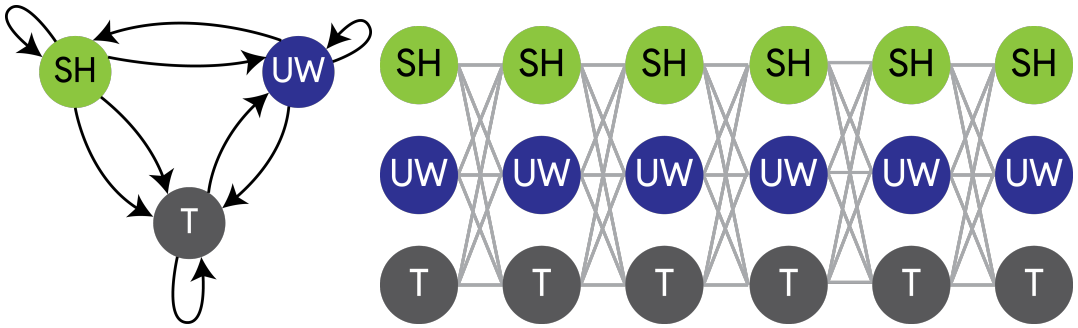

Figure 1.7: A simple HMM with the states being the phonemes (shown as ARPABET). The image on the left shows the HMM in its compact form, and the image on the right shows the HMM unrolled over time.

A set of features is computed over windows of a signal, and a set of features are extracted at each window. The Baum-Welch algorithm is then used to compute the statistics (i.e. "train") for the nodes and to compute the transition probabilities. Finally, when testing, the windows are classified over which states does it most likely belong to, such as with a GMM. The Viterbi-Trellis algorithm is then used to compute the optimal path over the trellis, thus defining all of the states over time. Because the algorithm is computed over a successive multiplication of the probabilities and transition probabilities, it is very likely that the numbers quickly decrease. Therefore, it is optimal to utilize log computations, especially for computers to be able to maintain float accuracies. Once done so, an optimal path is computed, which is depicted on Figure 1.8 as a black line.

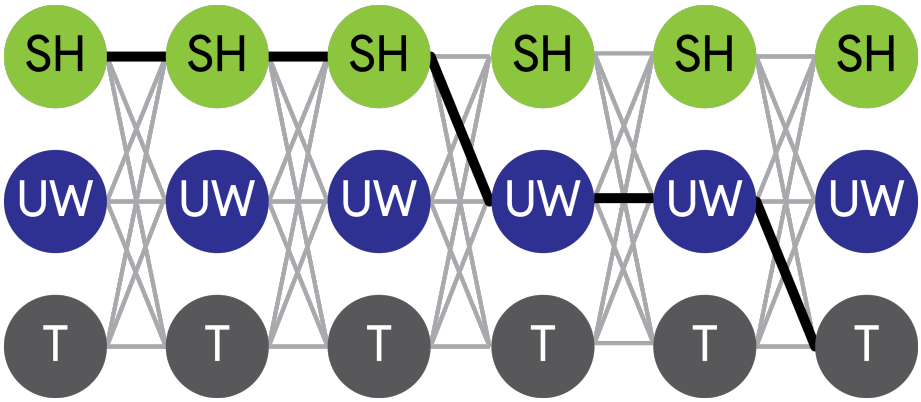

Figure 1.8: The theoretical Viterbi algorithm solution of a signal.

HMMs are often formulated where states have substates. Building on the example above, the phonemes could be broken down to substates, which is shown on Figure 1.9. On the illustration, dashed lines show the intra-state transition within each phonemes, and the solid lines represent the transition from one phoneme to another. Notice how, in a phoneme X, X_1 can only transition to itself or to a X_2. Likewise, X_2 can only transition to itself or to X_3, and X_3 can only transition to itself or to a new initial phoneme state Y_1.

Because of the powerful formulation which allows classifications to continuously repeat themselves over the analysis of an entire frame, HMM are one the more powerful techniques for time-series analysis or dynamic datastreams, while also having the ability to perform encoding over a signal.

1.6.2 Deep Learning Architectures

Another methodology that some consider it to be "state-of-the-art" is with deep learning, by using recurrent neural networks, which were previously described in 1.4.1, there have been promising results using CNNs [Zhang et al., 2016], as well as RNNs [Graves et al., 2013] [Graves and Jaitly, 2014], and LSTMs [Han et al., 2018]. The logic behind utilizing CNNs is the fact that a neuron would be a windowing function and classifier, which

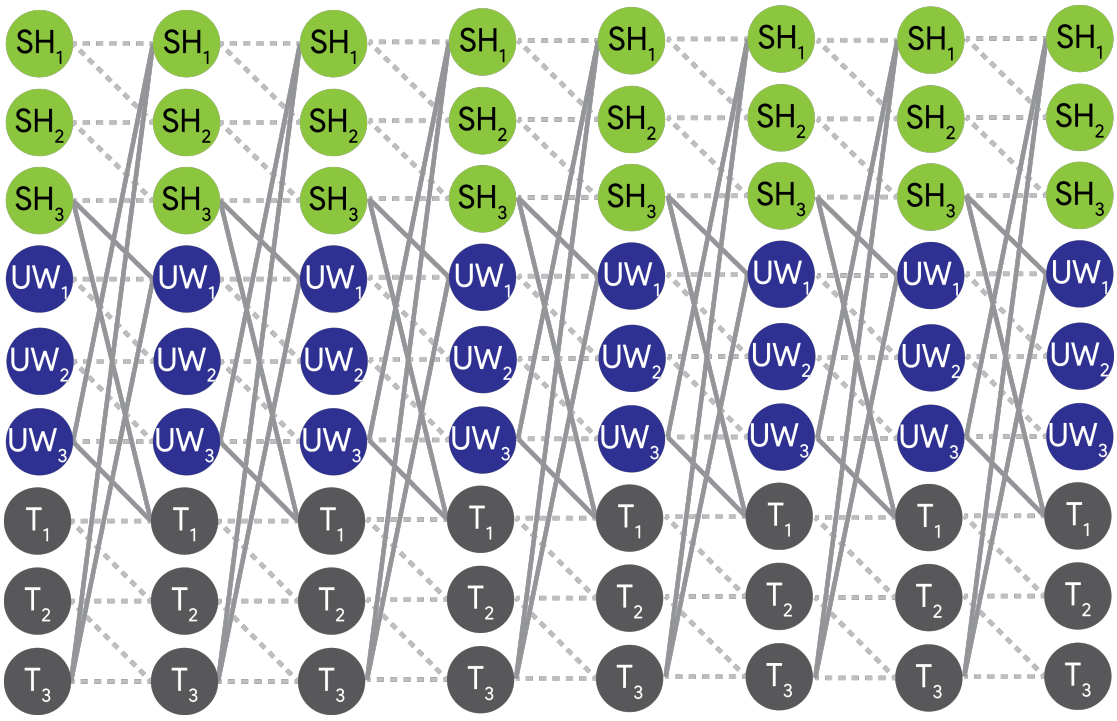

Figure 1.9: An extended version of the HMM for three phonemes, in which each phoneme contains three substates.

captures the content/interpretation of a signal as it slides over time, and thus it activates its successive neurons. However, because the authors provide no incentive of understanding the nature of their experiments, nor do they even include a discussion section over their work, and when they do, they simply state already known concepts in the field or regurgitate the aspects of their architecture, concluding the paper by stating that their future works will be to test on a larger dataset or try different architectures, I will move on with our discussion to more fruitful topics which contain explicit discussions with actual scientific approaches to solving problems.

1.7 Speaker Verification and Speaker Diarization

When using an ASR system in a medical setting, privacy and accuracy is paramount. In other words, it is extremely important that medical records do not get leaked for obvious reasons, and to make sure that not anyone is logging information into a patient's medical records is also of extreme importance. The concept of a speaker verification system becomes an attractive mechanism for an ASR, such that it ensures that important information is coming specifically from the physician, and not anyone else.

It is important to state that, in contrary to popular belief, speaker verification is not the same as speaker diarization (sometimes spelled as *diarisation*). Speaker verification is a methodology used to determine whether the speaker is the target speaker or not, and this technology was made for security purposes, which in the end of the pipeline, it terminates with a yes/no answer. Speaker diarization, on the other hand, is used to answer the question: "Who spoke when?" It is, however, often times simple to tie these two subfields together, since computation powers have been becoming very powerful, and many times, the adaptation occurs at the last step switching boolean answers to a multiclass boolean problem or vice-versa, therefore, this section presents technologies for both fields together, since what is really important is the meaty part of the methods.

One technique that has been considered state-of-the-art in the past few years is the i-vector methodology. In order to understand i-vectors, it is important to first understand Joint Factor Analysis (JFA), since i-vectors were based on the concepts from JFA. Prior to going into JFA, it is important to understand the difference between two types of session variations:

- **Inter-Speaker Variation**: This is a variation originated from two utterances from different speakers.

- **Inter-Session Variation**: This is a variation originated from utterances from the same speaker. This may be due to:
  - *Channel effects*: when utterances are recorded from different channels (e.g. microphones, environment)
  - *Intra-Speaker Variation*: when utterances vary due to the speaker's health or emotional state

1.7.1 Gaussian Mixture Model-Universal Background Model and Supervectors

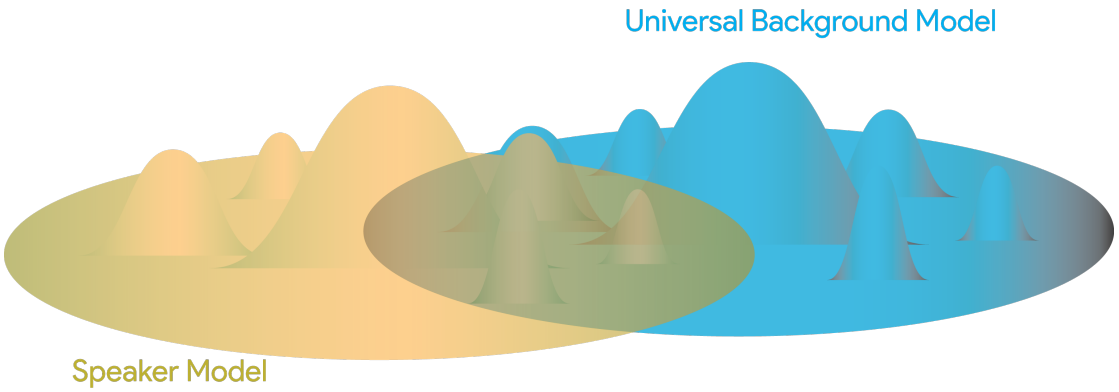

Figure 1.10: An illustration of a speaker model and the adapted speaker model.

The Gaussian Mixture Model - Universal Background Model (GMM-UBM) approach was first introduced in [Reynolds et al., 2000], which is a different approach to doing speaker verification. As mentioned before, GMMs are useful for classifications, but they are also good for creating complex models that cannot be easily modeled after a closed form solution. The GMM-UBM model takes advantage of that.

The UBM is built on a large dataset containing the background (or world) model. In other words, it contains data about the not-the-target, and this largely comprises the other speakers. A very large GMM is trained on it, and then the final UBM is obtained. Then the speaker model is created with a GMM such that it is adapted based on the UBM model. The illustration on Figure **??** shows the UBM in blue and the adapted speaker model in yellow (I'm colorblind, so I'm uncertain if this is correct). Once the UBM has been formed, the speaker model may be derived from the UBM via a maximum a posteriori adaptation.

Maximum A Posteriori Adaptation of the UBM

Assume a GMM-UBM model has been created and is represented by the following equation

$$f(\tilde{\boldsymbol{x}}_n|\Lambda) = \sum_{g=1}^{M} \pi_g \mathcal{N}(\tilde{\boldsymbol{x}}_n|\boldsymbol{mu}_g, \Sigma_g)$$

where $\tilde{\boldsymbol{x}}$ represents the n^{th} feature vector used for the UBM, π_g represents the weight of the g^{th} mixture component out of M Gaussian components, with a mean vector $\boldsymbol{\mu}_g$ and covariance matrix Σ_g. The set of parameters for the GMM-UBM model is denoted as $\Lambda = \{\pi_g, \mu_g, \Sigma_g | 1 \leq g \leq M\}$. Next let $X \in \{\boldsymbol{x}_n | 1 \leq n \leq T\}$ represent the set of acoustic feature vectors by a speaker s, the first probability values γ_n to be computed are

$$\gamma_n(g) = Pr(g|\boldsymbol{x}_n) = \frac{\pi_g Pr(\boldsymbol{x}_n|g, \lambda_0)}{\sum_g \pi_g Pr(\boldsymbol{x}_n|g, \lambda_0)}$$

These $\gamma_n(g)$ values are then used to calculate the 0th, 1st, and 2nd order Baum-Welch Statistics, shown respectively below, which represent the sufficient statistics for the weight, mean, and covariance parameters.

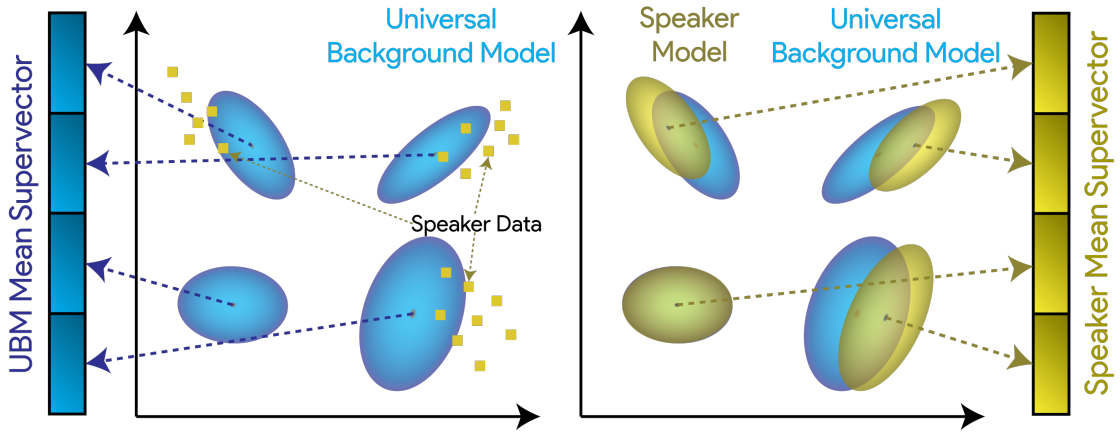

Figure 1.11: An illustration of (left) the GMM-UBM models with 4 components, which lead to the UBM supervector, and (right) the speaker adapted GMM based on the UBM model and the speaker data via a MAP algorithm to create the speaker supervector.

$$BW_s^0(g) = \sum_{n=1}^{T} \gamma_n(g)$$
$$BW_s^1(g) = \sum_{n=1}^{T} \gamma_n(g)\boldsymbol{x}_n$$
$$BW_s^2(g) = \sum_{n=1}^{T} \gamma_n(g)\boldsymbol{x}_n\boldsymbol{x}_n^T$$

Once computed, the posterior mean and covariance matrix of the features may be computed, given the data vectors X, shown below respectively.

$$\mathbb{E}_g[\boldsymbol{x}_n|X] = \frac{BW_s^1(g)}{BW_s^0(g)}$$
$$\mathbb{E}_g[\boldsymbol{x}_n\boldsymbol{x}^T|X] = \frac{BW_s^2(g)}{BW_s^0(g)}$$

Then maximum a posteriori adaptation rule equations for the weight, mean and covariance, which are used for the speaker verification are computed as follow, assuming the following definition of α_g

$$\begin{cases} \alpha_g = \dfrac{BW_s^0(g)}{BW_s^0(g) + r} \\ \hat{\pi}_g = \beta\left[\alpha_g \dfrac{BW_s^0(g)}{T} + (1-\alpha_g)\pi_g\right] \\ \hat{\boldsymbol{\mu}}_g = \alpha_g\mathbb{E}_g[\boldsymbol{x}_n|X] + (1-\alpha_g)\boldsymbol{mu}_g \\ \hat{\Sigma}_g = \alpha_g\mathbb{E}_g[\boldsymbol{x}_n\boldsymbol{x}_n^T|X] + (1-\alpha_g)(\Sigma_g + \boldsymbol{\mu}_g\boldsymbol{\mu}_g^T) - \hat{\mu}_g\hat{\mu}_g^T \end{cases}$$

The scaling factor β is based on all of the $\hat{\pi}_g$ ensuring that they all add up to 1, and the relevance factor r is a control parameter on how the adapted GMM will be affected by the observed data. It turns out that the only effective parameter are the mean vectors $\hat{\boldsymbol{\mu}}_g$.

Supervectors was first coined in [Kuhn et al., 1998], where the mean vectors $\boldsymbol{\mu}_g$ are concatenated together to create a single large vector. Therefore, for an M component Gaussian mixture with F features, the supervector would live on the space $\mathbb{R}^{MF\times 1}$. Figure 1.11 shows the UBM created over the background dataset, which is used

along with the speaker data to adapt it onto a speaker mean supervector. Therefore, since the models shown in Figure 1.11 have $M = 4$ components, and it is in a 2D graph (i.e. $F = 2$), then the supervector for both the UBM and the speaker would live in the space $\mathbb{R}^{MF\times 1} = \mathbb{R}^{8\times 1}$, and such vectors are suitable for SVM-based speaker recognition.

One great advantage of using the UBM model is that it has a very nice performance, even if the speaker-dependent data is small; however, the disadvantage is that it requires a gender-balanced large speakers set in order to train it efficiently [Tiwari and Lotia, 2012]. There has been a number of ways to represent the GMM supervector for a speaker s, but the generalized way to do so is to write

$$\boldsymbol{M}(s) = \boldsymbol{m}_0 + \boldsymbol{m}_{speaker} + \boldsymbol{m}_{channel} + \boldsymbol{m}_{residual}$$

where the $\boldsymbol{m}_0$ represents a speaker/channel/environment independent component, $\boldsymbol{m}_{speaker}$ represents a speaker-dependent component, $\boldsymbol{m}_{channel}$ represents a channel-dependent component, and $\boldsymbol{m}_{residual}$ represents a residual vector.

1.7.2 Eigenvoices and Eigenchannel

The concept of eigenvoices is drawn directly from principle component analysis (PCA), where it is solely a speaker-dependent mean supervector, with, of course, a speaker-independent supervector obtained from the UBM model, where $\boldsymbol{m}_{speaker} = \boldsymbol{V}y_s$.

$$\boldsymbol{M}(s) = \boldsymbol{m}_0 + \boldsymbol{V}y_s$$

The matrix $\boldsymbol{V}$ spans the speaker subspace, and $\boldsymbol{y}_s$ are the speaker factors. $\boldsymbol{V}$ then represents the eigenvoice matrix, and $\boldsymbol{y}_s$ represents the weight vector. The issue is that this model assumes that all of the speaker supervectors are completely contained in the eigenvoice subspace. Therefore, this model may be extended to the expression

$$\boldsymbol{M}(s) = \boldsymbol{m}_0 + \boldsymbol{V}y_s + \boldsymbol{D}\boldsymbol{z}_s$$

where the matrix $\boldsymbol{D}$ is a diagonal matrix and $\boldsymbol{z}_s$ is a normal random vector, which are used to make the residual term $\boldsymbol{m}_{residual} = \boldsymbol{D}\boldsymbol{z}_s$. A similar model may be created, but that is dependent on the channel factors instead, where $\boldsymbol{U}$ is a low-rank matrix spanning the channel subspace, and $\boldsymbol{c}_h \sim \mathcal{N}(0, \boldsymbol{I})$ is a random vector for an utterance h, making $\boldsymbol{m}_{channel} = \boldsymbol{U}c_h$

$$\boldsymbol{M}(s) = \boldsymbol{m}_0 + \boldsymbol{U}c_h + \boldsymbol{D}\boldsymbol{z}_s$$

1.7.3 Joint Factor Analysis

The joint factor analysis was formulated by [Kenny, 2005], where they bring the eigenvoice and eigenchannel model together into a single model. The following equation follows the same notation

$$\boldsymbol{M}(s) = \boldsymbol{m}_0 + \boldsymbol{V}y_s + \boldsymbol{U}\boldsymbol{c}_h + \boldsymbol{D}\boldsymbol{z}_s$$

In order to obtain the JFA matrices and the factor vectors, it is required to follow the following order:

1. Train the eigenvoice matrix $\boldsymbol{V}$ assuming that the eigenchannel matrix $\boldsymbol{U}$ and the residual matrix $\boldsymbol{D}$ are zero

2. Train the eigenchannel matrix $\boldsymbol{U}$ given the estimated $\boldsymbol{V}$, and still assuming $\boldsymbol{D}$ is zero.

3. Train the residual matrix $\boldsymbol{D}$ given the estimated $\boldsymbol{V}$ and $\boldsymbol{U}$.

4. Using the estimated matrices, compute the factor vectors $\boldsymbol{y}_s$, $\boldsymbol{c}_h$, and $\boldsymbol{z}_s$

Training the $\boldsymbol{V}$ matrix

Step 1. To train the $\boldsymbol{V}$ matrix, first compute the zeroth, first, and second order sufficient statistics

$$\gamma_n(g) = Pr(g|\boldsymbol{x}_n) = \frac{\pi_g Pr(\boldsymbol{x}_n|g,\lambda_0)}{\sum_g \pi_g Pr(\boldsymbol{x}_n|g,\lambda_0)}$$

$$BW_s^0(g) = \sum_{n=1}^{T} \gamma_n(g)$$

$$BW_s^1(g) = \sum_{n=1}^{T} \gamma_n(g)\boldsymbol{x}_n$$

$$BW_s^2(g) = diag\left(\sum_{n=1}^{T} \gamma_n(g)\boldsymbol{x}_n\boldsymbol{x}_n^T\right)$$

Step 2. Center the first and second order statistics

$$\hat{BW}_s^1(g) = BW_s^1 - BW_s^0\boldsymbol{\mu}_g$$
$$\hat{BW}_s^2(g) = BW_s^2 - diag(BW_s^0\boldsymbol{\mu}^T + \boldsymbol{\mu}(BW_s^1)^T - BW_s^0\boldsymbol{\mu}_g\boldsymbol{\mu}_g^T)$$

Step 3. One can then expand the statistics onto matrices

$$\boldsymbol{BW_s^0} = \begin{bmatrix} BW_s^0(1)\cdot\boldsymbol{I} & & \\ & \ddots & \\ & & BW_s^0(M)\cdot\boldsymbol{I} \end{bmatrix}$$

$$\boldsymbol{BW_s^1} = \begin{bmatrix} \hat{BW}_s^1(1) \\ \vdots \\ \hat{BW}_s^1(M) \end{bmatrix}$$

$$\boldsymbol{BW_s^2} = \begin{bmatrix} \hat{BW}_s^2(1)\cdot\boldsymbol{I} & & \\ & \ddots & \\ & & \hat{BW}_s^2(M)\cdot\boldsymbol{I} \end{bmatrix}$$

Step 4. Make an estimate of the speaker factor $\boldsymbol{y}_s$

$$v_s = I + V^T\cdot\Sigma^{-1}\cdot\boldsymbol{BW_s^0}\cdot V$$
$$\boldsymbol{y}_s \sim \mathcal{N}(v_s^{-1}\cdot V^T\cdot\Sigma^{-1}\cdot\boldsymbol{BW_s^1}, v_s^{-1})$$
$$\rightarrow \mathbb{E}[\boldsymbol{y}_s] = v_s^{-1}\cdot V^T\cdot\Sigma^{-1}\cdot\boldsymbol{BW_s^1}$$

Step 5. Compute statistics across the speakers

$$BW^0(g) = \sum_s BW_s^0$$
$$A(g) = \sum_s BM_s^0 v_s^{-1}$$
$$\mathbb{C} = \sum_s \boldsymbol{BW_s^1}\cdot\mathbb{E}[\boldsymbol{y}_s]^T$$
$$\boldsymbol{BW^0} = \sum_s \boldsymbol{BW_s^0}$$

It is possible to split $\mathbb{C}$ into

$$\mathbb{C} = \begin{bmatrix} \mathbb{C}_1 \\ \vdots \\ \mathbb{C}_M \end{bmatrix}$$

Step 6. Estimate $\boldsymbol{V}$

$$\boldsymbol{V} = \begin{bmatrix} V_1 \\ \vdots \\ V_M \end{bmatrix} = \begin{bmatrix} A^{-1}(1) \cdot \mathbb{C}_1 \\ \vdots \\ A^{-1}(M) \cdot \mathbb{C}_M \end{bmatrix}$$

Step 7. Compute the covariance update

$$\Sigma = (\boldsymbol{BW^0})^{-1}(\sum_s \boldsymbol{BW_s^2} - diag(\mathbb{C} \cdot V^T))$$

Step 8. Repeat steps 4-7

Training the U matrix

Because the eigenchannel matrix analyzes over the channel factors, it involves the changes between utterances.
Step 1. Compute the the 0th and 1st order statistics of each conversation of each speaker

$$N_{s,conv}(g) = \sum_{n \in conv,s} \gamma_n(g)$$
$$F_{s,conv}(g) = \sum_{n \in conv,s} \gamma_n(g)\boldsymbol{x}_n$$

Step 2. For each speaker s, compute a speaker shift using $\boldsymbol{V}$ and the speaker factors $\boldsymbol{y}_s$.

$$spkshift_s = \boldsymbol{m}_0 + \boldsymbol{V}\boldsymbol{y}_s$$

Step 3. Compute a speaker shifted version of the first order statistics given a Gaussian posterior weighing

$$\hat{F}_{s,conv}(g) = F_{s,conv}(g) - spkshift_s \cdot N_{s,conv}(g)$$

Step 4. Expand the statistics to matrices.

$$NN_{s,conv} = \begin{bmatrix} N_{s,conv}(1) \cdot \boldsymbol{I} & & \\ & \ddots & \\ & & N_{s,conv}(M) \cdot \boldsymbol{I} \end{bmatrix}$$
$$FF_{s,conv} = \begin{bmatrix} \hat{F}_{s,conv}(1) \\ \vdots \\ \hat{F}_{s,conv}(M) \end{bmatrix}$$

Step 5. Use the same methodology of training $\boldsymbol{V}$ and $\boldsymbol{y}_s$ to train $\boldsymbol{U}$ and $\boldsymbol{c}$ by using $NN_{s,conv}$ and $FF_{s,conv}$, and iterate.

Training the D matrix

Finally, to compute the residual matrix,
Step 1. For each speaker s, compute the speaker shift using $\boldsymbol{V}$ and the speaker factors $\boldsymbol{y}_s$

$$spkshift_s = \boldsymbol{m}_0 + \boldsymbol{V}\boldsymbol{y}_s$$

Step 2. For each conversation side *conv* of the speaker s, compute the channel shift using $\boldsymbol{U}$ and $\boldsymbol{z}$

$$channelshift_{s,conv} = \boldsymbol{U}\boldsymbol{c}_{s,conv}$$

Step 3. For each speaker that is being used for the JFA training, subtract the Gaussian posterior weighed speaker shift and the channel shifts from the first order statistics.

$$\hat{F}_(g) = F_s(g) - spkshift_s \cdot N_s(g) - \sum_{conv \in s} channelshift_{s,conv} \cdot N_{s,conv}(g)$$

Step 4. Expand the statistics to matrices.

$$NN_{s,conv} = \begin{bmatrix} N_{s,conv}(1) \cdot \boldsymbol{I} & & \\ & \ddots & \\ & & N_{s,conv}(M) \cdot \boldsymbol{I} \end{bmatrix}$$

$$FF_{s,conv} = \begin{bmatrix} \hat{F}_{s,conv}(1) \\ \vdots \\ \hat{F}_{s,conv}(M) \end{bmatrix}$$

Step 5. Estimate the residual factors $\boldsymbol{z}$

$$\begin{aligned} d_s &= \boldsymbol{I} + \boldsymbol{D}^2 \cdot \Sigma^{-1} \cdot NN_s \\ \boldsymbol{z}_s &\sim \mathcal{N}(d_s^{-1} \cdot D \cdot \Sigma^{-1} \cdot FF_s, d_s^{-1}) \\ \mathbb{E}[\boldsymbol{z}_s] &= d_s^{-1} \cdot \boldsymbol{D} \cdot \Sigma^{-1} \cdot FF_s \end{aligned}$$

Step 6. Obtain more statistics across the speakers

$$\begin{aligned} N(g) &= \sum_s N_s(g) \\ a &= \sum_s diag(NN_s \cdot d_s^{-1}) \\ b &= \sum_s diag(FF_s \cdot \mathbb{E}[\boldsymbol{z}_s]) \\ NN &= \sum_s NN_s \end{aligned}$$

Step 7. Compute the $\boldsymbol{D}$ estimate

$$\boldsymbol{D} = \begin{bmatrix} D_1 \\ \vdots \\ D_M \end{bmatrix} = \begin{bmatrix} a^{-1}(1) \cdot b_1 \\ \vdots \\ a^{-1}(M) \cdot b_M \end{bmatrix} \text{ where } b = \begin{bmatrix} b_1 \\ \vdots \\ b_M \end{bmatrix}$$

Step 8. Iterate steps 5-7

1.7.4 *i*-vectors

The identity-vectors (i-vectors) were created by [Dehak et al., 2011], which follows the works on JFA. The methods for JFA were very promising as they were shown to contain information about speakers, and to work very well in other speech tasks such as language recognition, meaning that it had the power to group different languages. A recent work has shown that i-vectors are able to identify the native language of a speaker speaking a second language, indicating that they are capable of capturing information about the accents of an individual [Senoussaoui et al., 2016]. The motivation was that experiments showed that channel factors also contain speaker-dependent information. Therefore, in the i-vector methodology, speaker and channel factors were combined into a single matrix called the total variability matrix $\boldsymbol{T}$, which works in conjunction with the i-vector w_s. Therefore the speaker and session dependent GMM can be represented by

$$\boldsymbol{M}_s = \boldsymbol{m}_0 + \boldsymbol{T}\boldsymbol{w}_s$$

The hidden variables in the i-vector are called the total factors, such that $\boldsymbol{w}_s \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Although the hidden variables are not observable, they can be estimated via their posterior expectation. In many sense, this methodology is very similar to the PCA model, and the $\boldsymbol{T}$ matrix is trained using the same algorithms as for the JFA model, however each utterance is treated as if they were obtained from a different speaker.

In order to obtain the i-vectors:

1. Run the exact training procedure used to train the $\boldsymbol{V}$, while treating the conversations of all training speakers as to belong to different speakers.
2. With a given $\boldsymbol{T}$, compute i-vectors for each conversation side.
3. For the channel compensation, perform a linear discriminant analysis, followed by a within-class covariance normalization
4. Perform a cosine distance scoring o the channel-compensated i-vectors for a pair of conversation sides

1.8 Features from Speech

Appendices

Appendix A

Methodologies

A.1 Long Short-Term Memory and Gated Recurrent Unit

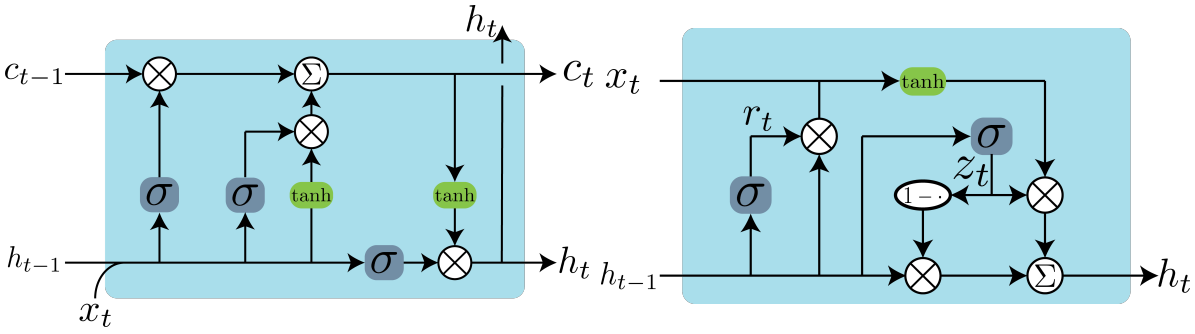

Figure A.1: A visual representation of the LSTM cell and the GRU cell

The LSTM and the GRU cell units are shown in Figure A.1, where σ depicts the sigmoid function. The equations that govern the LSTM are

$$\begin{cases} \begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \\ c_t f \cdot c_{t-1} + i \cdot g \\ h_t = o \cdot \tanh(c_t) \end{cases}$$

The equations that govern the GRU are

$$\begin{cases} \text{Update Gate:} & z_t = \sigma(W_z[h_{t-1}, x_t]) \\ \text{Reset Gate:} & r_t = \sigma(W_r[h_{t-1}, x_t]) \\ \text{Current Memory Content:} & \tilde{h}_t = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t]) \\ \text{Final Memory:} & h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \end{cases}$$

Although sigmoid functions are often frowned upon in the deep learning community due to the inevitable fact that often times, the gradients become zero at very high or low values, the sigmoids behave as switch knobs for the acceptance or rejection of previous information on the cell memory c_t or on the hidden states h_t, which are coming from previous iterations. The vanishing gradient problem is avoided in LSTM and GRU or any network with forget gates through the concept of linear carousel.

A.2 Gaussian Mixture Models

A.3 Spectral Clustering

A.4 Mel-Frequency Cepstrum Coefficients

A.5 Linear Prediction Coefficients and its Derivatives

Bibliography

[Anttila et al., 1995] Anttila, P., Paatero, P., Tapper, U., and Järvinen, O. (1995). Source identification of bulk wet deposition in finland by positive matrix factorization. *Atmospheric Environment*, 29(14):1705 – 1718.

[Barak, 1988] Barak, A. P. (1988). Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269.

[Cleeremans et al., 1989] Cleeremans, A., Servan-Schreiber, D., and McClelland, J. L. (1989). Finite-state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381.

[Dehak et al., 2011] Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P., and Ouellet, P. (2011). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798.

[Duda and Hart, 1973] Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Willey & Sons, New Yotk.

[Eggert and Korner, 2004] Eggert, J. and Korner, E. (2004). Sparse coding and nmf. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 4, pages 2529–2533 vol.4.

[Figueiredo and Jain, 2002] Figueiredo, M. A. T. and Jain, A. K. (2002). Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396.

[Forney, 1973] Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

[Fowler, 2018] Fowler, G. (2018). I live with alexa, google assistant and siri. here's which one you should pick. *The Washington Post: The Switch Review*.

[Gers et al., 2000] Gers, F. A., Schmidhuber, J., and Cummins, F. A. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471.

[Graves and Jaitly, 2014] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1764–II–1772. JMLR.org.

[Graves et al., 2013] Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778.

[Han et al., 2018] Han, K. J., Chandrashekaran, A., Kim, J., and Lane, I. R. (2018). The CAPIO 2017 conversational speech recognition system. *CoRR*, abs/1801.00059.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Jean-Marc et al., 2017] Jean-Marc, Xiph.Org, and Mozilla (2017). Rrnoise.

[Jordan, 1986] Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist seuential machine. *Cognitive Science Conference*, pages 531–546.

[Kenny, 2005] Kenny, P. (2005). Joint factor analysis of speaker and session variability: Theory and algorithms. Technical report.

[Kuhn et al., 1998] Kuhn, R., Nguyen, P., Junqua, J.-C., Goldwasser, L., Niedzielski, N., Fincke, S., and Contolini, M. (1998). Eigenvoices for speaker adaptation. In *ICSLP 1998, 5th International Conference on Spoken Language Processing, 30 November-4 December 1998, Sydney, Australia*, Sydney, AUSTRALIA.

[Lee and Seung, 1999] Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791.

[Lee and Seung, 2000] Lee, D. D. and Seung, H. S. (2000). Algorithms for non-negative matrix factorization. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 535–541, Cambridge, MA, USA. MIT Press.

[Li and Xiao,] Li, P. and Xiao, Y. Matrix factorization for speech enhancement.

[Moskvitch, 2017] Moskvitch, K. (2017). The machines that learned to listen. *BBC Future*.

[Ng et al., 2001] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press.

[Paatero and Tapper, 1994] Paatero, P. and Tapper, U. (1994). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Rabiner and Juang, 1986] Rabiner, L. R. and Juang, B.-H. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*, 3:4–16.

[Reynolds et al., 2000] Reynolds, D. A., Quatieri, T. F., and Dunn, R. B. (2000). Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10(1):19 – 41.

[Schmidt and Olsson, 2006] Schmidt, M. N. and Olsson, R. K. (2006). Single-channel speech separation using sparse non-negative matrix factorization. In *INTERSPEECH*.

[Senoussaoui et al., 2016] Senoussaoui, M., Cardinal, P., Dehak, N., and Koerich, A. L. (2016). Native language detection using the i-vector framework. In *INTERSPEECH*.

[Tiwari and Lotia, 2012] Tiwari, M. S. and Lotia, P. (2012). Speaker verification system using gaussian mixture model and ubm. *International Journal of Digital Application and Contemporary Research*, 1(3).

[VITERBI, 1967] VITERBI, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269.

[Zhang et al., 2016] Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Laurent, C., Bengio, Y., and Courville, A. (2016). Towards end-to-end speech recognition with deep convolutional neural networks. In *Interspeech 2016*, pages 410–414.