# Machine Learning Qualifying Exam Report

*Georgia Institute of Technology*

**Ph.D. Student**
*Nicolas Shu*

**Committee Members:**
*David Anderson*
*Gari Clifford (Advisor)*
*Evangelos Theodorou*

# Contents

## 0.1 Key Papers for the Report

- [Lee and Seung, 2000]
- [Eggert and Korner, 2004]
- [Kenny, 2005]
- [Schmidt and Olsson, 2006]
- [Li and Xiao, ]
- [Reynolds et al., 2000]
- [Hansen and Hasan, 2015]
- [E et al., 2018]
- [Shwartz-Ziv and Tishby, 2017]
- [Goldt and Seifert, 2017]
- [Chaudhari et al., 2018]

# 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

One public corpus developed by Dr. Clifford **??**, also referred to as MIMIC II, where over 900 pages of nursing notes had been deidentified such that they wouldn't be considered protected health information anymore. Three text-to-speech engines (Amazon Polly, Google Text-to-Speech, IBM Cloud Text-to-Speech) were used to convert 200 sentences from the de-identified text to synthesized speech, and two speech-to-text engines (Google Speech-to-Text, IBM Cloud Speech-to-Text) were used to convert them back. By using the Levenshtein normalized string distance Figure 1.1 (left) and Levenshtein normalized string similarity Figure 1.1 (right) were used to compare the efficiency of every combination of them.

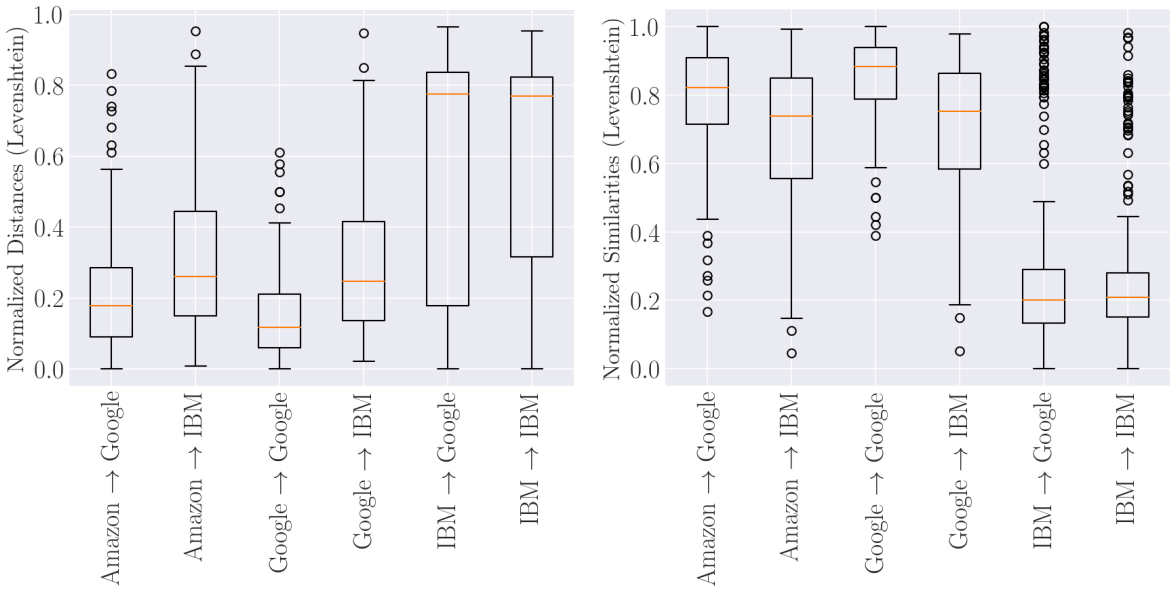


Figure 1.1: An illustration of the schematic of the automatic speech recognition system

By analyzing Figure 1.1, it is clear that the Google Text-to-Speech in combination with the Google Speech-to-Text has the highest performance. It is rather impressive that the combination of IBM Cloud Text-to-Speech and IBM Cloud Speech-to-Text performed so poorly. Although the Google-Google combination has the highest performance, there still is a large spread down to lower normalized string similarities, and for a medical transcription, it is paramount that it has extremely high accuracies for logical reasons. In other words, even some of the better engines need improvement, and itis very possible that they do not perform well under noisy environments.

### 1.2.2 Grade Reading Level of Various Texts

It is important to also evaluate the reading grade level of the corpus in comparison to other works of literature in order to gain some insight on how complex does the corpus compare to others.

| Scoring Method | Corpora | | | |
|---|---|---|---|---|
| | **DeID** | **Lord of The Rings: Fellowship of the Ring** | **Moby Dick** | **20000 Leagues Under the Sea** |
| **Flesch Reading Ease Score** | 54.3<br>Fairly difficult | 54.8<br>Fairly difficult | 67.4<br>Standard/Average | 56<br>Fairly difficult |
| **Gunning Fog** | 10<br>Fairly easy | 15.7<br>Difficult | 11.5<br>Hard | 13.1<br>Hard |
| **Flesch-Kincaid Grade Level** | 8.7<br>9th Grade | 13.7<br>College | 9.2<br>9th Grade | 10.6<br>11th Grade |
| **Coleman-Liau Index** | 10<br>10th Grade | 8<br>8th Grade | 8<br>8th Grade | 10<br>10th Grade |
| **SMOG Index** | 8.7<br>9th Grade | 9.7<br>10th Grade | 8<br>8th Grade | 10.1<br>10th Grade |
| **Automated Readability Index** | 7.2<br>11-13 years old | 15.1<br>College Graduate | 9.7<br>14-15 years old | 11<br>15-17 years old |
| **Linsear Write Formula** | 7.3<br>7th Grade | 18.9<br>College Graduate | 12.6<br>College | 13.4<br>College |

## 1.3 Automatic Event Recognition in a Noisy Clinical 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.

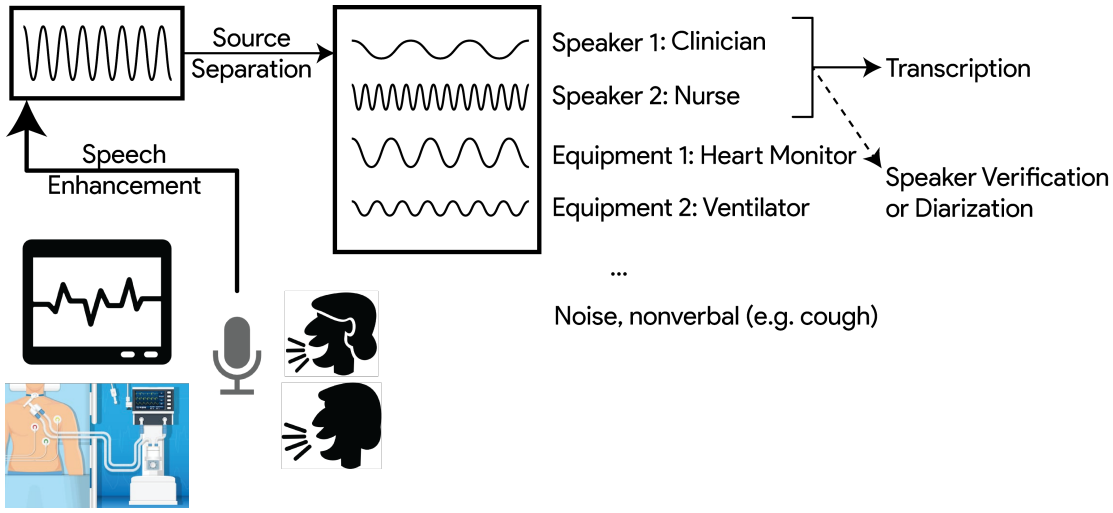


Figure 1.2: An illustration of the schematic of the automatic speech recognition system

As one can see from Figure 1.2, the ASR would involve a microphone or an array of microphones which would go under speech enhancement and then go under a blind source separation, as there could be multiple different sources, such as different people speaking, a heart rate monitor, background music, a ventilator, etc. After the source separation, a transcription engine would transcribe the audio of the selected speakers, which could be filtered through a speaker verification/diarization system.

### 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 1.3

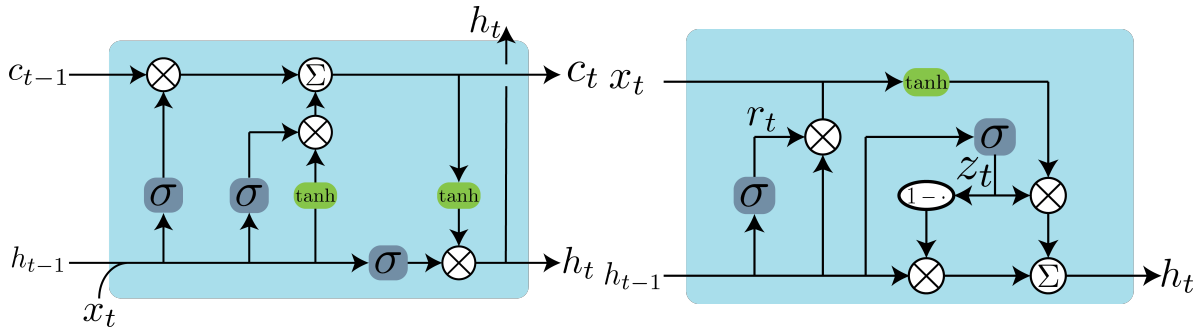


Figure 1.3: A visual representation of the LSTM cell (Left) and the GRU cell (Right). The LSTM contains three representations of its states: the cell state/memory $c_t$, the hidden states $h_t$, and the input state $x_t$, all of which occurs at time $t$. On the GRU, there are the input state $x_t$, the hidden state $h_t$, the update gate $z_t$ and the reset gate $r_t$. The $\sigma$ 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 1.4 is the topology of the RRNoise architecture, where it outputs voice activity detection (VAD) as well as the gains from the input features.

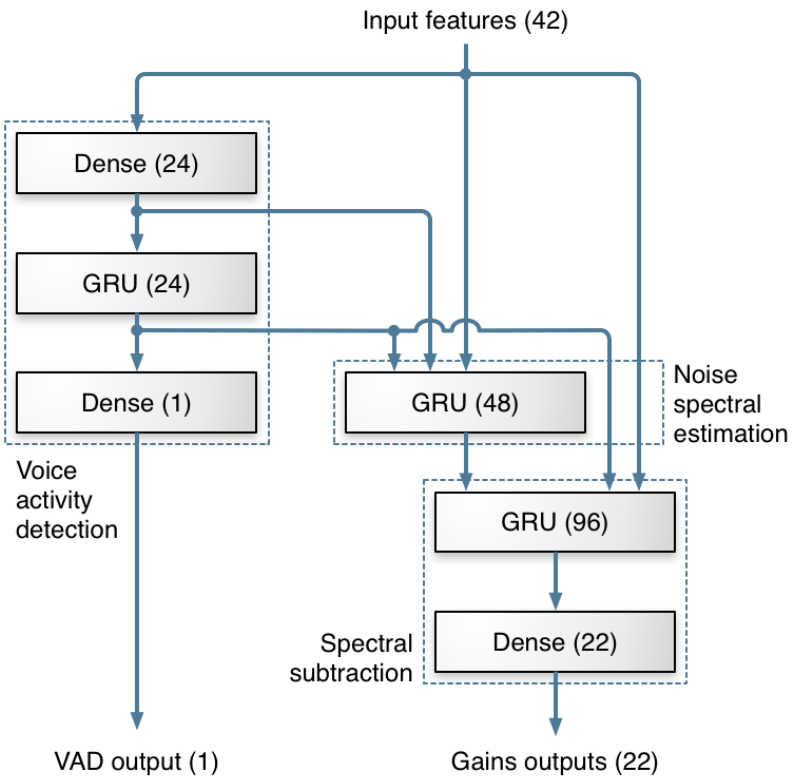


Figure 1.4: The topology of the RRNoise network used to perform noise reduction. This figure is taken from [Jean-Marc et al., 2017]

### 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

The first technique is the independent component analysis (ICA), in which it tries to solve the Cocktail Party Problem via a superposition of $J$ voices when being recorded by $N$ microphones (thus yielding $N$ recordings). For a discrete set of $M$ samples, there is a source matrix $\boldsymbol{Z} \in \mathbb{R}^{J \times M}$, with a data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times M}$. There should then exist a mixing matrix $\boldsymbol{A} \in \mathbb{R}^{N \times J}$, such that

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{Z}^T$$
$$\mathbb{R}^{M\times N} = \mathbb{R}^{N\times J}\mathbb{R}^{J\times M}$$

The goal of ICA is to then find a demixing matrix $\boldsymbol{W}$ such that

$$\boldsymbol{W} \approx \boldsymbol{A}^{-1}$$

In practice, to perform such separation is by minimizing cost functions that are represented by mutual information entropy, or kurtosis. Those are chosen because in ICA, one must find a demixing matrix $\boldsymbol{W}$ that maximizing non-Gaussianity of each source.

### Kurtosis

Kurtosis is the fourth statistical moment of a distribution which measures the relative peakness of a distribution with respect to a Gaussian. When the kurtosis is greater than 3, it is defined as *leptokurtic* (i.e. super-Gaussian); when it is less than 3, it is *platykurtic* (i.e. sub-Gaussian), and Gaussian distributionas are termed as *mesokurtic*. The definition of the kurtosis $\kappa$ and the empirical kurtosis $\hat{\kappa}$ are defined as

$$\kappa = \frac{\mathbb{E}[(x-\mu_x)^2]}{\sigma^4}$$
$$\hat{\kappa} = \frac{1}{M}\sum_{k=1}^{M}\left[\frac{x_i - \hat{\mu}_x}{\hat{\sigma}}\right]^4$$

Unfortuantely, disproportionate changes to the distribution tails cause large changes on the kurtosis.

### Negentropy

Negentropy is an outlier insensitive method to estimate the fourth moment. We know that the entropy $H(y)$ of a random variable $y_i$ with a probability distribution $Pr(y_i)$ is

$$H(y) = -\sum_i Pr(y_i)\log_2(Pr(y_i))$$

This can be expanded to a differential entropy of a random vector $\boldsymbol{y}$ as

$$H(\boldsymbol{y}) = -\int Pr(\boldsymbol{y})\log_2[Pr(\boldsymbol{y})]d\boldsymbol{y}$$

One interesting outcome from information theory is that a Gaussian variable has the largest amount of entropy between any other random variable with equal variance. Therefore, the negentropy $\mathcal{J}$ is defined as

$$\mathcal{J}(\boldsymbol{y}) = H(\boldsymbol{y}_G) - H(\boldsymbol{y})$$

where $\boldsymbol{y}_G$ represents a Gaussian random vector with the same covariance as $\boldsymbol{y}$, where $H(\boldsymbol{y}) \geq H(\boldsymbol{y}_G)$. There are a number of ways to compute the negentropy. One way is to use kurtosis, but it yields the same errors as previously mensioned

$$\mathcal{J}(y) \approx \frac{\mathbb{E}^2[y^3]}{12} + \frac{\kappa(\boldsymbol{y})^2}{48}$$

Another way to compute negentropy is as follows:

$$\begin{cases} \mathcal{J}(y) \approx \mathbb{E}[g(\boldsymbol{y})] - \mathbb{E}[g(\theta)] \\ g = \text{non-quadratic function s.t. it results always a non-negative result} = \dfrac{\ln(\cosh(\alpha y))}{\alpha} \text{ or } -\exp\left(-\dfrac{y^2}{2}\right) \end{cases}$$

Thus one tries to maximize the $\mathcal{J}$ via a gradient ascent

#### Ambiguities

There are some ambiguities that are important to know in regards to the ICA solutions. The first one is permutation. Let $P \in \mathbb{R}^{n \times n}$ be any permutation matrix. E.g.

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Thus, if $\overrightarrow{z}$ is a vector, $P\overrightarrow{z}$ is a permuted $\overrightarrow{z}$. So if one only has an observation $x \in \mathbb{R}^n$, then there's no way to distinguish between the unmixing matrix $W$ and $PW$. Therefore, the permutation of the original sources is ambiguous.

The second ambiguity is scaling, such that,

$$\begin{aligned} x &= As \\ &= (\alpha A)\left(\frac{1}{\alpha}s\right) \\ &= (2A)\left(\frac{1}{2}s\right) \\ &= (3A)\left(\frac{1}{3}s\right) \end{aligned}$$

Therefore, one cannot recover the correct scaling of the sources.

### 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

$$\boldsymbol{Y} = \boldsymbol{DH}$$

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 1.5, where the figure below was inspired on the works from [Schmidt and Olsson, 2006].

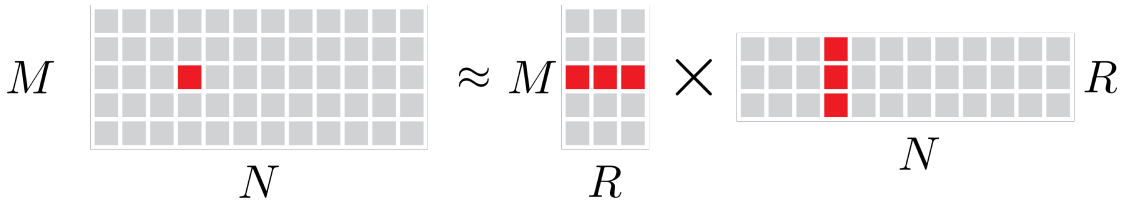


Figure 1.5: A very simple schematic of a non-negative matrix factorization, where the matrix on the left represents the mixed signal, the matrix on the center represents the dictionary matrix, and the matrix on the right represents the coding matrix

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

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

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

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

4.Repeat 2,3 until stability

If one performs an NMF, a visual representation is shown on Figure 1.6, where the figure below was inspired on the works from [Schmidt and Olsson, 2006].

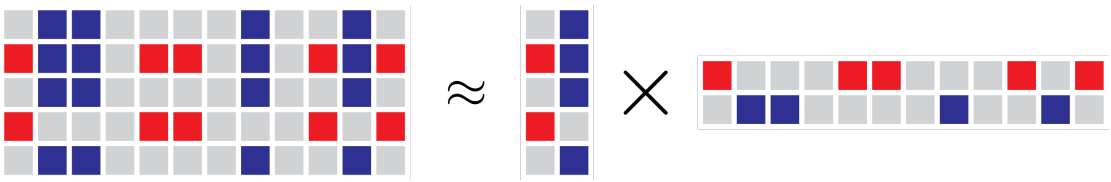

Figure 1.6: A visual example of an NMF result, with two sources being separated.

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 $\lambda$ is the parameter to control the degree of sparsity. To solve this optimization problem, if one was to set $\boldsymbol{R} = \boldsymbol{D}\boldsymbol{H}$, 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 1.7 shows an illustration on how to create the dictionary.

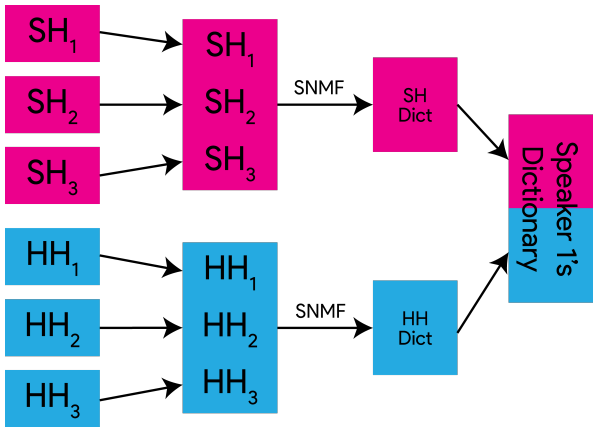


Figure 1.7: An illustration on how to create a speaker's dictionary $\boldsymbol{D}_i$

### 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 1.8, 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.2), 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.

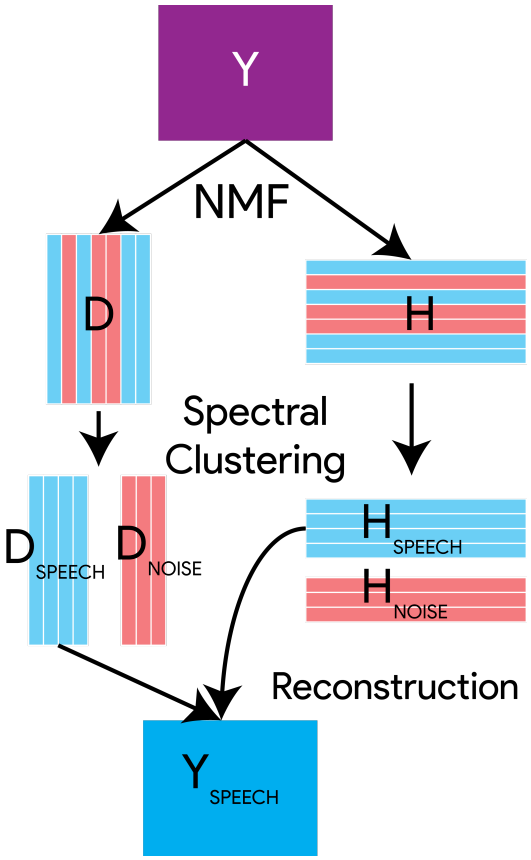


Figure 1.8: An illustration on how to use non-negative matrix factorization to denoise a signal, where $\boldsymbol{Y}$ represents the noisy signal, $\boldsymbol{D}$ represents the dictionary matrix obtained through NMF, and $\boldsymbol{H}$ represents the code matrix, also attained through the NMF.

## 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]. 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.2) 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.

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.

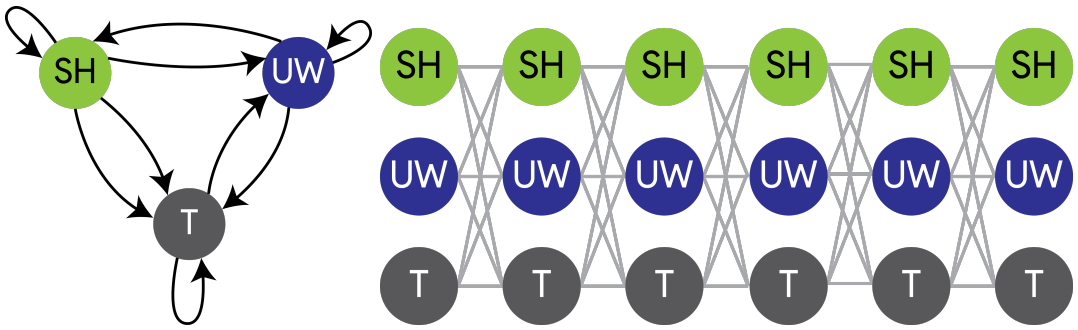


Figure 1.9: A simple HMM with the states being the phonemes (shown as ARPABET). The image on the left shows the HMM in its compact form, and the image on the right shows the HMM unrolled over time.

As an example, Figure 1.9 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. 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 1.10 as a black line.

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 1.11. 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].

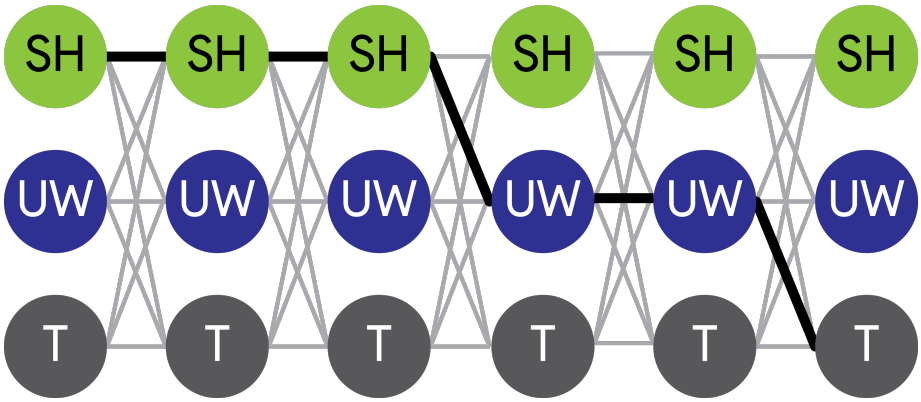


Figure 1.10: The theoretical Viterbi algorithm solution of a signal.

The logic behind utilizing CNNs is the fact that a neuron would be a windowing function and classifier, which captures the content/interpretation of a signal as it slides over time, and thus it activates its successive neurons. Recurrent neural networks and LSTMs are very fitting architectures for these time series data because they are able to iteratively move through windows of time, and are also able to iteratively output results, such as phonemes, letters, or words.

## 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

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.

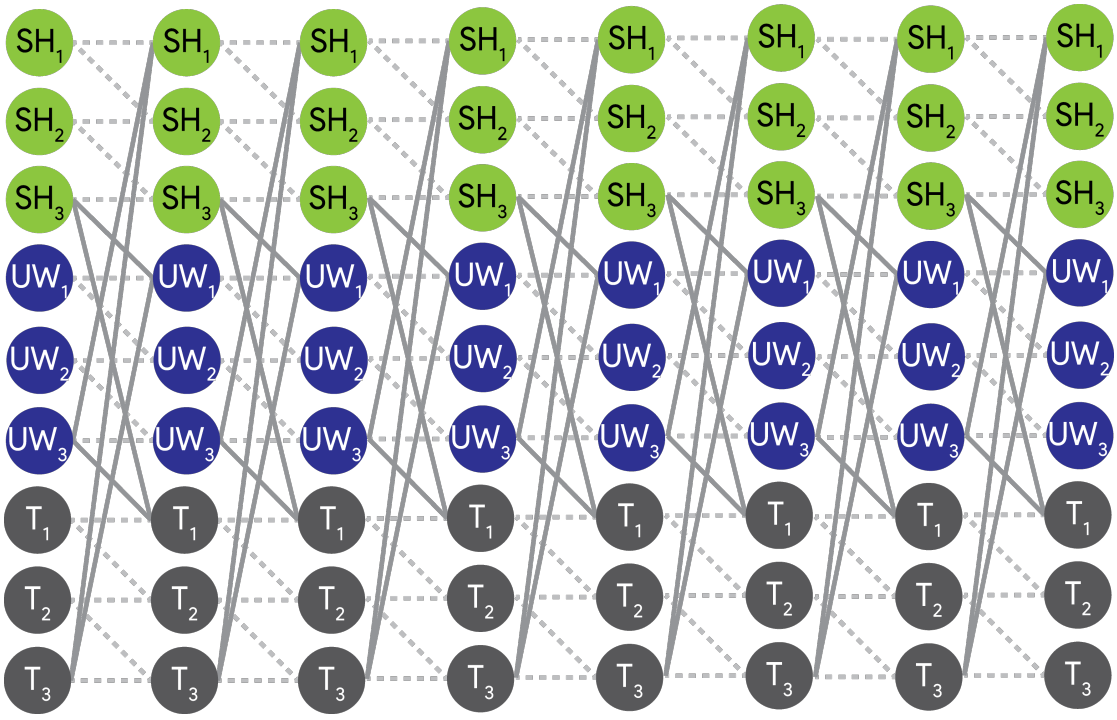


Figure 1.11: An extended version of the HMM for three phonemes, in which each phoneme contains three substates.

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, $\pi_g$ represents the weight of the $g^{th}$ mixture component out of $M$ Gaussian components, with a mean vector $\boldsymbol{\mu}_g$ and covariance matrix $\Sigma_g$. 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 $\gamma_n$ 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.

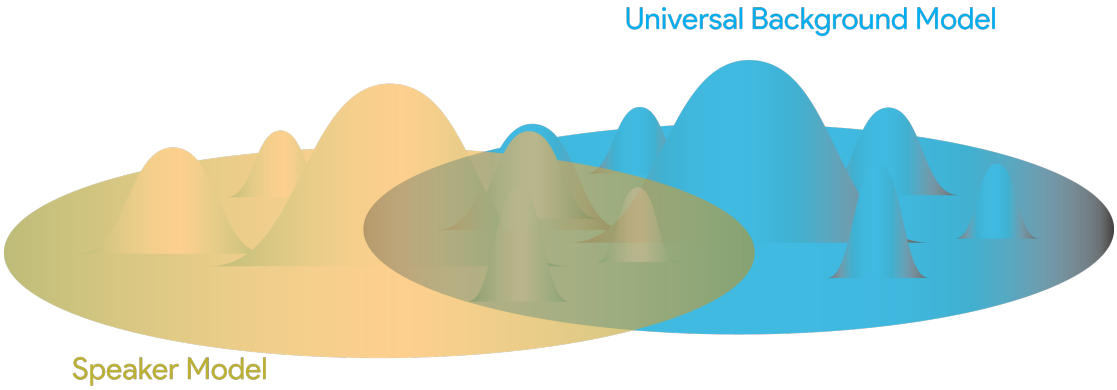


Figure 1.12: An illustration of a speaker model and the adapted speaker model.

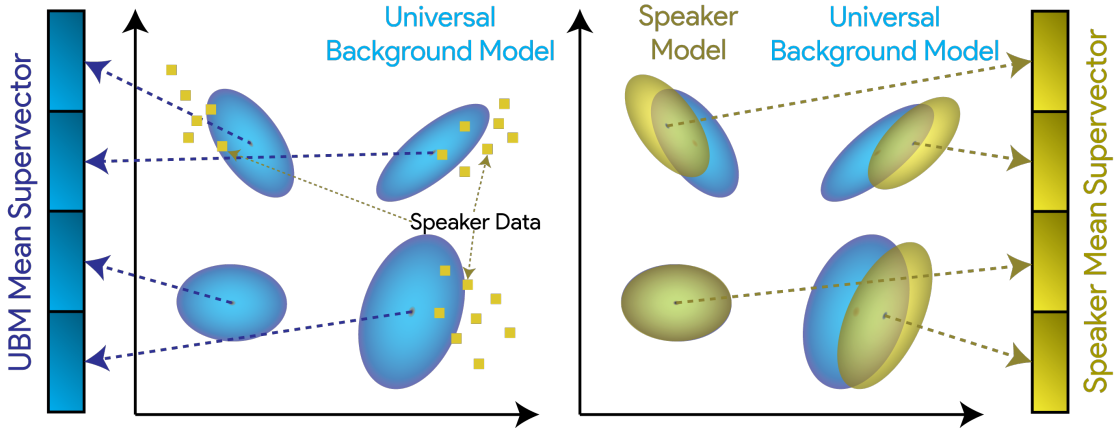


Figure 1.13: An illustration of (left) the GMM-UBM models with 4 components, which lead to the UBM supervector, and (right) the speaker adapted GMM based on the UBM model and the speaker data via a MAP algorithm to create the speaker supervector.

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

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 $\alpha_g$

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

The scaling factor $\beta$ 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 1.13 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 1.13 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

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

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 $\boldsymbol{D}$ 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

# Chapter 2

# Deep Learning Theory

Although the field of deep learning has been growing at an incredibly fast rate, there is still a lot of notions in deep learning that remains not understood. Because it is rather simple to implement and it doesn't require a lot of ingenuity to design a new architecture, a lot of empirical advancements have been published. However, because of the nature of the field, and a bad selection of reviewers for publications/conferences*, there isn't a lot of fundamental understanding of what is happening with deep learning. This becomes extremely crucial for healthcare, as the nature of that field requires a complete understanding of the reasoning behind diagnostic in order to diagnose a patient. Although DL is very powerful and it is able to approach functions with a very high degree of complexity, they are unable to enter the field of healthcare due to the lack of understanding. In other words, how would a patient respond if the hospital provided a diagnose of a deadly disease/disorder, and when asked why, the response would be: "Because the deep neural network with $X$ accuracy said so." In this chapter, four different approaches are described in each section attempting to provide a better understanding of deep learning.

* *It occurred last semester that one of our collegues (a PhD student) in Georgia Tech was invited to review papers for NIPS because he had a few publications in deep learning. He was in fact stressed about it and asking for help as he had never reviewed any publications before, and he was uncertain how to do so.*

**Note that each section has their own independent notation.**

## 2.1 The Stochastic Thermodynamics of Learning [Goldt and Seifert, 2017]

The first field that is described follows the field of thermodynamics and a little bit of information theory. Note that, although showing the work from [Goldt and Seifert, 2017], I am changing some of the notations to a set of notations that is a bit more familiar.

First, modeling a neuron, let a neuron make $N$ connections to ther neurons, which is fully characterized by $\boldsymbol{w} \in \mathbb{R}^N$, and it must learn whether or not to fire an action potential from a set of $P$ different examples $\boldsymbol{x}^p = \{(x_1^p, x_2^p, \cdots, x_N^p) | 1 \leq p \leq P\}$ (i.e. patterns), which describe the activity of all other connected neurons, and say that $x_n^p \in \{-1, 1\}$, which represents the $n^{th}$ feature (i.e. the active/inactive state coming from the $n^{th}$ neuron) of the $p^{th}$ observation. The observations $\boldsymbol{x}^p$ are paired with a ground truth label $y^p \in \{-1, 1\}$, and the neuron tries to predict a label $\hat{y}^p \in \{-1, 1\}$ such that it equates the true label $y^p$. The neuron may be modeled as a thermal environment with the transition rates as $k_p^{\pm}$ for the equilibrium between the states -1 and +1. The equilibrium process can be modeled as the following, where $\mathcal{A}^p$ is the input-dependent function, $k_B$ is the Boltzmann's constant, and $T_{temp}$ represents a fixed temperature.

$$\frac{k_p^+}{k_p^-} = \exp\left(\frac{\mathcal{A}^p}{k_B T_{temp}}\right)$$
$$\mathcal{A}^p = \frac{\boldsymbol{w} \cdot \boldsymbol{x}^p}{\sqrt{N}}$$

This work presents the concept of *learning efficiency*, but first, a few concepts need to be established. In such case, for simplicity, set $k_B = T = 1$ in order to render the energy and entropy dimensionless. Considering

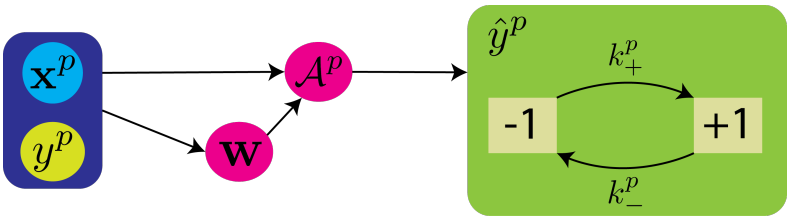


Figure 2.1: The model of a neuron, where the $p^{th}$ input-target pair is used to compute $\mathcal{A}^p$, which then is used to compute the transition rates ratio between the two labels $\{-1, +1\}$

a network with one weight, one sample, and one label ($N = P = 1$), the weight $w$ is assumed to follow the overdamped Langevin equation

$$\frac{dw(t)}{dt} = -w(t) + f(w(t), \boldsymbol{x}, y, t) + \zeta(t)$$

- $f(\cdot)$ = external force, which depents on the learning algorithm chosen. It creates the correlation between the weight $w$ and the input $\boldsymbol{x}$
- $\zeta(t)$ = Gaussian thermal noise, such that the correlation $\langle \zeta(t)\zeta(t') \rangle = 2\delta(t - t')$

By setting the initial conditions occurring at $t = t_0 = 0$, the weight is in thermal equilibrium (i.e. $Pr(w) \propto e^{-\frac{w^2}{2}}$), and the labels are equiprobable (i.e. $Pr(y) = Pr(\hat{y}) = 0.5$). With symmetric rates,

$$k_\pm = \gamma e^{fracA2}$$

$$\frac{k_+}{k_-} = \frac{\gamma \exp \frac{A}{2k_B T_{temp}}}{\gamma \exp -\frac{A}{2k_B T_{temp}}} = \exp \frac{A}{2k_B T_{temp}} \gamma \exp \frac{A}{2k_B T_{temp}} = \gamma \exp \frac{A}{k_B T_{temp}}$$

Therefore, the partial differential equation for the joint probability distribution $Pr(y, w, \hat{y}, t)$ is defined as

$$\begin{cases} \dfrac{\partial Pr(y, w, \hat{y}, t)}{\partial t} = -\dfrac{\partial j_w(t)}{\partial w} + j_{\hat{y}}(t) & \\ j_w(t) = \left[-w + f(w, \boldsymbol{x}, y, t) - \dfrac{\partial}{\partial w}\right] Pr(y, w, \hat{y}, t) & = \text{Probability current for weight} \\ j_{\hat{y}}(t) = k_{\hat{y}} Pr(y, w, \hat{y}, t) - k_{-\hat{y}} Pr(y, w, \hat{y}, t) & = \text{Probability current for the predicted label} \end{cases}$$

Per definition, a probability current (i.e. probability flux) is the mathematical quantity describing the flow of probability in terms of probability per unit time. It is quite similar to if one was to describe the probability density as a fluid, and the probability current is the rate of flow of the fluid.

### 2.1.1 Learning Efficiency

To measure the learning efficiency, one must first define the Shannon entropy $S(X)$ of a random variable $X$, the conditional entropy of $X$ given $Y$, $S(X|Y)$ and the mutual information $I(y : \hat{y})$

$$\begin{cases} S(Y) = -\sum\limits_{y \in Y} Pr(y) \log(Pr(y)) & \\ S(Y|\hat{Y}) = -\sum\limits_{y,\hat{y}} Pr(y, \hat{y}) \log(Pr(y|\hat{y})) = -\sum\limits_{y,\hat{y}} Pr(y, \hat{y}) \log\left(\dfrac{Pr(y, \hat{y})}{Pr(\hat{y})}\right) & \\ I(y : \hat{y}) = S(y) - S(y|\hat{y}) & \end{cases}$$

The $S(Y)$ measures the uncertainty of $Y$, and the mutual information is a natural quantity which measures the information learned. In other words, it measures the different between the uncertainty of the true label $y$ and

the uncertainty of the true label $y$ given $\hat{y}$. It is then possible to measure the mutual information $I(y:\hat{y})$ to the thermodynamic costs of adjusting the weight from time $t_0$ to time $t$, with the total entropy production.

$$\begin{cases} \Delta S_w^{tot} = \Delta S(w) - \Delta Q \\ \Delta S(w) = \text{difference of Shannon entropy of } P(w,t) = \sum_{y,\hat{y}} Pr(y,w,\hat{y},t) \text{ from time } t_0 \text{ to } t \\ \Delta Q = \text{heat dissipated into the medium by the dynamics of the weight} \end{cases}$$

Their work show that the thermodynamic costs of learning in fact bound the information learned with

$$I(y:\hat{y}) \leq \Delta S(w) + \Delta Q$$

for any arbitrary algorithm $f(w, \boldsymbol{x}, y, t)$ at all times $t > t_0$. This is important because they relate the entropy production in the weights with the change in mutual information between $y$ and $\hat{y}$. Therefore, one may define *learning efficiency* $\eta$ as

$$\eta = \frac{I(y,\hat{y})}{\Delta S(w) + \Delta Q} \leq 1$$

In a toy problem, they formulate the learning efficiency of a Hebbian learning, where $N = P = 1$ as $t \to \infty$. This makes $\boldsymbol{x} = x$ as a single input. The Hebbian learning formulation can be described as

$$w_{ij}(t+1) = w_{ij}(t) + \nu x_i(t) x_j(t)$$

where $\nu$ is the learning rate coefficient, and $x$ are the outputs of the $i^{th}$ and $j^{th}$ coefficient. Thus, for a neuron, when the input neuron fires, then the following neuron should fire,and the weight of their connection increases. If the following neuron has a response that differs from the input neuron, then the connection decreases. This means that $x = y = \pm 1$. Therefore, this yields a final weight that is proportional to the $\mathcal{F}$ which is defined as $\mathcal{F} = yx$. Then they choose a learning force $f$ that linearly increases with time

$$f(w,x,y,t) = \begin{cases} \dfrac{\nu\mathcal{F}}{\tau} = \dfrac{\nu yx}{\tau} & t \leq \tau \\ \nu\mathcal{F} = \nu yx & t \leq \tau \end{cases}$$

where $\tau > 0$ represents the learning duration and $\nu > 0$ represents the learning rate in machine learning. One can compute the total entropy loss $\Delta S_w^{tot}$, by computing $Pr(y,w,t)$ and $\Delta Q$. To compute $Pr(y,w,t)$, one should

1. Integrate $\hat{y}$ out of $\frac{\partial}{\partial t} Pr(y,w,\hat{y},t) = -\frac{\partial}{\partial w} j_w(t) + j_{\hat{y}}(t)$, which yields a Fokker-Planck equation
2. Solve the Fokker-Planck equation to obtain $Pr(y,w,t)$

In order to calculate $\Delta Q$, where $f = f(w(t), x, y, t)$, it is computed as

$$\Delta Q = \int_0^\infty dt \int_{-\infty}^\infty dw j_w(t)[-w(t) + f] = \frac{\nu^2 \mathcal{F}(e^{-\tau} + \tau - 1)}{\tau^2}$$

IF the learning duration is too long, no heat is dissipated, but if the learning duration is too sudden, then it the following solutions are

$$\begin{cases} \lim_{\tau\to\infty} \Delta Q = 0 \\ \lim_{\tau\to 0} \Delta Q = \dfrac{\nu^2\mathcal{F}^2}{2} \end{cases}$$

Then the Shannon entropy $\Delta S(w)$ can be computed from the marginal probability $Pr(w,t) = \sum_y Pr(y,w,t)$ which was obtained while computing $\Delta S_w^{tot}$. The stationary solution of a differential equation is defined as when $\frac{\partial \hat{f}}{\partial t}$ is zero, and the solution to the stationary solution of

$$\frac{\partial Pr(y,w,\hat{y},t)}{\partial t} = -\frac{\partial j_w(t)}{\partial w} + j_{\hat{y}}(t) = 0$$

yields the mutual information $I(y:\hat{y})$. This then is enough to create the plot shown in Figure 2.2

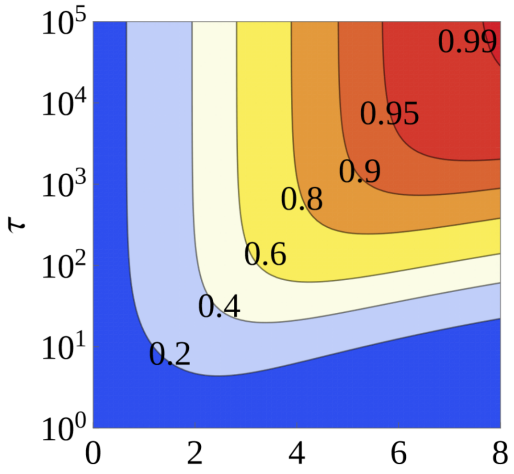


Figure 2.2: The learning efficiency profile with varying learning duration $\tau$ and varying learning rate $\nu$, with a Hebbian learning problem

This however is not a very feasible profile for any learning algorithm. In other words, to set the learning rate to as high as possible may not always work in factor to the optimization problem.

By extending this problem to a multidimentional problem, some of the formulation changes a little bit. A typical neuron usually connects to 1000 other neurons ($N \neq 1$), and it has multiple samples (i.e. $P \neq 1$). Therefore, with $\hat{\boldsymbol{y}} \in \mathbb{R}^P$, having the initial conditions at $t_0$ yield equiprobable results (i.e. $Pr(y) = Pr(\hat{y}) = 0.5$), the quantity that describes the amount of learning after a time $t$ has to be the sum of mutual information $I(y : \hat{y})$ over all inputs. They show that the information is bounded by the total entropy production for all of the weights

$$\sum_{p=1}^{P} I(y^p, \hat{y}^p) \leq \sum_{n=1}^{N} \Delta S(w_n) + \Delta Q_n$$

They set the problem by letting $P$ samples and $N$ dimensions go to infinity while keeping a ratio $\alpha = P/N$ on the order of 1, by setting a learning force the same as before, but substituting $\mathcal{F}$ with $\mathcal{F}_n$, such that

$$\mathcal{F}_n = \frac{1}{\sqrt{N}} \sum_{p=1}^{P} x_n^p y^p$$

By averaging over the noise with fixed $\boldsymbol{y}$, $w_n$ is normally distributed with mean $\langle w_n \rangle = \nu \mathcal{F}_n$, and variance 1. By averaging with respect to the quenched disorder $\boldsymbol{y}$, $\mathcal{F}_n$ is normally distrubed with $\bar{\mathcal{F}}_n = 0$, $\bar{\mathcal{F}}_n^2 = \alpha$, yielding $\langle \bar{w}_n \rangle = 0$ and $\langle \bar{w}_n^2 \rangle = \log(1 + \alpha\nu^2)$. Therefore,

$$\Delta S(w) = \log(1 + \alpha\nu^2)$$

From here, the heat dissipated by the $n^{th}$ weight $\Delta \bar{Q}_n$ is obtained by averaging

$$\Delta Q = \frac{\nu^2 \mathcal{F}^2 (e^{-\tau} + \tau - 1)}{\tau^2}$$

over $\mathcal{F} \to \mathcal{F}_n$. The mutual information $I(y^p : \hat{y}^p)$ is a functional of the marginalized distribution $Pr(y^p, \hat{y}^p)$, which can be found with

$$\begin{cases} I(y^p : \hat{y}^p) = \log(2) - S[Pr(y^p = \hat{y}^p)] \\ S[p] = -p\log(p) - (1-p)\log(1-p) Pr(y^p = \hat{y}^p) = \displaystyle\int_{-\infty}^{\infty} d\Delta^p Pr(\Delta^p) \frac{e^{\Delta^p}}{e^{\Delta^p} + 1} \\ \Delta^p = \dfrac{1}{\sqrt{N}} \boldsymbol{w} \cdot \boldsymbol{x}^p y^p = \boldsymbol{\mathcal{A}}^p y^p \end{cases}$$

This can be used to compute the Hebbian learning efficiency $\tilde{\eta}$ as

$$\tilde{\eta} = \alpha \frac{I(y^p : \hat{y}^p)}{\Delta S(w_n) + \Delta \bar{Q}_n}$$

Hence, if a Monte-Carlo integration of $Pr(\boldsymbol{y}, \boldsymbol{w}, \hat{\boldsymbol{y}})$ with $N = 10,000$, one obtains the following graphs.

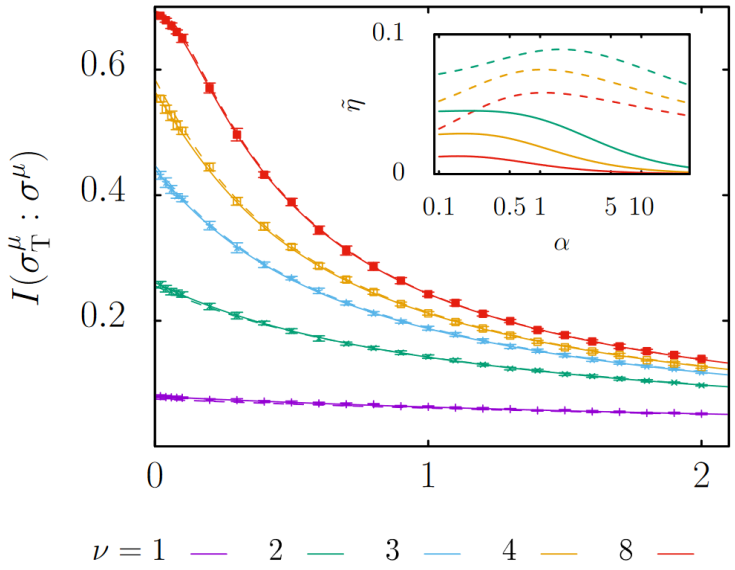


Figure 2.3: The mutual information profile with varying learning rates $\nu$ on with a Hebbian learning problem. The inset graph shows the learning efficiency $\tilde{\eta}$ in the limits $\tau \to 0$ (solid) and $\tau \to \infty$ (dashed)

It is important to note that as $P$ increases, the mutual information decreases because of hte thermal noise in the system, and the well-known failure of Hebbian learning to use information in the samples perfectly. In both plots, $\nu$ increases from bottom to top.

This paper was very interesting because, although it was a full learning efficiency analysis over a very specific type of learning, Hebbian learning, which is connected to a fully connected layer architecture in deep learning, it does open the idea to try to expand the same formulations to different paradigms in deep learning.

## 2.2 Deep Relaxation: PDEs for Optimizing Deep Neural Networks [Chaudhari et al., 2018]

This paper was another example that was the analysis over a very specific part of deep learning. In a previous work, [Chaudhari et al., 2017] had introduced a new type of stochastic gradient descent algorithm, the Entropy-SGD algorithm, which was shown to perform better than optimizers such as the Adam optimizer, yielding a modified loss function, called "local entropy" $f_\gamma(x)$

$$f_\gamma(x) = -\log\left(G_\gamma * e^{-f(x)}\right) \qquad \text{(local entropy)}$$

$$G_\gamma(x) = (2\pi\gamma)^{-N/2} \exp\left(-\frac{|x|^2}{2\gamma}\right) = \text{Gaussian heat kernel}$$

$$f(x) = \text{a loss function (e.g. cross-entropy loss w/ or w/o a regularizer)}$$

$$x = \text{weights of the neural network where } x \in \mathbb{R}^N$$

where $\gamma$ represents the variance (or covariance) in the Gaussian kernel.

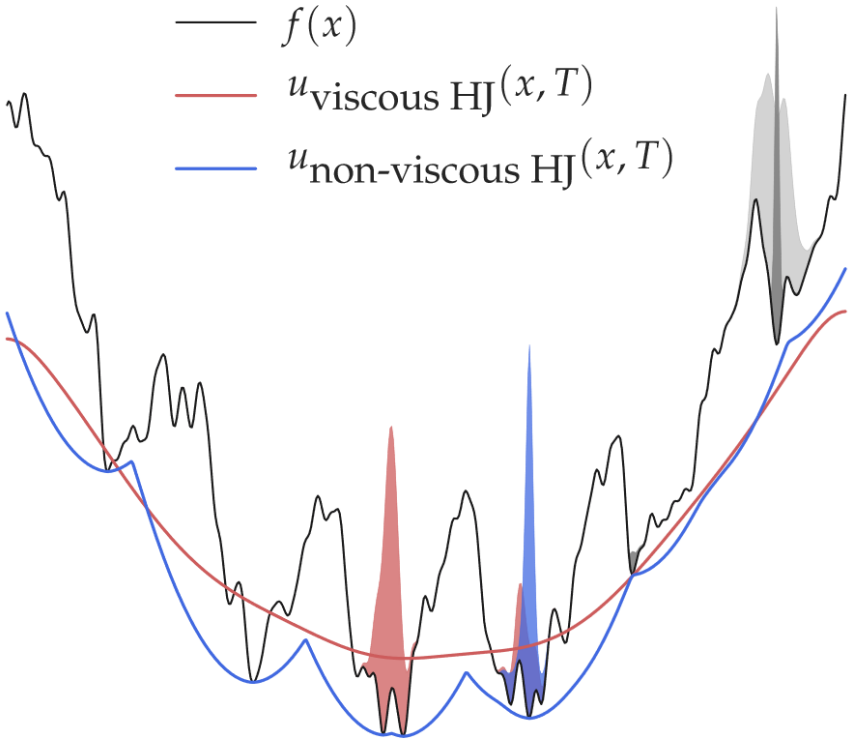


Figure 2.4: A 1D loss profile of different formulations of the loss function, and the location of which algorithms converge to as shown with their respective colors but with a transparency value lower than 1

### 2.2.1 PDE Interpretation of local entropy

It is well known that the solution to the heat equation $u_t = \frac{1}{2}\Delta u$ with an initial condition $u(x, 0) = f(x)$ yields the solution

$$u(x, t) = G_t * f(x)$$

As we know, although the $\gamma$ in (local entropy) represents variance, through the heat equation, it can easily be depicted as a time variable $t$. Then one can prove that the (local entropy) $f_\gamma(x)$ is the solution $u(x, t)$ of the viscous Hamilton-Jacobi PDE

$$\frac{\partial u}{\partial t} = -\frac{1}{2}|\nabla u|^2 + \frac{1}{2}\Delta u \quad \text{where } 0 < t < \gamma \tag{viscous HJ}$$

*Proof.* Define a generalized solution

$$u(x, t) = -\log(\nu(x, t))$$

Then the equation $\nu(x, t) = e^{-u(x,t)}$ solves the heat equation $\nu_t = \frac{1}{2}\Delta\nu$, given $\nu(x, 0) = e^{-f(x)}$. Therefore, the partial derivatives are defined as

$$\begin{cases} \nu_t = -\nu u_t \\ \nabla\nu = -\nu\nabla u \\ \Delta\nu = -\nu\Delta u + \nu|\nabla u|^2 \end{cases}$$

This then yields

$$\begin{aligned}\nu_t &= \frac{1}{2}\Delta\nu \\ -\nu u_t &= \frac{1}{2}\left(-\nu\Delta u + \nu|\nabla u|^2\right) \\ u_t &= \frac{1}{2}\left(\Delta u - |\nabla u|^2\right) = -\frac{1}{2}|\nabla u|^2 + \frac{1}{2}\Delta u\end{aligned}$$

□

Then they show that local entropy is a more powerful way to smooth than with gradient averaging, as well as to derive a simple update rule for the Entropy-SGD algorithm. But first, one needs to show that the gradient of the local entropy $\nabla f_\gamma(x)$ can be shown in two ways:

$$\nabla f_\gamma(x) = \nabla u(x,\gamma) = \begin{cases} \displaystyle\int_{\mathbb{R}^n} \frac{x-y}{\gamma}\rho_1^\infty(dy;x) \\ \displaystyle\int_{\mathbb{R}^n} \nabla f(x-y)\rho_2^\infty(dy;x)\end{cases}$$

$$\text{where } \begin{cases}\rho_1^\infty(y;x) = Z_1^{-1}\exp\left(-f(y) - \frac{1}{2t}|x-y|^2\right) \\ \rho_2^\infty(y;x) = Z_2^{-1}\exp\left(-f(x-y) - \frac{1}{2t}|y|^2\right)\end{cases}$$

This can then be extended towards the non-viscous form of the Hamilton-Jacobi equation

$$\frac{\partial u}{\partial t} = -\frac{1}{2}|\nabla u|^2 \qquad \text{(non-viscous HJ)}$$

where it is just like the (viscous HJ), but the coefficient in front of the Laplacian operator $\Delta$ is zero, which makes a simpler formula for the gradient. One may define, as shown in **Lemma 5** in [Chaudhari et al., 2018], that if $u(x,t)$ is the viscosity solution of (HJ) with $u(x,0) = f(x)$,

$$u(x,t) = \inf_y\left\{f(y) + \frac{1}{2t}|x-y|^2\right\} \qquad \text{(HL)}$$

Define the proximal operator (defined in the community) as

$$y^* = prox_{tf}(x) = \arg\min_y\left\{f(y) + \frac{1}{2t}|x-y|^2\right\}$$

If the proximal operator is a singleton, $\nabla_x u(x,t)$ exists, which is

$$\nabla_x u(x,t) = \frac{x-y^*}{t} = \nabla f(y^*) \qquad (2.1)$$

Using this, the discrete-time gradient descent dynamics using 2.1 is

$$x_{k+1} = x_k - \eta t^{-1}(x_k - prox_{tf}(x_k))$$

where $\eta > 0$ as the step-size. If $\eta = t$, then it becomes the proximal point iteration

$$x_{k+1} = prox_{tf}(x_k)$$

The standard proximal gradiant descent is given by

$$x_{k+1} = prox_{th}(x_k - t\nabla g(x_k))$$

where the loss function is made up of a convex function $g(x)$, and a potentially non-differentiable regularizer $h(x)$. The proximal operator equals to the implicit gradient descent, which is led due to the non-viscous HJ, shown as 2.2. Note how it differs from the gradient descent update, shown in 2.3

$$x_{k+1} = x_k - \eta \nabla f(x_{k+1}) \tag{2.2}$$

$$x_{k+1} = x_k - \eta \nabla f(x_k) \tag{2.3}$$

### 2.2.2 Derivation via Homogenization

Homogenization is a technique highly used in dynamical systems that have multiple time scalesm, and it couples a few fast variables with other variables that evolve very slowly. Therefore, in the limit that the time scales separate, this allows one to compute averages over the very fast variables, which then allows us to obtain averaged equations for the slow variables. Elastic-SGD is an algorithm by [Zhang et al., 2015] in order to minimize communication overhead between a set of workers that optimize replicated copies of the original function in distributed training of neural networks. Homogenization is then used to show that Elastic-SGDs are in fact minimzing the local entropy under ergodic conditions. Consider the following stochastic differential equations (SDEs)

$$\begin{aligned} dx(s) &= h(x,y)ds \\ dy(s) &= \frac{1}{\epsilon} g(x,y)ds + \frac{1}{\sqrt{\epsilon}} dW(s) \end{aligned} \tag{2.4}$$

where $h, g$ are sufficiently smooth functions, $W(s)$ represents the Wiener process, and the parameter $\epsilon > 0$ is the homogenization parameter. Therefore, in the limit of $\epsilon \to 0$, if the unique invariant measure $\rho^\infty(y;x)$ exists and it is ergodit, then the dynamics of $x(s)$ converge to

$$\begin{cases} dX(s) = \bar{h}(X)ds \\ \bar{h}(X) = \int h(X,y)\rho^\infty(dy;X) \end{cases}$$

where $\bar{h}(X)$ represents the homogenized vector field for $X$. The gradient of the local entropy has already been defined as

$$\nabla f_\gamma(x) = \gamma^{-1} \int_{\mathbb{R}^N} (x-y)\rho_1^\infty(dy;x)$$

Therefore we can say that

$$\begin{aligned} h(x,y) &= -\gamma^{-1}(x-y) \\ g(x,y) &= \nabla f(y) + \frac{1}{\gamma}(y-x) \end{aligned}$$

Then 2.4 becomes

$$\begin{aligned} dx(s) &= -\gamma^{-1}(x-y)ds \\ dy(s) &= -\frac{1}{\epsilon}\left[\nabla f(y) + \frac{1}{\gamma}(y-x)\right]ds + \frac{1}{\sqrt{\epsilon}} dW(s) \end{aligned} \tag{2.5}$$

While a standard neural network optimizes the optimization problem

$$x^* = \arg\min_x f(x)$$

an Elastic-SGD solves

$$\arg\min_{x^1,\cdot,x^n} \sum_{i=1}^n f(x^i) + \frac{1}{2\gamma}|x^i - x|2$$

Therefore, this yields

$$dx(s) = -\gamma^{-1}\sum_{i=1}^{n}(x - x^i)ds$$
$$dx^i(s) = -\frac{1}{\epsilon}\left[\nabla f(x^i) + \frac{1}{\gamma}((x^i - x)\right]ds + \frac{1}{\sqrt{\epsilon}}dW^i(s) \tag{2.6}$$

The homogenized vector field can the be given by

$$\bar{h}(X) = \gamma^{-1}\int (x^1 - X)\rho^{\infty}(dx^1; X)$$

Where all of these formulations match exactly the homogenized dynamics of Entropy-SGD. When comparing the heat equation to the viscous Hamilton-Jacobi equation, one may choose to study the equation below, as the convolution of hte exponentiated loss is quite complicated.

$$f_\gamma(x) = G_\gamma * f(x)$$

Since $\nabla f_\gamma^2$ represents the averaging of the original gradient $\nabla f(x)$ over Gaussian pertubations of variance $\gamma$, the gradient descent dynamics of $f_\gamma^2$ may be written as

$$dx(s) = -\nabla f(x - y)ds$$
$$dy(s) = -\frac{1}{\epsilon\gamma}yds + \frac{1}{\sqrt{\epsilon}}dW(s)$$

It is important to note that for the heat equation, $y(s)$ has no dependencies on $x(s)$, whereas Entropy-SGD in does have it, on the term $\nabla f(x - y)$. Although the heat equation smoothing is a lot more performed, the Entropy-SGD does have a much better empirical performance.

### 2.2.3 Stochastic Optimal Control Interpretation

The fact that the local entropy $f_\gamma(x)$ can be formulated as a viscous Hamilton-Jacopy eqation, and it is now possible to the gradient descent as an optimal control problem. Therefore, consider the following controlled SDE

$$dx(s) = -\nabla f(x)ds - \alpha(s)ds + dW(s) \tag{CSGD}$$

where $t \leq s \leq T$, $x = x(t)$, and $\alpha(\cdot)$ is the control. With a terminal cost function $V(x(T)) : \mathbb{R}^N \to \mathbb{R}$, one can represent the prototypical quadratic running cost

$$C(x, \alpha) = \mathbb{E}\left[V(x(T)) + \frac{1}{2}\int_0^T |\alpha(s)|^2 ds\right]$$

If one can then set the minimum expected cost over all controls and paths to be

$$u(x, t) = \min_{\alpha(\cdot)} C(x(\cdot), \alpha(\cdot))$$

Then one can set $u(x, t)$ to be the unique viscosity solution of the Hamilton-Jacobi-Bellman PDE

$$-\frac{\partial u}{\partial t} = -\nabla f(x)\cdot\nabla u(x, t) - \frac{1}{2}|\nabla u|^2 + \frac{1}{2}\Delta u$$

with a terminal condition $u(x, T) = V(x)$. This specific case leads the optimal control to be the gradient of the solution

$$\alpha^*(x, t) = \nabla u(x, t)$$

Then, they finally show that, assuming $x_{csgd}(s)$ be the solution for the controlled stochastic gradient descent (CSGD) and $x_{sgd}(s)$ be the solution for the stochastic gradient descent (SGD), where they are shown below again for the convenience of the reader

$$dx(t) = -\nabla f(x)dt - \alpha(t)dt + dW(t) \tag{CSGD}$$

They show that

$$\mathbb{E}[V(x_{csgd}(t))] \leq \mathbb{E}[V(x_{sgd}(t))] - \frac{1}{2}\mathbb{E}\left[\int_0^t |\alpha^*(x_{csgd}, s)|^2 ds\right]$$

As results, the authors compare the four different algorithms to test their effectiveness. For each of them, $L$ represents the number of gradient evaluations performed prior to a weight update, and thus the number of epochs times $L$ is a measure of the wall-clock time, since a larger $L$ leads to a slower computation time.

- Entropy-SGD ($L = 20$)
- Smoothing by the HEAT equation ($L = 5$)
- Non-viscous Hamilton-Jacobian Equation ($L = 5$)
- Stochastic Gradient Descent ($L = 1$)

All of these experiments are then performed under three different networks with the used datasets in parenthesis

- mnistfc (MNIST)
- LeNet (MNIST)
- All-CNN (CIFAR-10)

One very interesting thing about this paper was the introduction of understanding how to choose certain hyperparameters; most specifically $\gamma$. As we have seen before, $\gamma$ is a parameter for the Gaussian kernel, which is also used to represent time, and the higher the value, the better it is to smooth a function. By scoping through the loss function, they provide a decay rule so that it allows for a smooth loss function in the beginning to quickly progress down, and to then preserve the locations of the local minima in the end as $\gamma \to 0$, where $k/L$ is an integer, and $\gamma_0 \in [10^4, 10]$

$$\gamma = \gamma_0(1 - 10^{-3})^{k/L}$$

Below are a table of the results, where it summarizes the minimum validation error (%) @ effective epochs.

| | Item | | | |
|---|---|---|---|---|
| Model | Entropy-SGD | HEAT | HJ | SGD |
| mnistfc | **1.08 ± 0.02 % @ 120** | 1.13 ± 0.02 %@ 200 | 1.17 ± 0.04 %@ 200 | 1.10 ± 0.01% @ 149 |
| LeNet | 0.5 ± 0.01% @ 80 | 0.59 ± 0.02 %@ 75 | **0.5 ± 0.01% @ 70** | 0.5 ± 0.02% @ 67 |
| All-CNN | 7.96 ± 0.05% @ 160 | 9.04 ± 0.04% @ 150 | **7.89 ± 0.07% @ 145** | 7.94 ± 0.06% @ 195 |

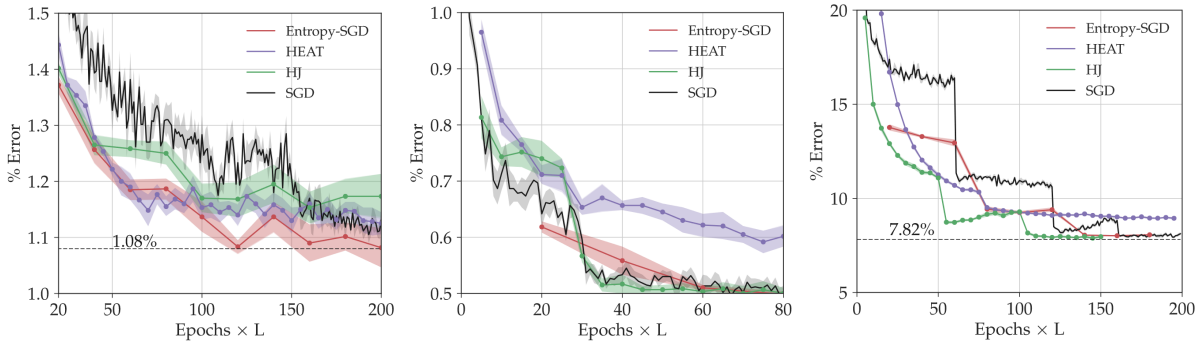


Figure 2.5: The figure above portrays the different progressions of the four previously mentioned methods on the (left) mnistfc network, (center) the LeNet, and (right) All-CNN network, showing that the HJ methodology often times achieves lower losses at a faster rate than the currently used optimization methods in the community

As the table shows, the Hamilton-Jacobi implementation will often lead to a lower expected percent error at a lower epoch. This means that by solving a stochastic control problem to the viscous HJ PDE yields a smaller

expected loss as compared to the stochastic gradient descent. This work also provides a better understanding to the choice of parameters. More specifically, $\gamma$ is equivalent to time for the gradient flow in the local entropy and Elastic-SGD.

## 2.3 Characterization of Neural Networks as an Encoder-Decoder with Mutual Information [Shwartz-Ziv and Tishby, 2017]

This work was another very interesting work, in which they approached the analysis of deep learning from the perspective of information theory. A lot of the analysis shown in this paper involves the comparison of mutual information between different variables to understand how is it that fully connected neural networks learn.

### 2.3.1 Background

In this work, the authors use the notation of $x \in X$ to be the input patters, and $y \in Y$ to be the labels. Because neural networks is a way of showing the inputs as reduced representations, $T(X)$ denotes a representation (i.e. a layer), and the network tries to learn from an empirical sample drawn from an unknown joint distribution $P(X, Y)$.

In this work, the authors postulate a that deep neural network (DNN) generates a Markov chain representation by minimizing an empirical error over the weights, and the optimization is taken through a stochastic gradient descent (SGD) over a noisy estimate of the gradient of the empirical error, also known as backpropagation. This work also treats a whole layer $T$ as a single random variable, where an encoder $Pr(T|X)$ and a decoder $Pr(Y|T)$ characterize the layer, and an invertible transformation of representations is important to ensure that the transformation preserves information. Therefore, it is important quantify the representations with numbers that are invariant to any invertible reparametrization of T, and the mutual information $I(T, X)$ and $I(T, Y)$ is a fitting quantification. The mutual information is defined as

$$\begin{aligned} I(X,Y) &= D_{KL}[Pr(x,y) \| Pr(X)Pr(Y)] \\ &= \sum_{x,y} Pr(x,y) \log\left(\frac{Pr(x,y)}{Pr(x)Pr(y)}\right) \\ &= \sum_{x,y} Pr(x,y) \log\left(\frac{Pr(x|y)}{Pr(x)}\right) \\ &= H(X) - H(X|Y) \\ &= \text{entropy of } X - \text{conditional entropy of X given Y} \end{aligned}$$

The mutual information quantifies the number of relevant bits that an input $X$ contains about label $Y$ on average. It is also important to mention that mutual information also has the commutative property.

It is also important to mention some properties with graphs and their mutual information characteristics. Given a Markov chain

$$X \to Y \to Z$$

the data processing inequality states that

$$I(X, Y) \geq I(X, Z)$$

which means that information is generally lost when transmitted through a noisy channel.

### 2.3.2 The Information Plane

In this work, [Shwartz-Ziv and Tishby, 2017] postulate that the SGD has two phases that occur in this sequence

1. Empirical error minimization (ERM)
2. Representation compression

where both of them have very different signal-to-noise ratios of the stochastic gradients at every layer. In the ERM, the gradient norms are much greater than the stochastic fluctuation, and in the compression, the fluctuation of gradients is much greater than the means, where the weights change as Weiner processes (i.e. random diffusions) with small influence on the error gradients. A lot of the work in [Shwartz-Ziv and Tishby, 2017] perform the analysis on the information plane, where the two axes are different mutual information measurements.

For a $K$-layer neural network, where $T_i$ represents the $i^{th}$ hidden layer as the single multivariate variable, any representation $T_i(X)$ is a mapping of $X$, characterized by $P(X,Y)$ or by the encoder distribution $P(T|X)$ and the decoder distribution $P(Y|T)$, ad the layers are mapped to $K$ monotonic connected paths in the plane, meaning that they satisfy the data processing inequality (DPI), where $\hat{Y}$ represents the predicted label,

$$I(X,Y) \geq I(T_1,Y) \geq I(T_2,Y) \geq \cdots \geq I(T_K,Y) \geq I(\hat{Y},Y)$$
$$H(X) \geq I(X,T_1) \geq I(X,T_2) \geq \cdots \geq I(X,T_K) \geq I(X,\hat{Y})$$

Thus this work shows graphs plotting $I(X,T_i)$ vs $I(T_i,Y)$.

### 2.3.3 Numerical Experiments

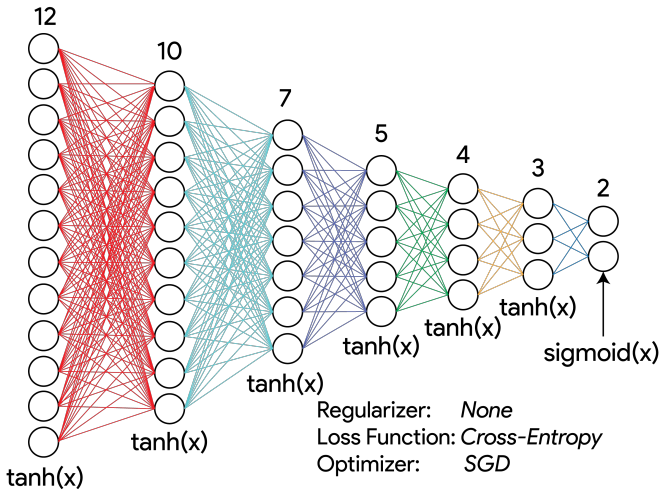


Figure 2.6: The network used for all of the numerical experiments

For the numerical experiments, as shown in Figure 2.7, the authors used a fully connected architecture of the sequence (12,10,7,5,4,3,2) nodes where the inputs were binary, all of the activation functions were the hyperbolic tangent except for the last one with a sigmoid, a cross-entropy loss without any regularizer, and a stochastic gradient descent. In order to estimate the mutual information, there would be 50 randomized initializations, and the hyperbolic tangent values were binned in 30 intervals between -1 and 1.

When observing the information plane profile of 50 randomly initialized networks after different epochs, as shown in Figure 2.7, it is possible to see that the layers first go through the empirical error minimization, where it allows the $I(T,Y)$ to greatly increase, and this phase occurs quickly. After the first phase, then the compression phase occurs, where the network starts to find a better compression on the input data, which is indicated by the reduction in $I(X,T)$.

Another approach that was to look at different training data sizes, which is shown in Figure 2.8. By doing so, it was clear to see that by having low training data size, the ERM phase remains relatively the same, but however, in the compression phase, the mutual information between the layers and the labels do decrease, meaning that it is overfitting to the small amount of data, which simplfies the layers' representations but loses the relevant information.

The results from Figure 2.9 show the profiles of the normalized mean and standard deviations of the weights in a network. It starts off with a high signal-to-noise ratio, represented as the drift, and then it ends with a low signal-to-noise ratio, represented as the diffusion. During the drift, the network reduces the empirical error, while in the diffusion, it adds random noise to the weights, making it evolve as if it was a Wiener process.

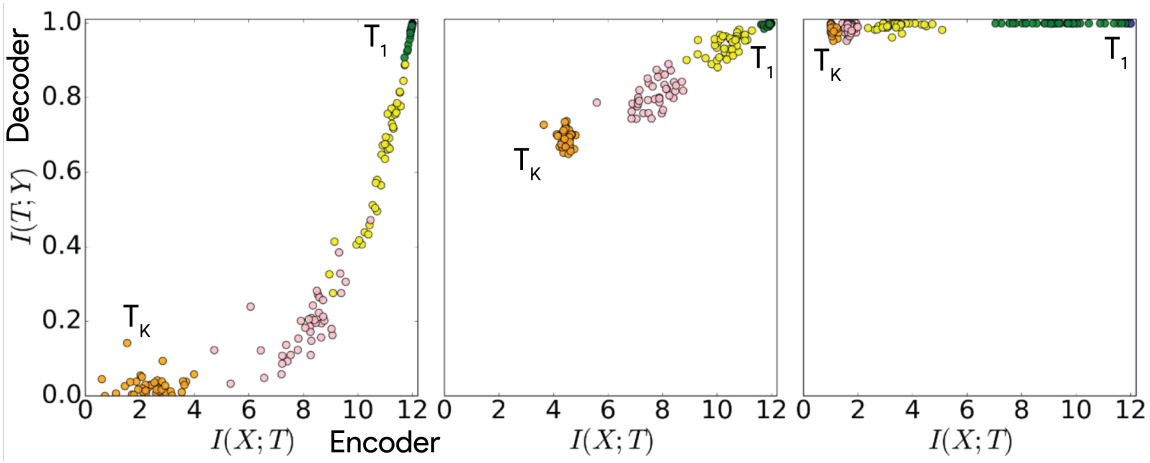


Figure 2.7: The profile of the information place over of 50 randomly initialized networks, (left) when initialized, (center) after 400 epochs, and (right) after 9000 epochs. The dots in orange represent the last layer while the dots in green represent the initial layer.

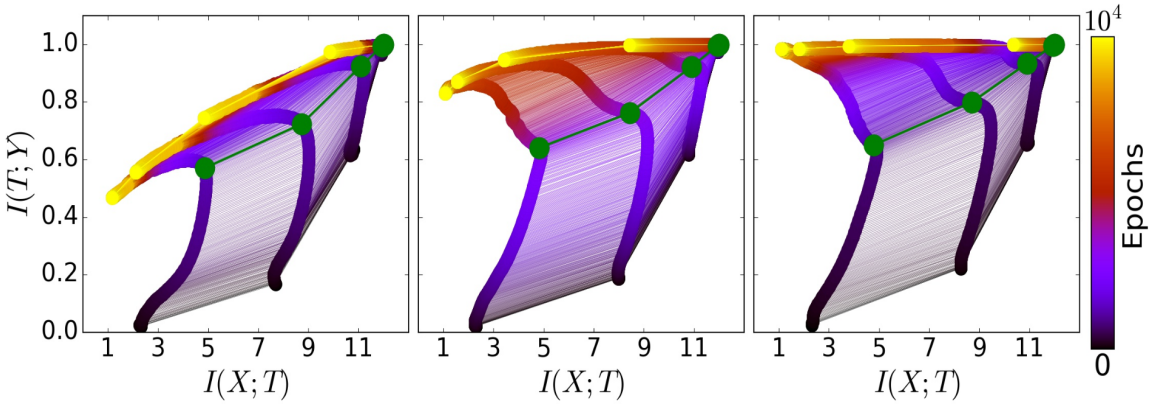


Figure 2.8: The evolution of the mutual information profile while using (left) 5% of the data, (center) 45% of the data, and (right) 85% of the data

These processes can then be described by a Focker-Planck equation, where the stationary distribution attempts to maximize the conditional entroy, thus minimizing the mutual information.

The final set of main results shown are in Figure 2.10, where the authors change the number of layers in the network, ranging from 1-6 hidden layers. This shows that the compression phase is significantly faster when the network is deeper, where the compression occurs faster on the deeper layers. Another remark they found is that, for this case, widening the hidden layers did not help much as all of the hidden layers eventually compress.

This paper was very interesting as it proposes a different methodology to understand how does information flow within neural networks. Although this is a topic that is very specific to these fully connected layered networks, it provides an entrypoint in a different perspective, which can be applied to different types of neural networks.

## 2.4 Residual Networks as a Mean-Field Optimal Control Problem [E et al., 2018]

### 2.4.1 Background

This paper was very interesting because it creates a optimal control formulation of a specific type of deep learning methodology. More specifically, it formulates a residual network, as shown in Figure 2.11, as an optimal control

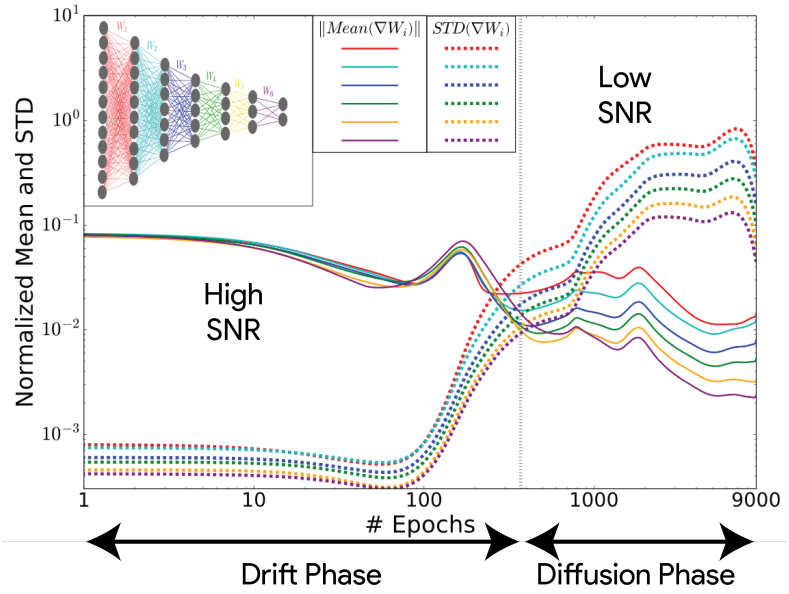


Figure 2.9: The evolution of the normalized means and standard deviations of the weights and over the epochs. The first phase is described as the drift phase, with high signal-to-noise ratio, and the second phase is described as the diffusion phase, with low signal-to-noise ratios. The colors are represented by the weights of the respective layers shown in the inset illustration of the neural network.

problem. This is specifically because the feed forward propagation of a $T$-layer residual network is represented by

$$x_{t+1} = x_t + f(x_t, \theta_t) \qquad t \in \{0, \cdots, T-1\} \tag{2.7}$$

where $x_0$ is the input to the network and $x_T$ is the final output, which is compared to a target $y_0$, which corresponds to $x_0$ via a loss function, and the goal is to train $\theta_0, \cdots, \theta_{T-1}$ such that $x_T$ is as close as possible to $y_0$. In this problem,$x_0 \in \mathbb{R}^d$ and $y_0 \in \mathbb{R}^l$ such that they are random variables jointly distributed according to the joint probability distribution $\mu_0 \mathbb{P}_{(x_0,y_0)}$. The set of admissible controls (i.e. training weights) is defined as $\Theta \subseteq \mathbb{R}^m$. The feed-forward dynamics $f$ maps $f : \mathbb{R}^d \times \Theta \to \mathbb{R}^d$, the terminal loss function $\Phi$ maps $\Phi : \mathbb{R}^d \times \mathbb{R}^l \to \mathbb{R}$, and the regularizer $L$ maps $L : \mathbb{R}^b \times \Theta \to \mathbb{R}$. The formulation for 2.7 is great, because, on can formulate a ordinary differential equation of it as

$$\dot{x}_t = f(x_t, \theta_t) \tag{2.8}$$

Here, the bold-faced letters are used to describe a path-space quantity. For instance, $\boldsymbol{\theta} = \{\theta_t | 0 \leq t \leq T\}$. Due to the equation 2.8 being a fitting formulation for residual networks, a risk minimization problem in deep learning, it can then be formulated as a optimal control problem, either as a mean-field as well as an empirical

$$\begin{cases} \text{Mean-Field:} & \inf\limits_{\boldsymbol{\theta} \in L^\infty([0,T],\Theta)} J(\boldsymbol{\theta}) = \mathbb{E}_{\mu_0}\left[\Phi(x_T, y_0) + \int_0^T L(x_t, \theta_t) dt\right] \text{ Subject to } \dot{x}_t = f(x_t, \theta_t) \\ \text{Empirical:} & \inf\limits_{\boldsymbol{\theta} \in L^\infty([0,T],\Theta)} J_N(\boldsymbol{\theta}) = \frac{1}{N}\sum\limits_{i=1}^N \left[\Phi(x_T^i, y_0^i) + \int_0^T L(x_t^i, \theta_t) dt\right] \text{ Subject to } \dot{x}_t^i = f(x_t^i, \theta_t) \end{cases}$$

From this point on, $w \in \mathbb{R}^{d+l}$ will denote the concatenated $(x, y)$, $\bar{f}(w, \theta)$ represents the extended $(d+l)$-dimensional feed-forward function, $L(\bar{w}, \theta)$ represents the extended $(d+l)$-regularization loss, and $\bar{\Phi}(w)$ represents the terminal loss function. The gradient operators are depicted as $\nabla$, the Fréchet derivative on Banach spaces are depicted as $D$, $a \cdot b$ denotes the inner product of two Euclidean vectors, the $\|\cdot\|$ denotes the Euclidean norm, and the $\|\cdot\|$ denotes the absolute value. A fixed and sufficiently rich probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is represented as $\Omega$, and the set of square integrable probability measures on the Euclidean space $\mathbb{R}^{d+l}$ is shown as $\mathcal{P}_2(\mathbb{R}^{d+l})$, which is equipped with the 2-Wasserstein distance

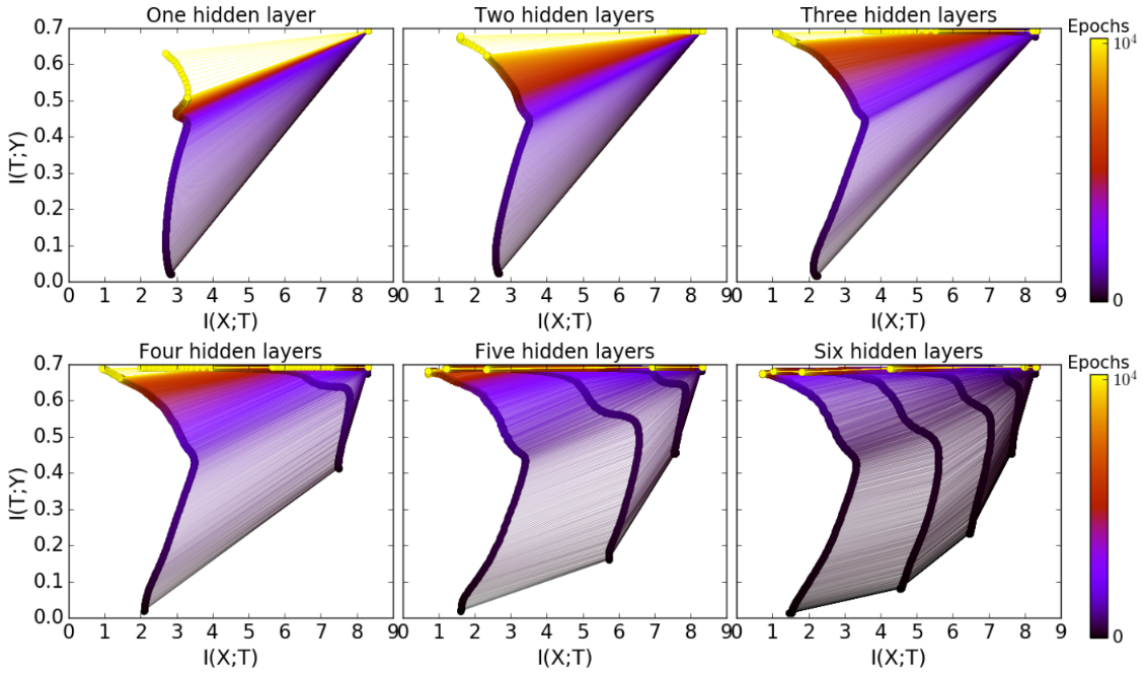


Figure 2.10: The information plane profiles with varying number of hidden layers, between 1-6.

$$W_2(\mu,\nu) = \inf\left\{\|X-Y\|_{L^2} | X, Y \in L^2(\Omega, \mathbb{R}^{d+l}) \text{ with } \mathbb{P}_X = \mu, \mathbb{P}_Y = \nu\right\}$$

With a measurable function $\psi : \mathbb{R}^{d+l} \to \mathbb{R}^q$ that is square integrable with respect to $\mu$, define

$$\langle \psi(\cdot), \mu \rangle = \int_{\mathbb{R}^{d+l}} \psi(w)\mu(dw)$$

Additionally, from this point on, the notation for the infimum will be shorthanded as

$$\inf_{\boldsymbol{\theta}} = \inf_{\boldsymbol{\theta} \in L^\infty([0,T],\Theta)}$$

### 2.4.2 Mean-Field Dynamic Programming and HJB Equation

In this section, the following assumptions are taken:

- $f, L, \Phi$ are bounded, nad are Lipschitz continuous with respect to $x$, while the constants $f$ and $L$ are independent of $\theta$
- $\mu_0 \in \mathcal{P}(\mathbb{R}^{d+l})$

By using the formulation previously specified, the Hamilton-Jacobi-Bellman equation (HJB), which is subject to 2.8, can be rewritten as follows

$$\begin{aligned} J(t,\mu,\boldsymbol{\theta}) &= \mathbb{E}_{(x_t,y_0)\sim\mu}\left[\Phi(x_T,y_0) + \int_t^T L(x_t,\theta_t)dt\right] \\ &= \langle \bar{\Phi}(\cdot), \mathbb{P}_T^{t,\mu,\boldsymbol{\theta}} \rangle + \int_t^T \langle \bar{L}(\cdot,\theta_s), \mathbb{P}_s^{t,\mu,\boldsymbol{\theta}} \rangle ds \end{aligned}$$

With this, the value function $v^*(t,\mu)$ can be obtained with

$$v^*(t,\mu) = \inf_{\boldsymbol{\theta}} J(t,\mu,\boldsymbol{\theta}) \text{ where } J(\boldsymbol{\theta}) = v^*(0,\mu_0) \text{ by definition}$$

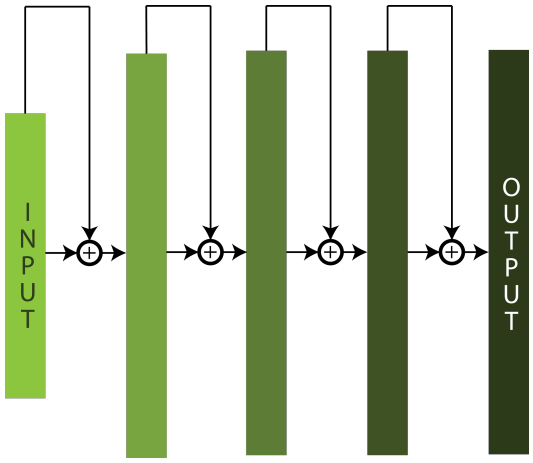


Figure 2.11: An illustration of a residual network, where every layer is added to the following layer

This value function is nice because it satisfies the dynamic programming principle, which states that for any optimal trajectory, if one start from any intermediate state in the trajectory, the remaining trajectory must be optimal. In other words

$$v^*(t,\mu) = \inf_{\boldsymbol{\theta}} \left[ \int_t^{\hat{t}} \langle \bar{L}(\cdot,\theta_s), \mathbb{P}_s^{t,\mu,\boldsymbol{\theta}} \rangle ds + v^*(t, \mathbb{P}^{\hat{\approx},\mu_{\hat{t}}\boldsymbol{\theta}}_{\hat{t}}) \right]$$

Given such, the authors obtain a HJB in the Wasserstein Space by performing a formal Taylor Series expansion on the 2.4.2, yielding

$$\begin{cases} \dfrac{\partial v}{\partial t} + \inf_{\theta} \langle \dfrac{\partial v(t,\mu)(\cdot)}{\partial \mu} \cdot \bar{f}(\cdot,\theta) + \bar{L}(\cdot,\theta), \mu \rangle = 0 & \text{on } [0,T] \times \mathcal{P}_2(\mathbb{R}^{d+l}) \\ v(T,\mu) = \langle \bar{\Phi}(\cdot), \mu \rangle & 0 \text{ on } \mathcal{P}_2(\mathbb{R}^{d+l}) \end{cases} \tag{HJB}$$

This then leads to the Proposition 3 in [E et al., 2018] that if $v$ is solution to (HJB) and there exists a $\theta'(t,\mu)$ that attains the infimum of (HJB), then $v = v^*$ and $\theta'$ is the optimal feedback control policy. This links the smooth solutions of HJB with the solutions of the mean-field optimal control problem, which means it also links to the population minimization problem in deep learning.

### 2.4.3 Viscosity Solutions of HJB

Many times, we cannot expect smooth solutions to the (HJB), and thus it's important to extend it to a weak solution by adding a viscosity to it in the Wasserstein space of the probability measures. By "lifting" the equations, it is possible to work in the Hilbert spaces instead of in the Wasserstein space. By defining Hamiltonian $\mathcal{H}$ as

$$\mathcal{H}(\xi, P) = \inf_{\theta} \mathbb{E}[P \cdot \bar{f}(\xi,\theta) + \bar{L}(\xi,\theta)]$$

The HJB can be set as

$$\frac{\partial V}{\partial t} + \mathcal{H}(\xi, DV(t,\xi)) = 0 \text{ on } [0,T] \times L^2(\Omega)$$
$$V(T,\xi) = \mathbb{E}[\bar{\Phi}(\xi)] \text{ on } L^2(\Omega)$$

Using the above, [E et al., 2018] show that the value function $v^*(t,\mu)$ is a viscosity solution of (HJB), showing that it exists and it is unique. Thus, the optimal control policy can be used to define an optimal control as the solution to the learning problem. This then provides an entry point to understanding DL as a variational problem, whose solution is characterized by the HJB.

### 2.4.4 Mean-Field Pontryagin's Maximum Principle

In classical optimal control, one is able to find the best possible local control from taking a dynamical system from a start state to a goal state via Pontryagin's maximum principle (PMP). However, in the mean-field formulation, a common control parameter is shared by all of the input-target pairs, which are taken from the same distribution $\mu_0$. Therefore, there must exist a maximum principle in an average sense. Below are the assumptions for this section

- The function $f$ is bounded, $f, L$ are continuous in $\theta$, and $f, L, \Phi$ are continuously differentiable with respect to $x$
- The distribution $\mu_0$ has a bounded support in $\mathbb{R}^d \times \mathbb{R}^l$.

The Mean-Field PMP can be as, assuming that $\boldsymbol{\theta}^*$ be a solution of the mean-field formulation of the HJB such that $J(\boldsymbol{\theta}^*)$

$$
\begin{aligned}
&\dot{x}_t^* = f(x_t^*, \theta_t^*) \text{ s.t. } x_t^* = x_0 \\
&\dot{p}_t^* = -\nabla_x H(x_t^*, p_t^*, \theta_t^*) \text{ s.t. } p_T^* = -\nabla_x \Phi(x_T^*, y_0) \\
&\mathbb{E}_{\mu_0}[H(x_t^*, p_t^*, \theta_t^*)] \geq \mathbb{E}_{\mu_0}[H(x_t^*, p_t^*, \theta)] \\
&H(x, p, \theta) = p \cdot f(x, \theta) - L(x, \theta) = \text{ Hamiltonian function } H : \mathbb{R}^d \times \mathbb{R}^d \times \Theta \to \mathbb{R}
\end{aligned}
$$

The last equation is important because it is an unique feature for the PMP-type sttatements, where it makes optimal solutiosn globally maximize the Hamiltonian function. This is advantageous as it can deal with the case where the dynamics are not differentiable with respect to the controls (i.e. training weights), or when the optimal controls lie on the boundary of the set $\Theta$.

The authors further show that the PMP solutions can be derived from the HJB equation using the method of characteristics, where the derived Hamilton's equations 2.9 are the characteristic equations of the HJB equation, thus the PMP pinpoints the necessary conditiona characteristic of HJB equation that originates from $\mu_0$ must satisfy.

$$\dot{x}_t = f(\xi_t, \theta^*) \tag{2.9}$$

$$\dot{p}_t = -\nabla_x f(x_t, \theta_t^*) p_t - \nabla_x L(x_t, \theta_t^*) \tag{2.10}$$

The authors conlude the paper proving that when a mean-field PMP is stable $\boldsymbol{\theta}^*$, then the sampled PMP $\boldsymbol{\theta}^N$ will have its solutions be near a stable solution from the mean-field PMP with probability

$$\mathbb{P}[|J(\boldsymbol{\theta}^N) - J(\boldsymbol{\theta}^*) \geq s] \leq 4 \exp\left(-\frac{Ns^2}{K_1 + K_2 s}\right)$$

Although this paper relied heavily on proofs, this paper presents a very important formulation for ResNets, establishing solid and proofed entrypoints to the understanding of the mathematical theory behind ResNets.

# Appendices

# Appendix A

# Methodologies

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

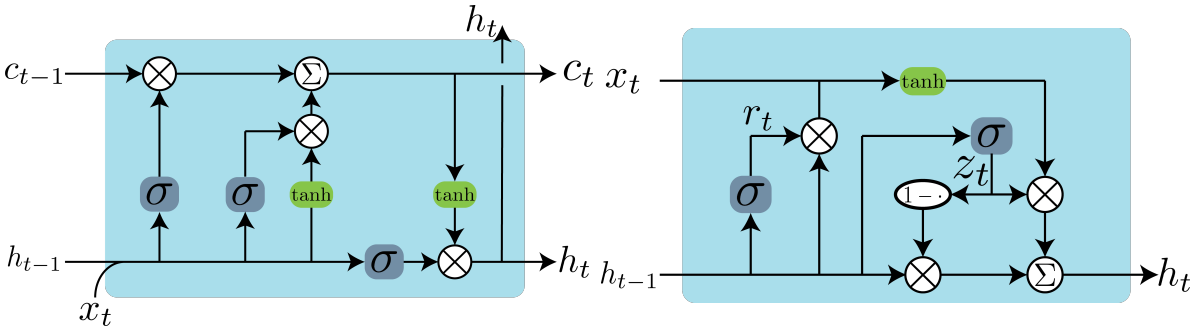


Figure A.1: A visual representation of the LSTM cell and the GRU cell

The LSTM and the GRU cell units are shown in Figure A.1, where $\sigma$ 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 Spectral Clustering

There are a lot of different clustering algorithms which may be used for unsupervised learning, but the issue is that, algorithms such as k-means or Gaussian mixture model clusterings assume that there is a compactness on the data. However, as seen in in Figure A.2. Therefore, spectral clustering is a methodology in which allows one to cluster data based on proximity/connectivity.

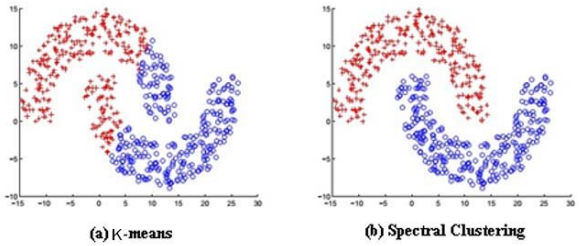


Figure A.2: A clustering result based on a (a) k-Means and (b) Spectral clustering

In order to do a spectral clustering, first, one needs to compute an adjacency matrix $A$, where, for the $i^{th}$ node and $j^{th}$ node,

$$A_{ij} = \begin{cases} w_{ij} & : \text{weight of the edge } (i,j)0 \\ & : \text{if no edge exists between } i \text{ and } j \end{cases}$$

Then one needs to obtain the Laplacian matrix $L$, which is defined as $L = D - A$, where D is the diagonal matrix of degrees

$$d_i = \sum_j w_{ij}$$

$$L_{ij} = \begin{cases} d_i & : \text{if } i = j \\ -w_{ij} & \text{if } (i,j) \text{ is an edge} \\ 0 & \text{if no edge between } i, j \end{cases}$$

Once the Laplacian matrix has been computed, then one can compute the eigenvectors and eigenvalues of $L$. The number of eigenvalues that equal 0 defines the quantity of clusters that are likely to be created, and to compute the algebraic connectivity of the graph. Then, a k-means clustering algorithm can be applied on the elements of the eigenvectors in order to identify the clusters, and see which datapoint (i.e. node) belongs to which class.

# 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.

[Chaudhari et al., 2017] Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2017). Entropy-sgd: Biasing gradient descent into wide valleys. In *International Conference on Learning Representations (ICLR)*.

[Chaudhari et al., 2018] Chaudhari, P., Oberman, A., Osher, S., Soatto, S., and Carlier, G. (2018). Deep relaxation: Partial differential equations for optimizing deep neural networks. *Research in the Mathematical Sciences*, 5(3):30.

[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.

[E et al., 2018] E, W., Han, J., and Li, Q. (2018). A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1):10.

[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.

[Goldt and Seifert, 2017] Goldt, S. and Seifert, U. (2017). Stochastic thermodynamics of learning. *Phys. Rev. Lett.*, 118:010601.

[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.

[Hansen and Hasan, 2015] Hansen, J. H. L. and Hasan, T. (2015). Speaker recognition by machines and humans: A tutorial review. *IEEE Signal Processing Magazine*, 32(6):74–99.

[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 sequential 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*.

[Shwartz-Ziv and Tishby, 2017] Shwartz-Ziv, R. and Tishby, N. (2017). Opening the black box of deep neural networks via information.

[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., 2015] Zhang, S., Choromanska, A. E., and LeCun, Y. (2015). Deep learning with elastic averaging sgd. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 685–693. Curran Associates, Inc.

[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.