SPEECH DENOISING USING NONNEGATIVE MATRIX FACTORIZATION AND NEURAL NETWORKS

BY

VINAY MADDALI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Assistant Professor Paris Smaragdis

ABSTRACT

The main goal of this research is to do source separation of single-channel mixed signals such that we get a clean representation of each source. In our case, we are concerned specifically with separating speech of a speaker from background noise as another source. So we deal with single-channel mixtures of speech with stationary, semi-stationary and non-stationary noise types. This is what we define as speech denoising. Our goal is to build a system to which we input a noisy speech signal and get the clean speech out with as little distortion or artifacts as possible. The model requires no prior information about the speaker or the background noise. The separation is done in real-time as we can feed the input signal on a frame-by-frame basis. This model can be used in speech recognition systems to improve recognition accuracy in noisy environments.

Two methods were mainly adopted for this purpose, nonnegative matrix factorization (NMF) and neural networks. Experiments were conducted to compare the performance of these two methods for speech denoising. For each of these methods, we compared the performance of the case where we had prior information of both the speaker and noise to having just a general speech dictionary. Also, some experiments were conducted to compare the different architectures and parameters in each of these approaches.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. Paris Smaragdis, for the encouragement and support that he has provided. His work was one of the main reasons I was interested in this field of machine learning in signal processing. I want to thank him for giving me the unique opportunity to work on this project and also for allowing me to work on a thesis under him during the senior year of my undergraduate studies here at the University of Illinois. I would also like to thank Minje Kim for being a mentor to me over the years. All the long discussions and brainstorming really helped me with my research work and also helped come up with ideas to solve problems. Also a special thanks to the rest of the computational audio lab members Johannes Traa, Yusuf Cem Subakan, Adam Miller and Ramin Anushiravani with whose help and advice my graduate school life here was much easier and more manageable.

Pursuing a master's degree here at ECE Illinois was the one of the best decisions I have ever made. The two years of experience have changed me as a person and the way that I tackle problems. My focus on machine learning applied on audio and speech has really taught me a lot in this field, and I hope I will be able to apply my advanced skills well when I go out into industry. It was a great experience working with my classmates Benjamin Delay, Michael Nute and Nathaniel Wetter Taylor on multiple assignments and projects. I would like to thank other people in ECE Illinois: Torin Kilpatrick, Sartaj Grewal and Bilal Gabula for all the help and guidance that I received from them during my six years here at the University of Illinois.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DESCRIPTION

The advance of computers and computational technology has opened up many fields of research and has led to development of applications that were not possible before. The use of multi-core processor CPUs and high performance GPUs have enabled us to use methods that were not feasible because of time and computation. One field which has greatly benefited from this is machine learning. The use of large datasets to build models for different applications has become widespread in the past decade or two. There are multiple research groups in universities across the world, research labs and even companies working on applications to solve various problems like speech recognition [1], object detection and classification in images [2], content extraction from audio and music [3], binary classification to get a simple yes or no on various topics, text language translation [4], and many more.

One such application of machine learning is to separate the various sources in a mixed signal. This research work is focused on the extraction of the speech signal from a noisy single-channel mixture. Speech recognition systems used mainly in phones, computers and gaming consoles might be very robust when it comes to word-to-word recognition, but a lot of them fail when there is noise in the background. It might be very difficult for the system to transcribe when speech and noise are mixed in the signal. This model can be used as an initial step so that the speech recognition step gets only clean speech as its input. This decreases the confusion that it might face when differentiating between speech and noise. We ideally want to build a robust model where we are free to denoise the speech of any random speaker with random noise in the background without having to change the training process and the model that we have already built.

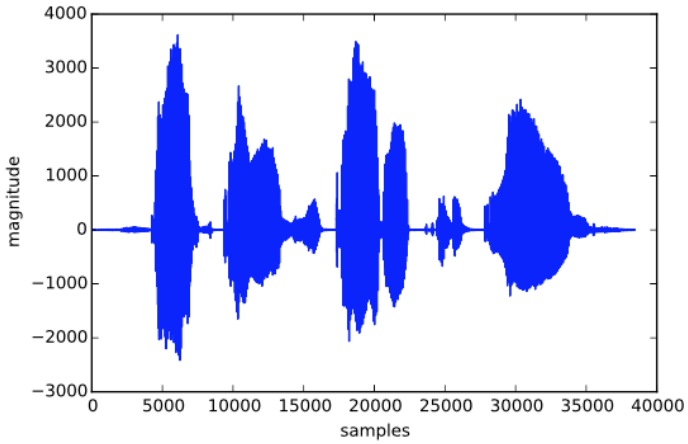


Figure 1.1: Time series representation of a signal containing a female speaker saying a few words taken from the TIMIT database [5]. The length of the signal is about 2 s.

1.2 SPEECH SIGNAL PROCESSING

In this research, the signal mixtures are taken in the frequency domain. Once the time domain signal of noisy speech, clean speech or noise is obtained, a short time fourier transform (STFT) is taken which gives the spectrogram representation of the signal. It is in the format shown in figures 1.1 and 1.2.

1.2.1 SAMPLING

We record the noisy mixture from a microphone which gives us an analog or continuous signal. This is then converted into the digital domain for processing by sampling this continuous signal. The sampling rate or the number of samples per second taken should be such that we prevent aliasing. It depends of the bandlimit of the signal which in the case of speech is generally 4 kHz. So the sampling rate used is 8 kHz as it makes the 4 kHz point the last one in the DFT. This way we do not have overlapping from the other replications in the DFT and hence there is no aliasing. So we have a signal $\mathbf{x[n]}$ that looks like:

$$\mathbf{x}[n] = [\mathbf{x[0]}, \mathbf{x[1]}, ..., \mathbf{x[N\text{-}1]}] \; for \; N samples \; taken \qquad (1.1)$$

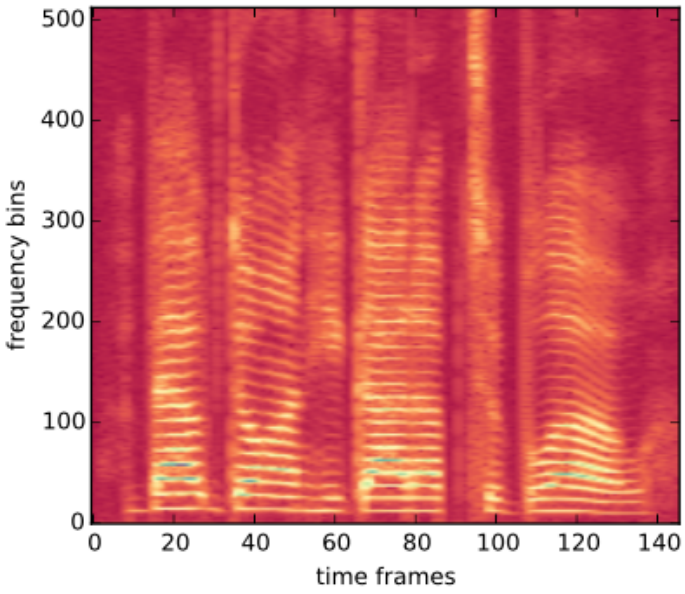


Figure 1.2: The corresponding spectrogram of the signal in figure 1.1 after taking the STFT. A 1024 point FFT was taken as we have 513 frequency bins represented on the y-axis. The x-axis shows 146 time frames based on the window size and overlap. The darker the color of the spectrogram, the greater the magnitude of speech at that point.

1.2.2 SHORT-TIME FOURIER TRANSFORM

STFT is a method in which we take the fast fourier transform (FFT) of the signal using windows taken on overlapping parts of the time series signal [6]. FFT is just a faster version of the discrete fourier transform (DFT):

$$\mathbf{X}[f] = \sum_{n=0}^{N-1} \mathbf{x}[n] e^{-j\frac{2\Pi f}{N}n} where\ n = 0, 1, ..., N-1 \tag{1.2}$$

The output we get from a DFT is such that the first $\frac{N}{2}$ samples are a complex conjugate of the last $\frac{N}{2}$ samples. Hence we can discard the second half of the DFT output and just consider the first $\frac{N}{2}$ samples. This is the reason why in figure 1.2, we have 513 or $\frac{N}{2}+1$ samples when we took a 1024 point FFT. The additional point is because we have samples from 0 to 1024 and hence they are symmetrical around the middle point $\frac{N}{2}+1$.

1.2.3 WINDOW ANALYSIS

The window that we use at each portion of the input **x[n]** to get the spectrogram is a square root Hann window:

$$\mathbf{w}[n] = \begin{cases} \sqrt{0.5 - 0.5\ cos\left(\frac{2\pi n}{N-1}\right)}, & 0 \leq n \leq N-1 \\ 0 \qquad\qquad\qquad\qquad , & else \end{cases} \tag{1.3}$$

The square root Hann window becomes zero at the transition edge and is most useful in applications of audio analysis and source separation where we need a good reconstruction of the speech without too much distortion or artifacts. We slide this window over the input time domain signal and perform an FFT each time to get the spectrogram:

$$\mathbf{X}_t[f] = \sum_{n=0}^{N-1} \mathbf{x}_t[n]\mathbf{w}[n]e^{-j\frac{2\Pi f}{N}n} where\ n = 0, 1, ..., N-1 \tag{1.4}$$

x[n] represents that portion of the input that we are considering over the current position of the sliding window.

The window is shifted across the time series signal by an amount called the hop size. The hop size used in all our test cases was 25% which means that for every time frame t that we take the FFT, we shift by $\frac{N}{4}$ samples over the signal.

1.2.4 RECONSTRUCTION

Once we process the noisy signal to get our clean output, we would like to analyze it using metrics using the time domain version of the signal. We obtain this by taking the inverse-STFT which uses the inverse-DFT equation:

$$\mathbf{x}[n] = \frac{1}{N}\sum_{f=0}^{N-1} \mathbf{X}[f]e^{j\frac{2\Pi f}{N}n} where\ f = 0, 1, ..., N-1 \tag{1.5}$$

So we reconstruct the output signal using the inverse-STFT and also take into consideration the overlap of samples that we applied the window over:

$$\mathbf{x}[n] = \sum_{t=0}^{T-1} \left(\mathbf{F}^{-1}(\mathbf{X}_t[f]) \odot \mathbf{w}[n] \right) * \delta(n - t\frac{N}{4}) \tag{1.6}$$

where $\odot$ implies elementwise multiplication and $*$ is convolution. $\delta(n-t\frac{N}{4})$ is the Kronecker delta shifted by $t\frac{N}{4}$ samples which accounts for the overlap of samples taken while performing the FFT operation with the sliding window. When we take the STFT of the signal, we get a complex valued spectrogram as the result. But we do not take phase into account in any of the algorithms we used as we deal mainly with extracting the spectral and temporal features of the audio signal and build models based on that. So we save the phase of the spectrogram:

$$P(\mathbf{X}[f]) = \frac{\mathbf{X}[f]}{|\mathbf{X}[f]|} \tag{1.7}$$

where $abs(\mathbf{X}[f])$ is the absolute valued spectrogram where we just take into account the magnitude of the complex values. Before we perform the inverse-STFT in equation 1.6 we apply this saved phase to the processed clean signal:

$$\mathbf{X}[f] = \tilde{\mathbf{X}}[f] \odot P(\mathbf{X}[f]) \tag{1.8}$$

where $\tilde{\mathbf{X}}[f]$ is the real-valued spectrogram of the clean speech output from the algorithm.

1.3 TRAINING CASES

All results shown in this thesis were based mainly on three cases. These cases are different from each other in terms of the training information and its similarity to the test cases.

1.3.1 FULLY-SUPERVISED

First we consider the fully-supervised case, where we use random utterances of a speaker in training and test on a sentence of the same speaker. Similarly, we train on the same noise type as we use in the test signal. This way the training process builds accurate speech and noise models that form a good

representation of the sources in the noisy test signal.

1.3.2 SEMI-SUPERVISED

Second is the semi-supervised case where we train on the speech of the same speaker as in the test, but the noise training will be something random. So the trained model contains a good representation of the speech but may not have an accurate idea of what the noise might be.

1.3.3 UNSUPERVISED

Third is the unsupervised case where we build a model based on no prior information of the test speaker's speech or the noise in the background. This makes it hard for the system to differentiate between speech and noise in general. For that reason we use speech of some random speakers in training so that we can teach the model what speech in general sounds like, not the specific speaker. This way we can test on any speaker keeping the training process fixed. The noise as in the previous case can be anything random for training. The last unsupervised case is what we would like to build ideally as it is a very good real-world case.

1.4 TEST CASES

We modeled two scenarios for all test cases. In one scenario, we do denoising offline, which means that we get the signal as a whole and process all the time frames together. In the other, a more real-world scenario, we do denoising of the speech online or in real-time. This case is applicable to speech recognition systems where the speaker is constantly talking and system is trying to transcribe in real-time. This is a more difficult case of denoising especially in semi-supervised and unsupervised cases since we do not have all the information about one or both sources in the test signal. So we cannot build an accurate enough model for the unknown sources in real-time. We elaborate measures taken to prevent this overfitting while training online for each algorithm in the later sections. So the maximum number of frames that can be processed at a time to maintain this real-time scenario depends on

the sampling rate and window hop-size as well. Ideally, a real-time speech based system should be able to process signals at least every second.

1.5 DATASETS

The speech examples used for training speech models and generating test signals were obtained from the TIMIT Database [5]. This database contains speech of different speakers, specifically 10 sentences of each speaker. For training purposes, we used a randomly chosen 9 of these sentences so that we can test on the left out 10th utterance. This forms a case of cross-validation testing where we split the dataset into training and test and assign the data samples randomly each time. For unsupervised cases, we had a big training set where we used all 10 utterances of say 10 or 20 speakers. This was still reasonable because we tested on the sentences of speakers different from the ones in training. There is very little overlap between the content of speech in the different speakers' dataset, so there is no bias involved in training the unsupervised case.

Different noise types have been used to account for all the real-world noises and results were shown as an average performance across all these noise types. To simulate noisy environments in the background of speech, we used noise datasets from [7]. These contain a mixture of stationary, semi-stationary and non-stationary noise signals. Stationary noise is the easiest to source-separate as there is a specific pattern in its structure that can be easily learned as features looking at the spectrogram. Semi-stationary and non-stationary noise types are more difficult to separate as it is hard to learn features based on their training signal. These noise types are more random and can have different spectral and temporal structure in their spectrogram. Having just a fraction of this type of noise in training might not accurately represent what is present in the test signal. The test signal could be having the same noise type as the trained features, but the extract and content specifically might be slightly different as there are different variations to this noise type. But in case of stationary noise, the features learned in training are adequate to represent the noise in the test signal due to uniformity in spectral and temporal patterns.

1.6 IDEAL SOURCE SEPARATION

Figure 1.3 shows an input noisy mixture, which is the spectrogram on the left, to the model trained by a source separation algorithm. Ideally we should get the clean spectrogram out as shown in the center figure which corresponds to the exact same speech as in the noisy mixture without any distortions or alterations. In the real-world case, this is probably not possible as there will be some interference from the other sources and some artifacts introduced in the speech. This is because we cannot achieve near perfect training models in the real-world. In this thesis, we indicate the performances of each algorithm and show how one is better than the other.

1.7 EVALUATION METRICS

To judge the performance of the algorithms used and for comparison purposes, we used the BSS-Eval Toolbox [8]. This toolbox deals with time series signals. We modeled each processed signal $\hat{s}$ of the output from equation 1.6 to be composed of the following [9]:

$$\hat{s} = s_{clean} + a_{interf} + a_{artif} \tag{1.9}$$

where s_{clean} is the clean source signal which we compare with, a_{interf} is the mixture of all unwanted or irrelevant sources which in our case is the background noise and a_{artif} is the artifacts like musical sound and other kinds of noise introduced in the source signal.

The ratios that we have measured for all the results of the source separations algorithms are:

Sources to Interference Ratio:

$$SIR = 10 \log_{10} \frac{\|s_{clean}\|^2}{\|a_{interf}\|^2} \tag{1.10}$$

Sources to Distortion Ratio:

$$SDR = 10 \log_{10} \frac{\|s_{clean}\|^2}{\|a_{interf} + a_{artif}\|^2} \tag{1.11}$$

Sources to Artifacts Ratio:

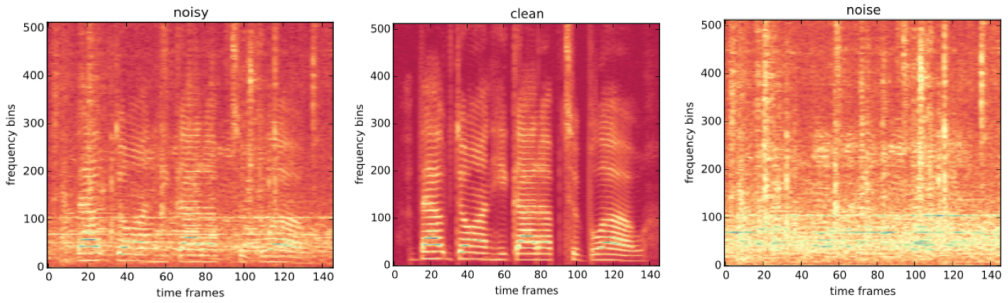


Figure 1.3: The spectrogram on the left is one of a noisy signal which is a mixture of a 2 s speech signal and some *casino* background noise [7]. The spectrogram in the middle is the ideal clean speech source in the mixture and the one on the right is the noise that was mixed with this speech signal.

$$SAR = 10 \log_{10} \frac{\left\| s_{clean} + a_{interf} \right\|^2}{\left\| a_{artif} \right\|^2} \tag{1.12}$$

The SIR is a good measure to evaluate the source separation quality. It gives us an idea of how much the source we are talking about has been separated from all the other sources. The SDR and SAR give us an idea of how much the algorithm has distorted the speech signal and varied it by introducing artifacts. There is always a tradeoff between the SIR and SDR/SAR figures, as when source separation quality is high, we have a more distorted or altered speech signal. When we have a better quality output speech signal, we can probably still hear some parts of the other sources mixed in with the source in context.

1.8 TOOLS

The algorithm for the NMF based approach was implemented mainly in MATLAB and used a CPU with a 2.2 GHz Intel quad-core i-7 processor. Initially the fully-connected neural networks were implemented in MATLAB, but we later moved to using third-party tools for accuracy and handling bigger datasets better. We mainly used two tools for making neural network models: Caffe [10] and Keras [11].

Caffe is a vision-based deep learning software made at the Berkeley Vision and Learning Center (BVLC) and is focused mainly on solving computer

vision problems. We used this software to make models for audio and solve regression problems, which in our case were speech denoising. We generated datasets and did the initial setup in MATLAB, then fed the data and parameters to be run on the Caffe framework. We used a Tesla K20 GPU during the training process as neural network training requires much computation.

Keras is a Theano-based deep learning library. It can be used only on Python and requires the use of Theano software [12], [13] as well. Theano is a Python library that is useful for multi-dimensional array processing and has been widely used for deep learning research. Keras is a tool that is built on a layer above Theano and has its own functions that implement any neural network architecture necessary without having to build upon the algorithm from the ground up as in Theano. We used a Tesla K40 GPU to train the neural networks designed on this software.

1.9 CONTRIBUTIONS

In previous work on source separation problems, we either assume training information of the speaker or the noise type in the background. Most of these models are offline based and need the whole signal in order to perform the denoising.

In this thesis we have used two methods for denoising: nonnegative matrix factorization and neural networks. We have shown the performance of the unsupervised case where we do not have any prior training information of the speaker or noise type in the signal for both these methods. The degradation in performance for this case is not very drastic compared to that of the fully-supervised case. We have also proposed how to perform these methods in real-time where the denoising can be done frame by frame or for a group of frames together. Even this case had performance that was not too much worse than the offline method. Hence we have developed two robust models that can denoise any noisy signal, regardless of the speaker and the noise type, and in real-time. Also we have used various noise examples that are different from each other and both genders of speakers to generate the results. This shows that our algorithms can be used in a real-time voice recognition device in order to perform the prior denoising to improve the speech recognition accuracy.

CHAPTER 2

NONNEGATIVE MATRIX FACTORIZATION

2.1 NMF INTRODUCTION

The first method we used to solve this problem of speech denoising is one using nonnegative matrix factorization (NMF). It assumes the input data to be nonnegative and imposes nonnegativity on the features it produces as well. NMF has been used to solve source separation problems in applications of music [14] and multi-speaker separation [15].

In this section, we introduce the theory behind the NMF algorithm in general and then propose a modification of this in order to develop a real-world system that is robust enough to denoise any signal without any prior information of the test speaker or the noise in the test signal. Additionally we show that the denoising performance of this method is close enough to methods that have these additional requirements, thereby making it a viable alternative. In addition to demonstrating the performance of this approach, we also illustrate some of the tradeoffs that one might find in the parameters involved and how they influence performance.

2.1.1 DEFINITION

It is a method in which we can decompose an input matrix into two constituent matrices. Hence given an input matrix $\mathbf{V}$, we can decompose it into:

$$\mathbf{V} \approx \mathbf{W} \cdot \mathbf{H} \tag{2.1}$$

This method imposes a constraint of nonnegativity on all the matrices. Given a matrix $\mathbf{V}$ with all nonnegative values, this decomposes it into two nonnegative matrices $\mathbf{W}$ and $\mathbf{H}$. NMF is a method of analyzing or decom-

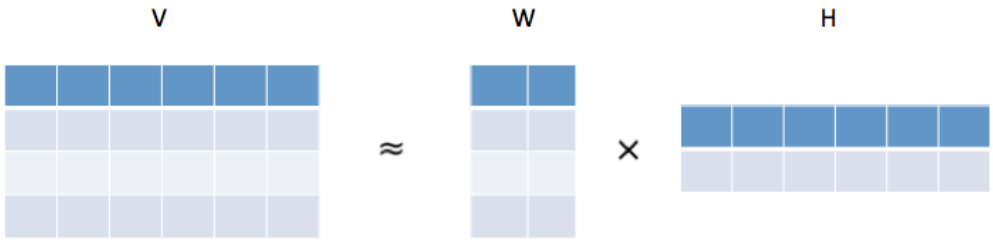


Figure 2.1: An input non negative matrix **V** can be decomposed into its column matrix **W** with r columns and row matrix **H** with r rows.

posing multivariate data to get its constituent features [16]. Suppose we consider n dimensional vectors in our case; we have an input matrix **V** of size $n \times m$, where m is number of tokens or samples in the dataset. NMF decomposes this into **W** of size $n \times r$ and **H** of size $r \times m$ as shown in figure 2.1, where r is called the number of basis vectors and controls the approximation in equation 2.1. **W** is a summarization of the columns of **V** and hence is an approximation of the vertical information of the matrix, and **H** summarizes the rows and hence is an approximation of the horizontal information of **V**. r controls this level of summarization and when $r = m$, the constituent matrices **W** and **H** might not be very helpful even though the approximation might be good [17]. For this reason, we usually keep r smaller than n and m.

2.1.2 COST FUNCTION

Since we decompose a matrix **V** into matrices **W** and **H**, we want to get as close an approximation for equation 2.1 as possible. This is done by trying to minimize the equation for the cost function at every iteration. The different cost functions that can be used to solve NMF are:

EUCLIDIAN DISTANCE

This just takes the Euclidian distance between the original matrix **V** and the target matrix (**W**·**H**) in terms of getting the difference between corresponding

elements in each matrix.

$$C = \|\mathbf{V} - \mathbf{WH}\|^2 = \sum_{ij} (\mathbf{V}_{ij} - (\mathbf{WH})_{ij})^2 \tag{2.2}$$

where i, j are the row and column indices respectively of each matrix where $i \in [1, n]$ and $j \in [1, m]$.

KULLBACK-LIEBLER DIVERGENCE

This is more a non-symmetric measure of how much two matrices differ from each other. It measures the relative entropy between the two matrices. In the case of NMF, the cost function becomes the divergence equation:

$$C = D(\mathbf{V}\|\mathbf{WH}) = \sum_{ij} \left(\mathbf{V}_{ij} \log \frac{\mathbf{V}_{ij}}{(\mathbf{WH})_{ij}} - \mathbf{V}_{ij} + (\mathbf{WH})_{ij} \right) \tag{2.3}$$

where i, j are the row and column indices respectively of each matrix where $i \in [1, n]$ and $j \in [1, m]$.

IKATURA-SAITO DIVERGENCE

Another function which has commonly been used for speech and audio is the Ikatura-Saito (IS) divergence function [18]:

$$C = D_{IS}(\mathbf{V}\|\mathbf{WH}) = \sum_{ij} \left(\frac{\mathbf{V}_{ij}}{(\mathbf{WH})_{ij}} - \log \frac{\mathbf{V}_{ij}}{(\mathbf{WH})_{ij}} - 1 \right) \tag{2.4}$$

2.1.3 SIMILARITY TO PCA

Principal component analysis (PCA) is a method in which we get features from a data matrix similar to the matrix $\mathbf{V}$ that we have been talking about in NMF. PCA gives us eigenvectors of this matrix and can be used to project into a different feature space where the dimensionality of the vectors, which was previously n, changes.

$$\mathbf{H} = \mathbf{A} \cdot \mathbf{V} \tag{2.5}$$

where A is the feature matrix containing the eigenvectors of the input matrix V and H is the new feature space with m vectors of different dimensions.

As can be seen, this is similar to the NMF equation if we take the Moore-Penrose inverse of the W matrix which is denoted in equation 2.6 by $\mathbf{W}^+$.

$$\mathbf{H} = \mathbf{W}^+ \mathbf{V} \tag{2.6}$$

Since PCA models the feature matrix by stacking the eigenvectors of the input together, it imposes the orthogonality constraint. PCA hence tries to solve the same cost function with the orthogonality constraint [17].

2.1.4 KL-NMF ALGORITHM

This uses the Kullback-Liebler (KL) Divergence equation for the cost function and hence we take the derivative of equation 2.3 with respect to **W** and **H** to find the local minima of this function. We use gradient descent in order to update the **W** and **H** at every iteration to get as close an approximation as possible.

GRADIENT DESCENT

The gradient descent equation is such that we move on the steep part of the convex curve and reach the point of local minimum:

$$x^{l+1} \leftarrow x^l - \eta \nabla f(x^l) \tag{2.7}$$

where x^{l+1} is the variable at the next iteration and the current value of the variable x^l is being used to update the value at every iteration. Matrices **W** and **H** are updated this way. η is the step-size with which we move on the curve and ∇ implies gradient with respect to x.

UPDATE EQUATIONS

$$(\mathbf{W}, \mathbf{H}) = \ _{\mathbf{W},\mathbf{H} \geq 0} D(\mathbf{V} \| \mathbf{W}\ \mathbf{H}) \tag{2.8}$$

$$\mathbf{W} \leftarrow \mathbf{W} \odot \left(\frac{\frac{\mathbf{V}}{\mathbf{W} \cdot \mathbf{H}} \cdot \mathbf{H}^\top}{\mathbf{1} \cdot \mathbf{H}^\top} \right) \tag{2.9}$$

$$\mathbf{H} \leftarrow \mathbf{H} \odot \left(\frac{\mathbf{W}^\top \cdot \frac{\mathbf{V}}{\mathbf{W} \cdot \mathbf{H}}}{\mathbf{W}^\top \cdot \mathbf{1}} \right) \tag{2.10}$$

where $\odot$ implies element-wise multiplication and the fraction implies element-wise division. These updates are repeated until we reach convergence which will be a local minimum of the divergence function. This point is not guaranteed to be a global minimum since we have a non-convex function, but in practice it often finds a usable solution.

2.1.5 APPLICATION IN MUSIC AND AUDIO

NMF has been used widely in audio applications to solve problems of single and multi-channel source separation problems [19]. The separation could be sources of different musical instruments, singing voices, speech of various speakers or speech denoising as in our case.

Since NMF decomposes the input nonnegative matrix $\mathbf{V}$ into its column summarization $\mathbf{W}$ and row summarization $\mathbf{H}$, we can use an absolute valued spectrogram of the mixture or source as the input to the NMF algorithm. $\mathbf{W}$ will give us an approximation of the spectral information and $\mathbf{H}$ will give us an approximation of the temporal information or time activations. Figure 2.3 shows a spectrogram of four piano notes played together decomposed into its constituent spectral and time activation matrices. We took $r = 4$ as we have four notes in the excerpt. As explained before, $\mathbf{W}$ shows the frequency or spectral information of each of the notes individually as each column or basis vector. Similarly $\mathbf{H}$ shows the respective time activation of each note in the respective row [17]. If we want to separate the sound of a single note, we can do:

$$\mathbf{V}_n = W_n \cdot H_n \tag{2.11}$$

where n corresponds to the basis vector of the specific note.

The basis vector representation is a little more complex for speech and noise. The speech source will correspond to a collection of these basis vectors and a single basis vector might not be able to explain much about either source. So we allocate a certain number of basis vectors for speech and noise separately and source-separate using only those basis vectors. So n from

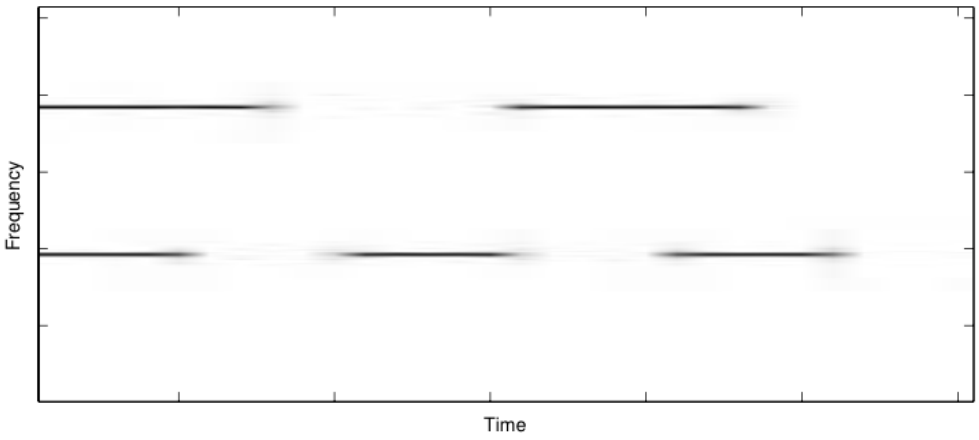


Figure 2.2: This is a simple spectrogram from that shows activity of mainly two frequencies [17].

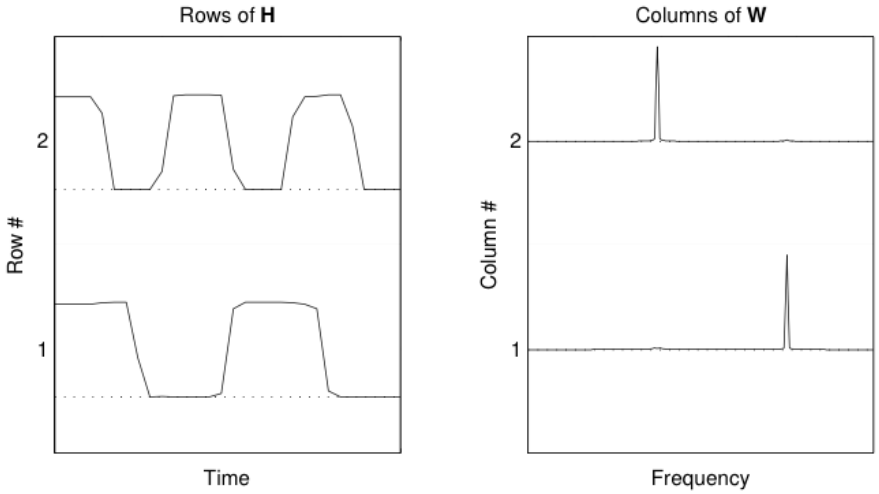


Figure 2.3: This is the NMF decomposition of the spectrogram in figure 2.2 with $r = 2$ where the columns of the $\mathbf{W}$ matrix show the two frequencies and the rows of the $\mathbf{H}$ matrix show the respective time activations [17].

equation 2.11 will correspond to a group of basis vectors allocated for the desired source.

2.2 MODIFIED NMF ALGORITHM

Nonnegative matrix factorization (NMF) and probabilistic latent component analysis (PLCA) have been used for source separation tasks in a variety of situations [19], [20], [15], with a very common use being that of speech denoising [21]. Although these methods can perform well, they often necessitate that the speaker in the provided mixtures is known a priori or even the noise type and that an appropriate model for that source has already been constructed. Furthermore, many of these approaches are offline which when combined with the previous requirement makes for a system that is not amenable to real-world use. Recently we have seen the introduction of systems that do not require a priori knowledge of the speaker in the mixture [22] and models that can operate in an online manner [7, 23, 18]. In this research we propose an algorithm that combines these features to create a system that is well suited for real-world deployment since it does not require advance knowledge of the speaker or the noise in a mixture; it can operate in an online manner which facilitates a real-time implementation, and it is efficient enough to run in real-time without excessive hardware requirements.

2.2.1 FULLY-SUPERVISED

This is the case where we have prior information of the test speaker's speech and of the noise type in the background of the test signal as well. In the case of NMF, this is obtained by performing the KL-NMF algorithm using some training signals. This equation for NMF is roughly equivalent to PLCA [24] which has been used for denoising in more real-world cases [7], [23]. When we talk about having prior information of the speaker, we refer to the model having some knowledge of what the source might sound like. The spectral features of the source which are given by $\mathbf{W}$ can be used in our case. We call this is a speech or noise dictionary based on the source. We use KL-NMF for training and testing alike as we want to reach the same local minimum. Hence in the training process, we learn $\mathbf{W}_S$, which is the

speech dictionary, and $\mathbf{W}_N$, which is the noise dictionary, and discard the respective time activations $\mathbf{H}_S$ and $\mathbf{H}_N$, which are irrelevant in our case due to the speaker having to say different things in training and test. The test process tried to solve the equation:

$$\mathbf{H} = \min_{\mathbf{W},\mathbf{H}\geq 0} D\left(\mathbf{V}||[\mathbf{W}_S, \mathbf{W}_N] \cdot \mathbf{H}\right) \tag{2.12}$$

where $\mathbf{W} = [\mathbf{W}_S, \mathbf{W}_N]$ is the fixed dictionary from training and hence we do not update this during the test process. The activations $\mathbf{H}$ can be then represented as $\begin{bmatrix} \mathbf{H}_S \\ \mathbf{H}_N \end{bmatrix}$, that is in parts that contain the activations $\mathbf{H}_S$ of the speech bases $\mathbf{W}_S$ and the activations $\mathbf{H}_N$ of the noise bases $\mathbf{W}_N$. The clean speech magnitude spectrogram can then be recovered as $\hat{\mathbf{V}}_S = \mathbf{W}_S \cdot \mathbf{H}_S$. Using the phase of the original sound we can transform $\hat{\mathbf{V}}_S$ back to the time domain and obtain a speech waveform. Instead of the reconstruction shown above, we can also use alternative approximations:

$$\hat{\mathbf{V}}_S = \mathbf{V} \odot \frac{\mathbf{W}_S \cdot \mathbf{H}_S}{\mathbf{W} \cdot \mathbf{H}} \tag{2.13}$$

2.2.2 SEMI-SUPERVISED

In this research we also use KL-NMF in the context of the semi-supervised denoising method [7]. We do so because we want to deploy this model in situations where the noise in the recordings is not known and needs to be automatically estimated. In this case, we can use KL-NMF to build a speaker dictionary $\mathbf{W}_S$ using as input the magnitude spectrogram $\mathbf{V}_S$ of a speech training example. Once this speaker dictionary is obtained we can fix it and decompose a mixture containing the voice of that speaker plus noise. Thus, for a mixture magnitude spectrogram $\mathbf{V}$, we estimate the activations $\mathbf{H}$ and a noise dictionary $\mathbf{W}_N$ by solving the problem:

$$(\mathbf{W}_N, \mathbf{H}) = \min_{\mathbf{W},\mathbf{H}\geq 0} D\left(\mathbf{V}||[\mathbf{W}_S, \mathbf{W}_N] \cdot \mathbf{H}\right) \tag{2.14}$$

Estimating $\mathbf{W}_S$ and $\mathbf{H}$ can be easily done by a simple modification of the update equations 2.9 and 2.10.

2.2.3 UNIVERSAL SPEECH MODEL

We now consider the unsupervised case where we have no training information of the particular speaker or noise type in test. Since we are using a model of speech with no prior information of the speaker, we can apply what is a called a universal speech model (USM) obtained by training on multiple male and female speakers. By allocating multiple basis vectors for each speaker and training on M speakers using KL-NMF, we obtain a USM by concatenating each speaker's dictionary $\mathbf{W}_i$ to a larger USM dictionary:

$$\mathbf{W}_{USM} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_M] \tag{2.15}$$

Our speech dictionary now contains information from M different speakers with their basis vectors grouped together in blocks.

2.2.4 BLOCK SPARSITY

As shown in [22], using all this information to model the time activations of one speaker might cause overfitting leading to a degraded speech output. To alleviate this we can impose a sparsity constraint on the activation matrix in order to use information of the speaker closest to the one in the signal. More specifically, we use block sparsity where the sparsity constraint is imposed on each block, i.e., on the set of basis vectors representing each speaker. This forces the fitting process to predominantly choose the basis vectors of a speaker who is most similar sounding to the one in the input signal. This is enforced through the following modified cost function:

$$(\mathbf{W}, \mathbf{H}) =_{\mathbf{W},\mathbf{H}\geq 0} D(\mathbf{V}||\mathbf{W}\cdot\mathbf{H}) + \lambda\Omega(\mathbf{H}_S) \tag{2.16}$$

where the Ω is the penalty imposed to induce sparsity in the speech activation matrix $\mathbf{H}_S$ [22]. The parameter λ can be adjusted to control how much energy we draw from one speaker. A large λ uses only one speaker's basis vectors. The separation from noise tends to be good but it introduces a lot of artifacts into the speech. A smaller λ allows the fitting to use all the speakers' basis vectors. This induces fewer artifacts but the noise is not removed as well since it can be modelled through various bases from other speakers. Tuning the parameter λ allows us to optimize this trade-off and can be appropriately

adjusted. This is shown in figure 2.4. The three figures depict how the tuning of λ affects the source separation quality. In [22] and [18] we see the use of a $\log/l1$ penalty for block sparsity as convergence can be achieved using multiplicative updates which fits well with our model. The sparsity penalty is defined as:

$$\log/l1 = \sum_{i=1}^{M} log(\epsilon + ||\mathbf{H}_i||_1) \tag{2.17}$$

where $\mathbf{H}_i$ is the subset of $\mathbf{H}$ that corresponds to speaker i.

ADDITIONAL WEIGHT

We make one more addition to the USM model in order to achieve better speech separation from the noise. At each iteration of training we add a constant nonnegative value w to all the elements of $\mathbf{H}$ that correspond to the noise bases. This allows us to specify how much energy from the mixture we want to allocate to the noise model or the speech model. Like the sparsity constraint, this also introduces artifacts in the speech since it might allocate some of the speech energy to the noise, but it dramatically improves the separation when properly tuned.

2.2.5 BKL-NMF ALGORITHM

Since we model the NMF such that the basis vectors are divided into blocks based on many speakers or a single speaker used in the training process, we refer to this modified approach as Block Kullback-Liebler nonnegative matrix factorization (BKL-NMF). We use the BKL-NMF algorithm for both training and test purposes. The training algorithm is shown in algorithm 1 as pseudocode. This algorithm is performed for the M speakers we use in training. In supervised and semi-supervised cases, we have $M = 1$ as we train on the same speaker's speech. In the unsupervised case, M is greater than 1, generally in the order of 10 to 20. Our input is the spectrogram of the training excerpt of the speaker in context. We randomly initialize speech dictionary $\mathbf{W}$ and speech activations $\mathbf{H}$ since we do not have any knowledge of them in the beginning. We then perform equations 2.9 and 2.10 until we reach an optimum or a point of local minima. In the end, we concatenate

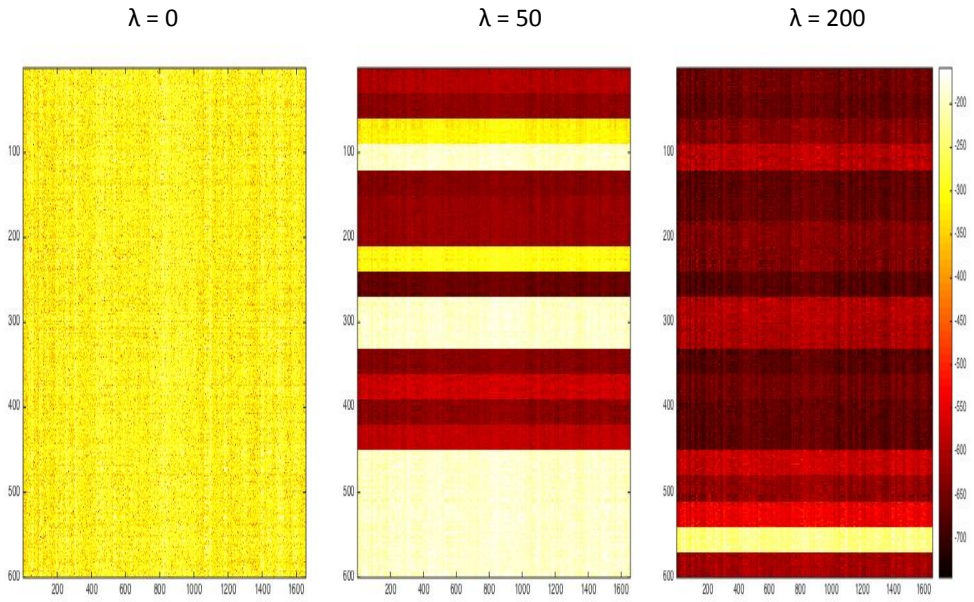


Figure 2.4: This figure shows the sparsity imposed in the $\mathbf{H}_S$ matrix. This is the time activations matrix when we use a universal speech model for training with multiple speakers. The color scale is such that the darker the region in the matrix, the more sparse the matrix. There are three cases: 1. On the extreme left, $\lambda = 0$. There is no sparsity imposed and hence every speaker's basis vectors have the same energy. 2. The middle figure with $\lambda = 50$, where the energy is drawn from the basis vectors of a group of speakers. This case balances the source separation and artifacts tradeoff. 3. The rightmost figure with $lambda = 200$ picks up the basis vectors of only one speaker. In this test case, we used a female speaker and the USM contains 10 male speakers followed by 10 female speakers in training. Hence the algorithm picked up a female speaker's dictionary. This is the case of high λ which leads to high source separation but more artifacts in the speech.

Algorithm 1 BKL-NMF Training

for $i = 1 : M$ **do**

inputs $\mathbf{X}_i$

initialize random $\mathbf{W}_i$, $\mathbf{H}_i$

repeat

$\mathbf{R}_i = \frac{\mathbf{X}_i}{\mathbf{W}_i \cdot \mathbf{H}_i}$

$\mathbf{W}_i = \mathbf{W}_i \odot \left(\mathbf{R}_i \cdot \mathbf{H}_i^{\top}\right)$

renormalize the columns of $\mathbf{W}_i$ to 1

$\mathbf{H}_i = \mathbf{H}_i \odot \left(\mathbf{W}_i^{\top} \cdot \mathbf{R}_i\right)$

renormalize the rows of $\mathbf{H}_i$ to 1

until convergence

end for

$\mathbf{W}_{USM} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_M]$

these training matrices in unsupervised cases to obtain the universal speech model $\mathbf{W}_{USM}$.

The test algorithm is slightly different as we keep the speech dictionary fixed and learn the activations and also the noise dictionary in case of the semi-supervised and unsupervised. This is shown in algorithm 2. We have inputs $\mathbf{X}$, which is the input noisy test spectrogram to be denoised and $\mathbf{W}_{USM}$, which is the universal speech dictionary or simple single speaker training dictionary obtained from algorithm 1. We initialize random positive speech and time activation matrices $\mathbf{H}$ and the noise dictionary $\mathbf{W}_N$ in case of semi-supervised and unsupervised cases. So we perform the update for $\mathbf{H}$ similar to training, but in the case of the dictionary, only the noise dictionary is updated for the related cases. In the case of unsupervised alone, we impose the $log/l1$ sparsity constraint with a chosen value of λ on the $\mathbf{H}$ matrix. We do this individually M times on each block of $\mathbf{H}$ with the basis vectors allocated for each speaker included in the USM. We then add the weight to the noise block of $\mathbf{H}$ to improve the source separation performance. The noise dictionary $\mathbf{W}_N$ is then updated based on the new values of $\mathbf{H}$.

Algorithm 2 BKL-NMF Test

inputs $\mathbf{X}$, $\mathbf{W}_{USM}$
if supervised **then**
  input $\mathbf{W}_N$ from training
else
  initialize random $\mathbf{W}_N$
end if
initialize random $\mathbf{H} = \begin{bmatrix} \mathbf{H}_S \\ \mathbf{H}_N \end{bmatrix}$
repeat
  $\mathbf{W} = [\mathbf{W}_{USM}, \mathbf{W}_N]$
  $\mathbf{R} = \frac{\mathbf{X}}{\mathbf{W} \cdot \mathbf{H}}$
  $\mathbf{H} = \mathbf{H} \odot (\mathbf{W}^\top \cdot \mathbf{R})$
  if unsupervised **then**
    for $i = 1 : M$ **do**
      $\mathbf{H}_{S_i} \leftarrow \mathbf{H}_{S_i} \frac{1}{1 + \frac{\lambda}{\epsilon + ||\mathbf{H}_{S_i}||_1}}$
    end for
  end if
  $\mathbf{H}_N = \mathbf{H}_N + w$
  if semi-supervised or unsupervised **then**
    $\mathbf{A} = \mathbf{W} \odot (\mathbf{R} \cdot \mathbf{H}^\top)$
    $\mathbf{W}_N = \mathbf{A}_N$
    renormalize columns of $\mathbf{W}_N$ to sum to 1
  end if
until convergence

Finally we reconstruct the clean speech spectrogram $\mathbf{V}_S$ using:

$$\mathbf{V}_S = \mathbf{W}_{USM} \cdot \mathbf{H}_S \tag{2.18}$$

In the unsupervised case, the sparsity imposed on $\mathbf{H}_S$ ensures that only a few basis vectors corresponding to a specific speaker or selection of speakers are activated when performing this reconstruction.

2.2.6 ONLINE METHOD

In order to perform online or real-time denoising we need to process input magnitude spectrum frames as they appear. Therefore, once we train a USM or a single speaker's speech dictionary we can apply it on every input frame or a group of frames of the mixture spectrogram individually. In semi-supervised and unsupervised cases, we need to appropriately update a noise model $\mathbf{W}_N$, thus estimating the noise dictionary and its respective activations.

Updating the noise dictionary in the two cases might not be as straightforward as in the offline method. This is because we are trying to fit some information about noise using just a single spectrogram frame. We need to allocate some number of basis vectors to model the noise dictionary accurately, but modeling this based on a single frame will lead to overfitting. So we can model the noise dictionary $\mathbf{W}_N$ based on not only the current frame but some previous frames in the past. This way we are fitting the dictionary based noise information from many samples which will give us a much more accurate representation of the noise source. Also, instead of creating a new noise dictionary while denoising each frame, we can update the $\mathbf{W}_N$ of the previous frame using some weights:

$$\mathbf{A} = (1-\mu)(\mathbf{W} \odot (\mathbf{R}_G \cdot \mathbf{H}_G^\top)) + \mu(\mathbf{W} \odot (\mathbf{R}_B \cdot \mathbf{H}_B^\top)) \tag{2.19}$$

We denote the time indices of the input frames currently being denoised by $\mathbf{G}$. In addition to the current input frames $\mathbf{V}_G$, we maintain a running buffer $\mathbf{B}$ containing the b frames directly preceding $\mathbf{V}_G$. The variable $\mathbf{H}_G$ contains the basis activations for the current frames and $\mathbf{H}_B$ contains the activations for the frames maintained in the buffer. Operating on this set prevents potential overfitting as we are estimating the basis vectors of the

noise dictionary using multiple time frames and not just one or a small number of frames from the input [23]. μ here is the weight we decide to impose on the information from our buffer $\mathbf{B}$ and is between the values of 0 and 1. Hence we apply a weight of $1-\mu$ on the current frame and its activation $\mathbf{H}_G$.

Algorithm 3 describes the BKL-NMF algorithm taking the buffer $\mathbf{B}$ into consideration. $\mathbf{H}_G$ is split into the activations that relate to the speakers $\mathbf{H}_{GS}$ and the noise $\mathbf{H}_{GN}$. The speaker activations are furthermore split into sets that correspond to each speaker $\mathbf{H}_{GS_i}$ in the unsupervised case. The activations for the previous b frames are saved in matrix $\mathbf{H}_B$ after being calculated in preceding steps. This is the same process described in Algorithm 2, but at every step instead of using all frames of the input noisy signal together, we only denoise the current g frames and use an additional b frames from the running buffer. We impose the weights μ and $1-\mu$ as described above to model the noise dictionary based on current frames and the previous frames in the buffer. At the end of this process we will have estimated an updated noise model and an estimate of the speaker's magnitude spectrum as: $\mathbf{V}_{SG} = \mathbf{W}_{USM} \cdot \mathbf{H}_{SG}$.

The main online BKL-NMF algorithm is represented in Algorithm 4. This does the denoising of any noisy speech signal in real-time and can be used for any of the training cases described earlier. Once we get the dictionary from training, we can denoise frame by frame or a bunch of frames together based on how fast we want our model to work. We skip denoising the first b frames of the signal since we need to maintain a buffer of that size to model the activations and the noise dictionary as well in some cases. We normalize the current spectrogram $\mathbf{V}_G$ containing the frames being denoised to maintain a probability based model like in [7]. We then input the pretrained dictionaries, frames containing $\mathbf{V}_G$ and those in the buffer $\mathbf{B}$ and the time activations $\mathbf{H}_B$ of the frames in $\mathbf{B}$ to the BKL-NMF algorithm described in Algorithm 3. This is used as a subroutine for every set of frames to obtain the time activations $\mathbf{H}_G$ and noise dictionary $\mathbf{W}_N$ in the semi-supervised and unsupervised cases. This is the same as Algorithm 2 but with the buffer of previous frames and their activations taken into account as well. After this, we reconstruct the clean signal of those frames $\mathbf{V}_{SG}$ using reconstruction $\mathbf{W}_{USM} \cdot \mathbf{H}_{SG}$. At every iteration of the set of frames we denoise, we update the buffer $\mathbf{B}$ by replacing the oldest g frames with the current frames that we denoised. This is so that for the next g frames we denoise, we are modeling parameters based on a

Algorithm 3 BKL-NMF

inputs $\mathbf{X}$, $\mathbf{W}_{USM}$, $\mathbf{H}_B$
if supervised **then**
  input $\mathbf{W}_N$ from training
else
  initialize random $\mathbf{W}_N$
end if
initialize random $\mathbf{H}_G = \begin{bmatrix} \mathbf{H}_{GS} \\ \mathbf{H}_{GN} \end{bmatrix}$
repeat
  $\mathbf{W} = [\mathbf{W}_{USM}, \mathbf{W}_N]$
  $\mathbf{R} = \frac{\mathbf{X}}{\mathbf{W} \cdot \mathbf{H}} = [\mathbf{R}_B, \mathbf{R}_G]$
  $\mathbf{H}_G = \mathbf{H}_G \odot (\mathbf{W}^\top \cdot \mathbf{R}_G)$
  if unsupervised **then**
    for $i = 1 : M$ **do**
      $\mathbf{H}_{GS_i} \leftarrow \mathbf{H}_{GS_i} \frac{1}{1 + \frac{\lambda}{\epsilon + ||\mathbf{H}_{GS_i}||_1}}$
    end for
  end if
  $\mathbf{H}_N = \mathbf{H}_N + w$
  if unsupervised or semi-supervised **then**
    $\mathbf{A} = (1 - \mu)(\mathbf{W} \odot (\mathbf{R}_G \cdot \mathbf{H}_G^\top)) + \mu(\mathbf{W} \odot (\mathbf{R}_B \cdot \mathbf{H}_B^\top))$
    $\mathbf{W}_N = \mathbf{A}_N$
    renormalize columns of $\mathbf{W}_N$ to sum to 1
  end if
until convergence

buffer with the last b frames. Similarly we update $\mathbf{H}_G$ as well to prevent the model from overfitting.

2.3 EXPERIMENTS AND RESULTS

The model was tested as a real-time speech denoising tool. The TIMIT database was used for training the speech dictionary. This database has 10 spoken sentences for each speaker and has a large collection of male and female speakers and accents based on regions in the United States.

For the fully supervised and semi-supervised cases, we used a random one of these sentences in the test signal and trained on the remaining 9 sentences. In the unsupervised case, we picked ten male and ten female speakers in a

Algorithm 4 The online semi-supervised denoising algorithm

input $\mathbf{W}_{USM}$
input $\mathbf{W}_N$ if fully-supervised
for $t = (1 + b) : T$ **do**
  $\mathbf{V}_G$ is the current set of frames being denoised
  normalize : $V_G(f) = \frac{V_G(f)}{\sum_f V_G(f)}$
  $\mathbf{X} = [\mathbf{B}, \mathbf{V}_G]$
  if supervised **then**
    $(\mathbf{H}_G)$ = BKL-NMF$(\mathbf{X}, \mathbf{W}_{USM}, \mathbf{H}_B, \mathbf{W}_N)$
  else
    $(\mathbf{W}_N, \mathbf{H}_G)$ = BKL-NMF$(\mathbf{X}, \mathbf{W}_{USM}, \mathbf{H}_B)$
  end if
  reconstruct clean speech : $\mathbf{V}_{SG} = \mathbf{W}_{USM} \cdot \mathbf{H}_{SG}$
  replace the oldest g frames in buffer $\mathbf{B}$ with the current g frames
  replace the g oldest time activations in $\mathbf{H}_B$ with $\mathbf{H}_G$
  $t = t + g$
end for

random manner to build the universal speech model. Each speaker had about thirty utterances which were slightly different from each other. For training and testing, we used the same number of basis vectors for speech but varied this parameter for getting the best performance.

The noise was trained prior in the fully supervised case using the whole length noise signal. The test signal is generated such that a random excerpt from the noise signal is chosen to be mixed with the speech signal. These two sources, namely speech and noise, can be mixed according to various ratios depending on how loud we want the noise to be compared to the speech. Since we are modeling systems that have background noise in speech signals, we generally have this ratio with higher energy of speech than noise.

We experimented with a lot of the parameters, but used a fixed length for the fast fourier transform (FFT), window size/type and overlap for the short-time fourier transform (STFT). We used a sampling rate of 16 kHz, a 1024-point FFT with a Hann window of the same length and an overlap of 75%.

Noise data from [7] were mixed with the clean speech signals. Types of noise included semi-stationary cases such as casino ambience and ocean sounds, but also non-stationary cases such as bird and motorcycle sounds. We ran a series of tests first to compare the performance of tuning various pa-

rameters in the algorithm. We then tested the algorithm on the three cases of having different training information. It is important to determine the degradation in performance as you have less training. Also the online case models parameters based only on a few frames, so we ran tests to compare how well this performs compared to the offline method.

The parameters that were varied to test the performance of the algorithm under different noise conditions, genders and noise levels are: λ, w, μ, number of speech and noise basis vectors. The results for these are shown in figures 2.5, 2.6 and 2.7. As can be seen, varying these parameters affects the values of the SIR, SAR and SDR ratios. The source separation quality measured by SIR and artifacts ratios measured by SAR are inversely related and this proves the existence of the trade-off between these two. After various tests, we tried to balance the trade-off between the SIR and SAR [23]. We decided to use $\lambda = 10$, $w = 1$, $\mu = 1/3$, $G = 40$ frames at a time, size of buffer $B = 60$, 40 speech basis vectors for each speaker for training and testing and 20 noise basis vectors in the offline case, and 5 noise basis vectors for every iteration in the real-time case. The SIR and SAR values of the clean output speech signals for all the cases were obtained using the BSS-eval toolbox [8].

The results of these experiments, averaged over all the speakers and noises are shown in figures 2.8 and 2.9. Figure 2.8 compares the performances of the algorithm in the fully-supervised, semi-supervised and unsupervised training cases. The fully-supervised case performs the best because of having a dictionary of the test speaker in advance and also having an idea of the noise type in the background. The semi-supervised case does not have training information of the noise in advance and hence uses a randomly declared positive matrix as the noise dictionary. The unsupervised case used some random speech from 10 male speakers and 10 female speakers as the universal speech model and a random noise dictionary. These two cases, even though having little or no prior information of the test signal, perform well and are not significantly inferior to the fully-supervised case. There is only a 1-2 dB dip from one case to another in the different ratios.

Figure 2.9 compares the performance of the online and offline algorithms in the case of unsupervised training. The online algorithm processes the signal frame by frame or with a bunch of frames put together. This leads to problems of overfitting the noise dictionary. The online algorithm is almost as good as the offline one. There is only a 2-3 dB dip in SIR and 1-2 dB

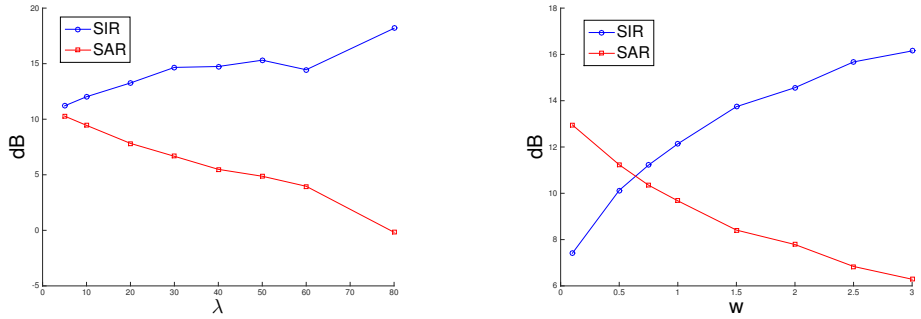


Figure 2.5: The figure on the left shows the performance of denoising of 5dB mixture for different λ values. Increasing λ makes use of dictionary elements from fewer speakers from the USM model and improves the output SIR at the expense of a reduced SAR. The figure on the right shows the performance of the algorithm for different w values. By boosting w, we allow the noise dictionary to explain more of the input mixture, thus removing more noise and improving the final SIR, but degrading the speech signal and reducing the SAR.

decrease in SDR/SAR.

2.4 CONCLUSION

NMF has been used in many previous cases for source separation. It has proved to be a good tool for speech denoising as well. In our case, we have developed a model where we do not need any prior information of the speaker or noise. It is a real-time speech denoising tool that does the denoising when given just a general idea of speech.

The disadvantage with the unsupervised case is that it picks up any speech from the background as well. This is because of the universal speech model with sparsity that tries to activate any speech in the signal. The supervised and semi-supervised cases do a better job in this scenario due to their speech dictionaries containing spectral information of the particular speaker. This approach of unsupervised real-time speech denoising does come at a cost of lower source separation performance and degraded speech quality. But as shown by figure 2.8, the degradation is not too much and the performance of the unsupervised is quite comparable to the fully supervised case. In the real-world when trying to use a speech denoising tool, we cannot expect to have training of the noise type because of the randomness of what is

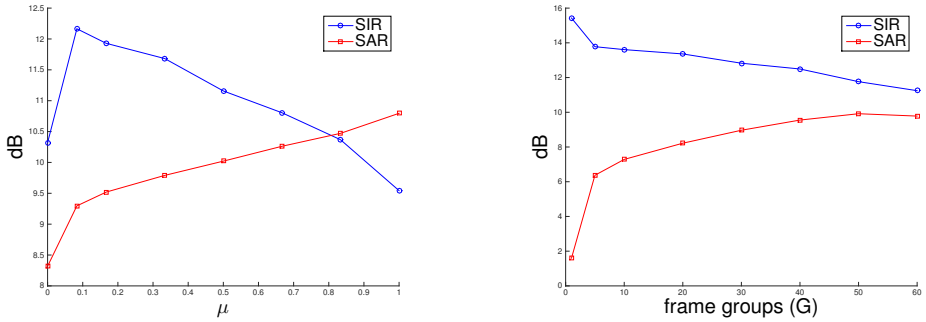


Figure 2.6: The figure on the left shows the performance of the algorithm for different μ values. A small μ allows us to adapt to the noise source faster, producing a higher SIR, but results in degrading the speech signal in the process and reducing the SAR. The figure on the right shows the performance of the algorithm for different g values. By fitting the data to more frames we get more averaging, which decreases the SIR but improves the SAR.

called noise. It could be from a wide range of sources like vehicles, musical instruments, channel distortion, commotion of people like footsteps, etc. A robust model would work for any speaker speaking to the system. In case of a voice recognition system, we want the denoising to be done in real-time and independent of the speaker. The model that we have developed does a good job of achieving this goal in terms of information required and performance.

The main contributions from this chapter are that we have developed an NMF model that uses a general speech dictionary USM and a random positive noise dictionary to perform the denoising of a signal. Through algorithm 4 we have shown that this is an online model and performs well with some artifacts. We have used both male and female speakers and different noise types in the test cases to generate the shown results. This proves that our unsupervised NMF algorithm is very robust and performs independently of the type of speaker, speech and noise in the background.

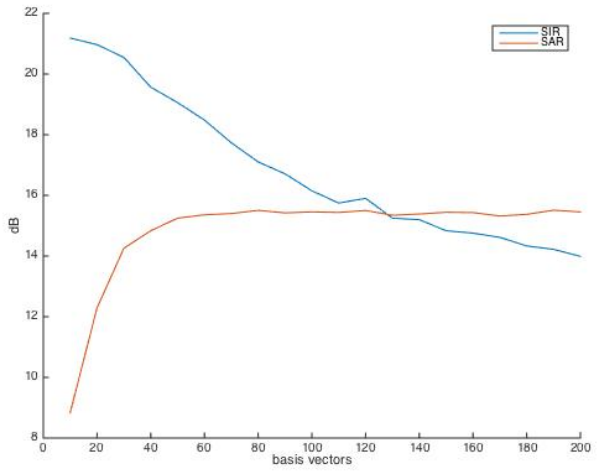


Figure 2.7: The change in SIR and SAR measures plotted against the change in basis vectors allocated for speech. The trend is similar in that there is an inverse relation between the two. Using lesser basis vectors for speech does good noise separation but cannot model the speech well due to under-fitting. Similarly, using a lot of basis vectors models the speech well by learning more information, but this comes at the cost of source separation.

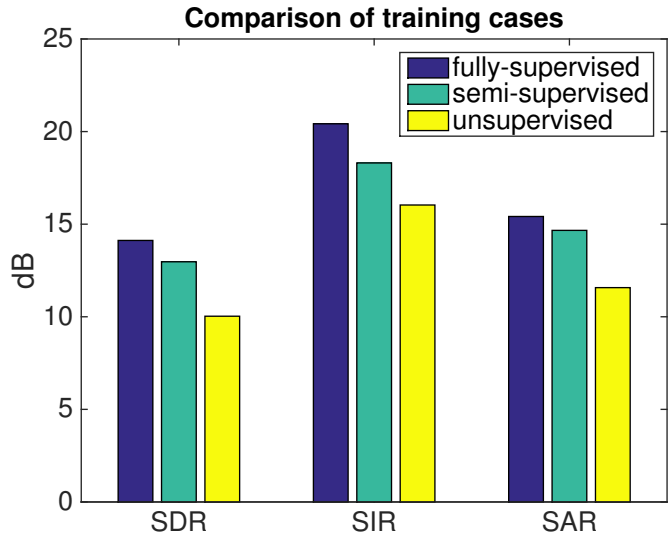


Figure 2.8: The figure compares the performance of the three different training cases: fully-supervised, semi-supervised, unsupervised. The denoising was done offline. As expected, the fully-supervised case performs the best, followed by semi-supervised and then unsupervised. But they are quite comparable as the degradation is not too great.

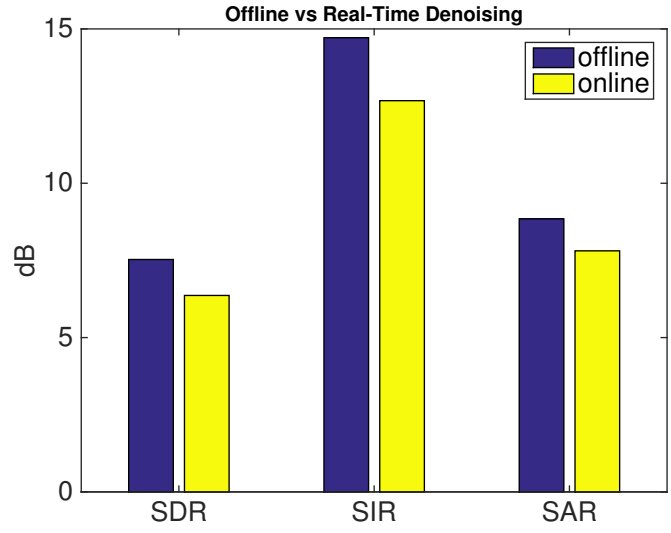


Figure 2.9: This figure compares the performance of the algorithm offline and in real-time with unsupervised training. The real-time case even though modeling the noise dictionary frame-by frame, has a comparable performance to the offline version.

CHAPTER 3

NEURAL NETWORKS

3.1 NEURAL NETWORK INTRODUCTION

This chapter shows the use of fully-connected neural networks (FNN) for speech denoising. Neural networks are being used recently in a wide range of applications mainly for classification tasks or source separation problems [25], [26]. We first introduce the theory behind neural network algorithms and why they are a viable method to try to solve various problems. We then explain the architecture used for our specific neural network and show results to prove that the chosen parameters and options were most optimal.

The experiments in this section go over the three cases of training: fully-supervised, semi-supervised and unsupervised. We try to model a system where we do not need any training information of the speaker in test or of the noise in the background. We show how this model, because of the additional complexity and approach, performs better than the NMF algorithm explained in the previous chapter. We also talk about how this algorithm can be modeled in real-time.

3.2 THEORY

This section goes over the theory behind the formation of a neural network. We start with classification for linear cases, how more complex non-linear data is handled, and then to neural networks and how they are used for regression.

3.2.1 LINEAR CLASSIFIER

The linear classifier creates a hyperplane across data points in order to classify them into labels. This is good for classifying multi-dimensional data as it becomes more difficult to train Gaussians due to the requirement of a lot of samples. The naive Bayes classifier assumes independence across dimensions which might not be ideal in all cases [27]. A linear classifier creates a boundary which is hyperdimensional and is relatively easy to train.

$$y_n = w^\top \cdot x_n,\ n = 0...N-1\ samples \tag{3.1}$$

where w is the weight matrix or the features learned to perform this classification and $\mathbf{x}$ is the data being classified. $\mathbf{x}$ is multi-dimensional with D dimensions and N samples:

$$\mathbf{x}_n = [\mathbf{x}_n^1, \mathbf{x}_n^2, \cdots, \mathbf{x}_n^D]^T,\ n = 0...N-1 \tag{3.2}$$

$$w = [w^1, w^2, \cdots, w^D]^T \tag{3.3}$$

w is a feature matrix that transforms the input $\mathbf{x}$ into a new space. In this case, w is only a vector since we just get a binary label. This is a case of binary classification where we are trying to classify between two classes ω_1 or ω_2.

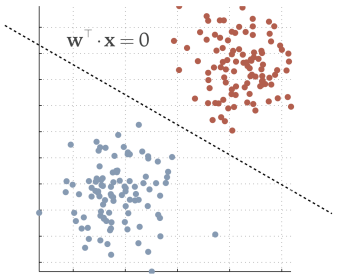
$$\begin{aligned} y = w^\top \cdot x_n > 0,\ choose\ \omega_1 \\ y = w^\top \cdot x_n < 0,\ choose\ \omega_2 \end{aligned} \tag{3.4}$$

Figure 3.1a is a good representation of a binary linear classifier.

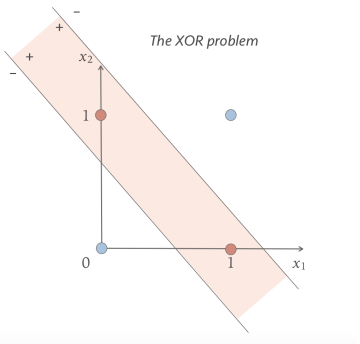
Multiclass classification is also possible where a w is learned between every two classes. Hence we have different permutations of feature matrix w between each pair of classes.

3.2.2 TRAINING USING GRADIENT DESCENT

In the training step, we need to fit a classifier such that there is zero error in the labels we assign each sample. We need to choose an error function such that we get a good estimate of the number of wrong labels. We use the Euclidian distance equation to find error:



(a) Binary classified data separated by linear classifier [28]



(b) The XOR case exemplifying non-linear data. This needs a different classifier, one with two decision lines [28]

Figure 3.1: The two figures above show examples of classification of linear and non-linear data.

$$e = \sum_{n=0}^{N-1} (y_n - t_n)^2 \tag{3.5}$$

We start with some random value of the weight matrix w and use gradient descent to update it at every iteration so that we reach some point of local minimum. This is done by:

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \eta \frac{\partial e}{\partial w} \tag{3.6}$$

where i is the current iteration of the gradient descent. Figure 3.2 shows how gradient descent finds an optimum by descending slopes of the error function. This minimum achieved could be a point of local or global minimum. The global minimum is the most optimal point and reduces the error function most. But it really depends on the starting point, which in our case is random. The initialization of $\mathbf{w}$ in our case determines whether we reach a local or global optimum.

3.2.3 NONLINEAR CLASSIFIER

The data in figure 3.1a is linear and hence we can model a linear classifier to separate the two classes. If we take the case where we have data like in the XOR case, the data is non-linear and hence the classifier above will not

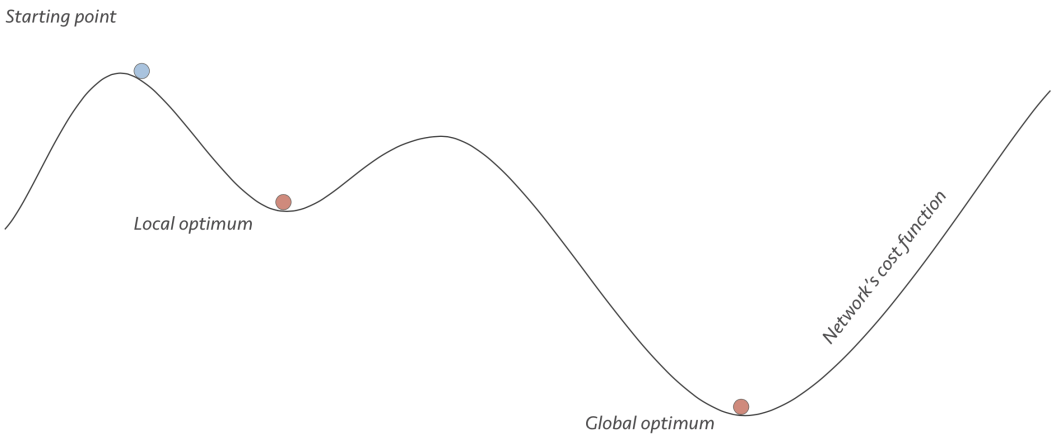


Figure 3.2: This shows the various points of local minimum and the global minimum. It portrays how the gradient descent in equation 3.6 goes about trying to minimize the error and reach an optimum [28].

work. We need a classifier like the one shown in figure 3.1b where we have two decision lines [28].

This can be achieved by projecting the data into a different space so that it becomes easily separable. So we use two layers to do this. As shown in figure 3.3, the first layer projects the data into a space that makes it much easier to classify the data. The problem then becomes one of linear classification and the second layer can do that easily. So this forms a two-layer neural net as shown in figure 3.4 [28]. Similarly if this problem is still non-separable after the first layer's projection, we add one more layer such that the three-layer network has two layers for making the case linear and the third can be used for the final classification. The nodes shown in figure 3.4 are called the hidden nodes of each layer. They represent the dimensionality of the feature space into which the particular layer is projecting the input data. In this case, the weight matrix $\mathbf{w}^1$ of the first layer is of size $D1$ x $D2$ where $D1$ is the number of input dimensions and $D2$ is the number of nodes in the first hidden layer. This way a multilayered network would have as many weight matrices as the number of layers. Each will have a size according to the dimensionality of the next layer and current layer's feature space.

We can train the different $\mathbf{w}$ matrices separately using equation 3.6 for each of them. We use the chain rule for differentiation. This will be illustrated in more detail in later sections.

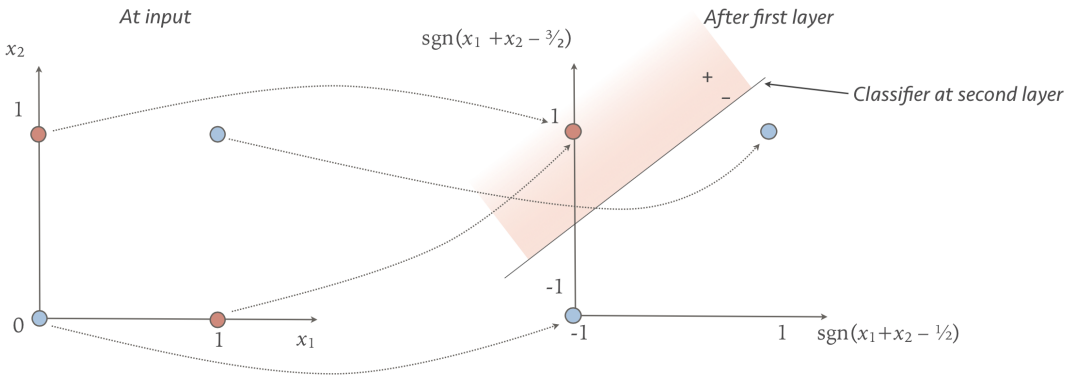


Figure 3.3: Taking data from figure 3.1b, we project it into a space such that it can be linearly classified [28].

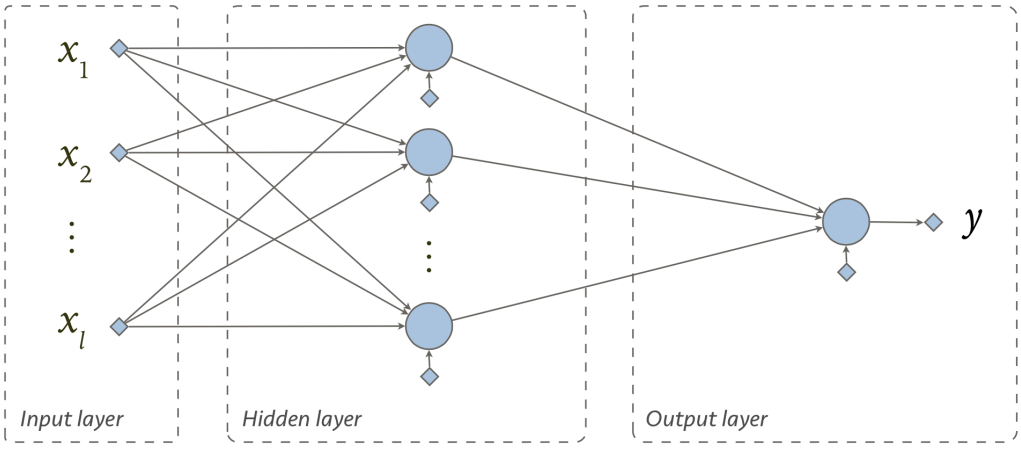


Figure 3.4: A two-layer feedforward neural net which can be used to solve the XOR case like in figure 3.3 obtained from [28].

3.2.4 DISCRIMINANT FUNCTION

In classification and regression tasks, each layer will require some function to scale or suppress the output based on what we want to feed to the next layer. We hence use a discriminant or activation function:

$$\begin{aligned} a_n &= w^\top \cdot x_n \\ y_n &= g(a_n) \end{aligned} \tag{3.7}$$

where $g(\cdot)$ is an activation function. Assuming y_n is the label we want for the nth sample, this activation function is going to help scale the label to two values, say 0 and 1 or 1 and -1 like a unit step function. The issue is that this function is not differentiable and hence we cannot do the differentiation in the gradient descent step. We need to use functions that are differentiable and can approximate the unit step function. One commonly used function is [29]:

HYPERBOLIC TANGENT

$$g(a_n) = \frac{e^{a_n} - e^{-a_n}}{e^{a_n} + e^{-a_n}} \tag{3.8}$$

The hyperbolic tangent is a popularly used activation function and can be used due to the differentiability required for gradient descent. This is shown in figure 3.5.

3.3 NETWORK ARCHITECTURE

This section explains the neural network architecture we used for speech denoising. We also illustrate the algorithm and various steps needed to train this network. We show the test phase which is just a single pass through the network in order to separate the speech from the noise.

3.3.1 REGRESSION

We explained linear and non-linear classifiers for binary classification in the previous section and how they are useful for classifying data with high dimensionality. We move from classification to regression, which is a method

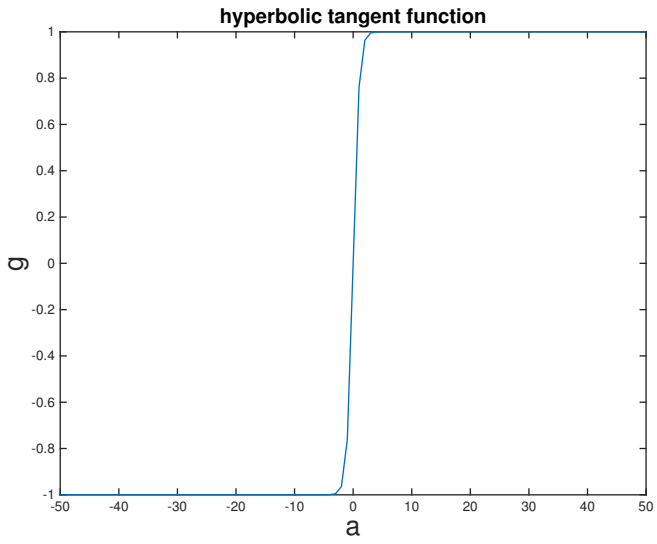


Figure 3.5: The hyperbolic tangent approximates a unit step function and is differentiable.

that outputs continuous values rather than discrete labels. This is because we want to output a signal that is clean speech which has the same dimensionality and samples as the input noisy mixture. So we want to train a network that can create this regression between clean speech and noise and output a spectrogram that contains just one of the sources, which in our case is the speech. The training labels in our case would be the equivalent clean speech. So we can train a model where we input a noisy speech signal and have the equivalent clean speech of the input as the training labels. We can use equation 3.5 in the same way to compare the output from the final layer to this clean speech label at every iteration. We can then use equation 3.6 to update the weights of the various layers in order to perform the regression.

3.3.2 ALGORITHM

We use a neural network somewhat similar to a denoising autoencoder. This is needed because we want the output to have the same dimensionality as the input.

FEEDFORWARD MODEL

This network is a feedforward model as illustrated in figure 3.4 and generally has one or two hidden layers. The training step uses the equations 3.5 and 3.6. Since we have multiple layers here, we need to use the chain rule of differentiation when updating each weight matrix. These are the feedforward steps required for a fully connected network with one hidden layer [30]:

$$\begin{aligned} &a_n^1 = w^1 \cdot x_n^0, \ \textit{where } x_n^0 \textit{ is the input} \\ &x_n^1 = g(a_n^1) \\ &a_n^2 = w^2 \cdot x_n^1, \ \textit{this is the hidden layer} \\ &x_n^2 = g(a_n^2), \ x_n^2 \textit{ is the required output (clean speech)} \end{aligned} \tag{3.9}$$

Here x_n^l is the nth sample's output from the lth layer and input to the $(l+1)th$ layer. Once the network weights are trained, the test step is just a single pass through this model.

MODIFIED ACTIVATION FUNCTION

At each layer, we want the values of the output to be in the range of $[0, \infty)$ since we are dealing with spectrograms here. Hence using the hyperbolic tangent in equation 3.8 might not be ideal. We can use the linear function:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.10}$$

But all the negative values are driven to one single value, zero. We use a modified rectified linear activation function instead [31]:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \frac{-\epsilon}{x-1}, & \text{if } x < 0 \end{cases} \tag{3.11}$$

ERROR BACK PROPAGATION

In order to get the weight matrices $\mathbf{w}^l$ to compute the necessary regression, we need to do some error-back propagation starting from the final layer:

$$\delta_n^l = \frac{\partial g(a_n^l)}{\partial a^l} \cdot \frac{\partial e_n}{\partial a^l} \tag{3.12}$$

Depending on what layer's parameters we are trying to model or optimize for the task, the general equation would be following the chain rule:

$$\delta_n^l = \frac{\partial g(a_n^l)}{\partial a^l} \cdot (\mathbf{w}^{\top l+1} \cdot \delta_n^{l+1}) \tag{3.13}$$

GRADIENT

We calculate the gradient of the error function with respect to each layer's weight:

$$\frac{\partial e}{\partial \mathbf{w}^l} = \delta_n^l \cdot x_n^{l-1} \tag{3.14}$$

This is the update step at each iteration:

$$\mathbf{w}^l = \mathbf{w}^l - \eta \frac{\partial e}{\partial \mathbf{w}^l} \tag{3.15}$$

INITIALIZATION

The initialization step is important as this determines where we start on the error function curve. We generally initialize the weights to somewhat small values [30]. If the values are too small, the signal as it goes from layer to layer becomes tiny and insignificant. If the initial values are too big, the output from each layer explodes. Hence we have used the *xavier* initialization in Caffe and *glorot uniform* in Keras for this.

SCALING

We scale the input clean spectrogram $\mathbf{x}$ and label clean speech $\mathbf{t}$ down by some factor to help convergence. This is to avoid the values of the parameters and error from becoming too big.

$$\mathbf{x} = \mathbf{x}/factor \tag{3.16}$$

Algorithm 5 FNN TRAINING

inputs noisy spectrogram $\mathbf{x}$, label clean speech $\mathbf{t}$
initialize $\mathbf{w}^l$ for all layers

repeat
  for $l = 1{:}L-1$ **do**
    calculate δ^l from equation 3.13 for all n samples
    calculate gradient for the layer weight $\mathbf{w}^l$ from equation 4.2
    Update $\mathbf{w}^l$ using equation 4.3
  end for
  for $l = L$ **do**
    calculate δ^L from equation 3.12 for all n samples
    calculate gradient for the layer weight $\mathbf{w}^L$ from equation 4.2
    Update $\mathbf{w}^L$ using equation 4.3
  end for
until convergence

3.4 RESULTS

This section shows the performance of the proposed neural network structure under various parameters. We ran tests to show how this algorithm can perform better than the NMF algorithm proposed in the previous chapter due to the non-linearity and complexity of the model. We compare the performances of the fully-supervised and unsupervised cases for this network. All the training datasets and the way the test signals were generated are very similar to the ones for NMF in the previous chapter.

We ran experiments to compare the performance of different architectures of the neural network. The regression model can have multiple hidden layers and also the number of hidden nodes in them can be experimented with. This is shown in figure 3.6 with the numbers in the legend representing the number of hidden nodes in each layer. The number of hidden layers is represented by how many hidden node numbers are separated by a slash. Since our training database is not too big, a few layers would be enough to perform the regression we need. In the figure, we have compared different cases of using one and two hidden layers. These are all fully-supervised cases where we have prior training information of the same speaker and noise type as in test. As can be seen, using one hidden layer gives a good enough performance. The two hidden layer cases increase the source separation ratio (SIR) by a

little but cause a decrease in the speech quality (SAR/SDR) which is lower than that of the one hidden layer case. Also training the two hidden layer case requires much more computational power than the others and hence it makes more sense not to have more than one hidden layer. In the one hidden layer case, using very few nodes decreases the ratios significantly due to fewer feature vectors being modeled. So the cases with 10 and 100 hidden nodes have very poor ratios.

The plot in figure 3.7 compares the performance of using various FFT sizes. These are all fully-supervised cases with one hidden layer. The 1024 point FFT case shows a significant edge over the others in the SIR ratio and has SAR and SDR measures comparable to the others. We have also compared the performances of using various activation functions in the one hidden layer case. These are shown in figure 3.8. The train and test phases are separated by a slash. $tanh/relu$ implies using $tanh$ activation function for training and $relu$ in test. The $relu/relu$ case gives the best performance in terms of both source separation and output speech quality.

This fully-connected neural network (FNN) algorithm is shown to perform much better than the KL-NMF algorithm in the previous chapter by figure 3.9. The SIR, SAR and SDR measures are higher by more than 5 dB each, which is a significant increase in performance. This is because neural nets assume a non-linear model and can handle data better with their multiple layers. Also the more complex structure can model the parameters for regression better. This is the reason that neural networks have become very popular in various fields of research and have been shown to perform better than other existing methods.

The FNN can be assumed to be an online model since we impose the weight matrix on each sample individually. At each layer l in the test phase, we have a weight matrix $\mathbf{w}^l$ that has dimensionality according to the previous and current layers' feature space. This matrix is used to transform each sample of the input and the inputs from the previous layers. Similarly for the semi-supervised and unsupervised cases, there is no extra parameter computation like in the NMF approach. Hence the FNN algorithm is an online approach.

We had experiments to compare the performance of the unsupervised case with the fully-supervised case. The plot for this is shown in figure 3.10. The unsupervised case has a 4 dB decrease in each of the measures, which is the same as in the KL-NMF algorithm. The unsupervised case for FNN does the

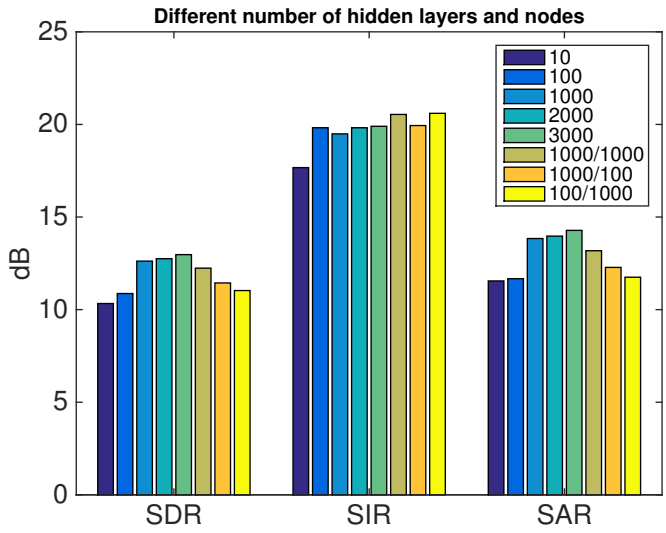


Figure 3.6: This figure compares the different architectures used for the neural network. The numbers are the hidden nodes used in each hidden layer. The slash implies multiple hidden layers. The ones with the single number have just one hidden layer.

denoising well taking into consideration that the network is not trained on the speaker or the noise type. It is quite comparable to the fully-supervised case.

We could achieve convergence in training in Keras after very few iterations as shown in figure 3.11. This is because the model used in Keras for training adjusts the learning rate according to the error measure. If the error increases after a certain update, it implies that we are overshooting on the curve and hence need to take smaller step sizes. Hence the learning rate or step size is decreased in this case. Similar is the case when the error does not change. This happens when we take a step that might have missed the minima and the function keeps bouncing around the trough. If the error is decreasing, we keep the constant learning rate but there is room for experimentation as to whether a bigger step size will decrease the error faster. The general trend in the BSS eval [8] measures should look like in figure 3.11. The convergence might not be as quick for the error function itself, but this figure is a plot of the SDR/SIR/SAR measures of a random test signal denoised every 50 iterations. This proves that the training process always does some overfitting with respect to the training signal. After a certain point in training, the decrease in error does not correspond to better denoising.

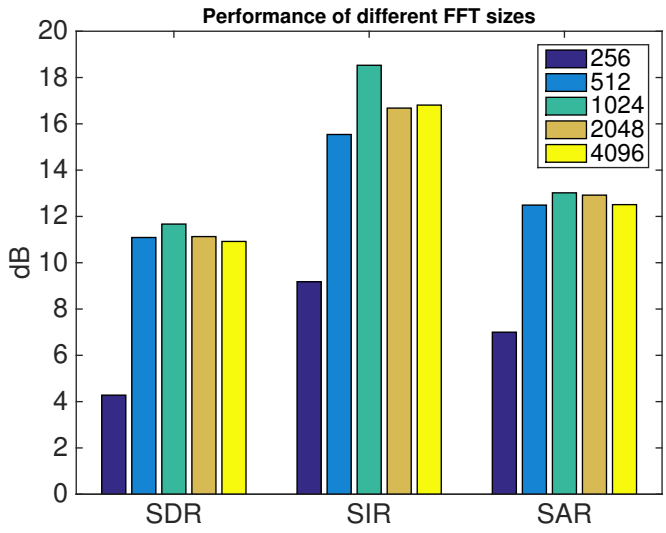


Figure 3.7: This plot compares the performance of using various FFT sizes while taking the STFT of the input audio time series signals.

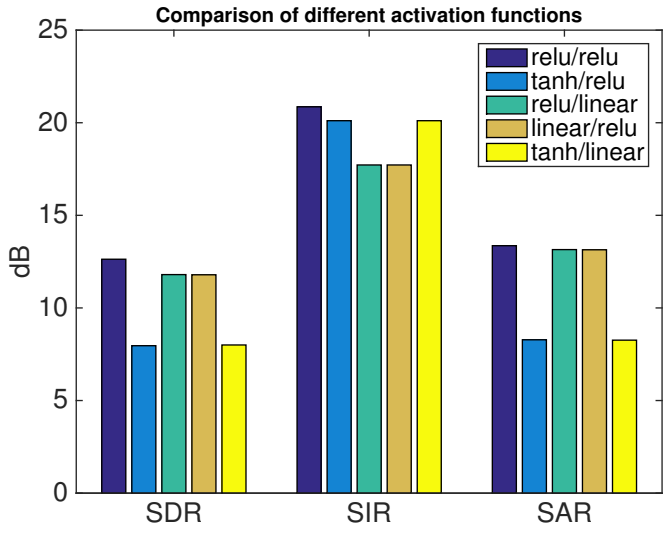


Figure 3.8: This figure compares the performance of using various activation functions at the training and test step. The activation function before the / is the one used for training and the one after the / is for test. *tanh* is the hyperbolic tangent function from equation 3.8. *relu* is the rectified linear activation function from 3.11 and *linear* is the function from equation 3.10.

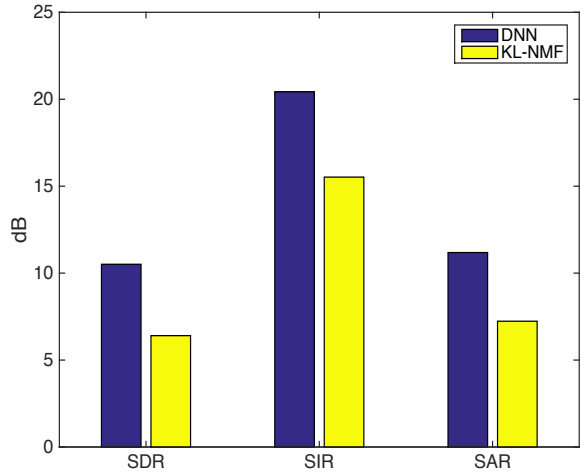


Figure 3.9: This plot compares the performance of fully supervised neural networks represented as Deep Neural Network (DNN) and the KL-NMF algorithm from the previous chapter. The proposed neural network algorithm is significantly better than in the order of >5dB in all the SAR, SIR and SDR measures.

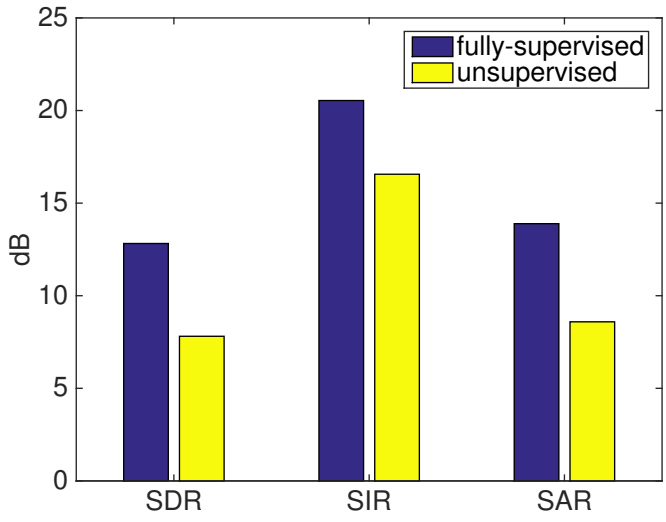


Figure 3.10: This plot compares the performance of this algorithm in the unsupervised case to the fully-supervised one. Both the algorithms were tested on the same test dataset. There is only a 4-5 dB decrease in each of the measures, which is reasonable.

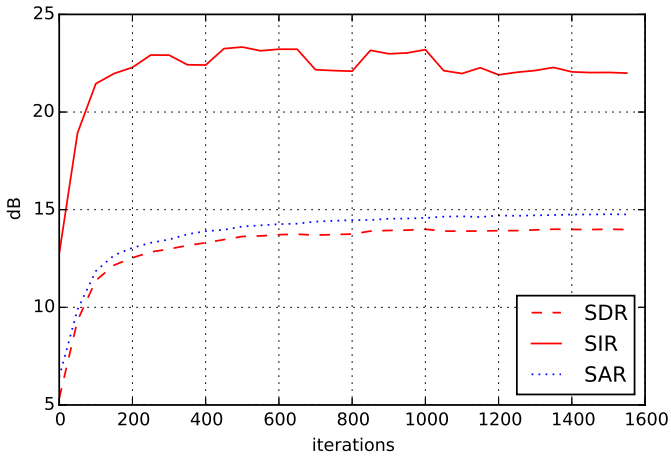


Figure 3.11: This is a plot of the SDR/SIR/SAR measures of a test signal while training a network. The trained model was used to denoise this test signal after every 50 iterations and the BSS eval measures were plotted every time. As can be seen, the measures converge and become constant after very few iterations. This is a validation test performed in Keras.

3.5 CONCLUSION

Neural networks have become very popular due to their model of having multiple layers and understanding features like the human brain. They have been extremely successful in classification tasks as well due to this reason. They can transform the data into different feature spaces to make it easier for the final layer to complete the required task. Using fully-connected neural networks for the purpose of speech denoising has helped us get very good results.

This model can do online speech denoising like NMF but gives much better performance. This is because NMF is more like PCA approaches where we model a single feature matrix. The FNN algorithm with semi-supervised and unsupervised training cases gives comparable measures to the fully-supervised case. As shown in figure 3.10 the degradation with lesser training information is not too drastic. Also, the learning rate or η could be adjusted according to whether the error was decreasing or overshooting. This really helped train better models for regression between speech and noise as we can minimize the error at every step by following the right direction and step-size in the gradient descent.

The FNN algorithm is very computationally heavy in the training process

but its test phase involves just a single pass of the input noisy mixture through the network. Since all the training is done offline, real-time denoising is much more efficient with this algorithm than NMF. In the BKL-NMF algorithm, we have to denoise individual frames, which involves about 20 iterations. So using an FNN with one or two hidden layers for real-time denoising requires much less computation when doing the actual denoising. This is very useful in speech recognition systems as we can reduce the CPU usage of this algorithm and allocate most of the resources to the actual speech recognition.

To summarize, the main contributions from this chapter are that we have proposed a model similar to the one in chapter 2 but using fully-connected neural networks. This algorithm performs significantly better than the NMF as the speech is better separated from the noise and the output speech is much more clear as it has less distortion and artifacts. Also when used in a real-time speech recognition system, this FNN method requires much less computation than NMF due the single-pass test phase. Given that we can afford the time required for training this model, it is much better than using the previous NMF model on a speech recognition system due to the better and faster performance.

CHAPTER 4

CONVOLUTIONAL NEURAL NETWORKS

4.1 INTRODUCTION

The last section explained the architecture and working of a fully-connected neural network for speech denoising. It clearly beat the KL-NMF algorithm explained in chapter 2 in terms of the source separation performance and the output speech quality. Also it was shown to have less of a computational demand in the real-time denoising phase. We now introduce the use of convolutional neural networks (CNN) for the same purpose of speech denoising.

CNNs have been used widely in the field of vision and image processing. A lot of image classification [2] and feature generation tasks [32] have been solved by using a CNN due to its capability of generating two-dimensional features. The need for learning image features and templates in vision has made it helpful to use CNNs for these various tasks. This might be useful in cases of non-stationary or semi-stationary noise types in the background. They can be modeled better if we can get a two-dimensional template like that for an object image. Some work has been done previously on this topic of using CNNs for audio source separation [33], but the convolution part was based on how the window was used during STFT. The actual network used was still an artificial neural network or, as we called it in this research, a fully-connected neural network (FNN).

4.2 THEORY

A convolutional neural network is so called because it involves one or multiple convolutional layers. A CNN generally has these convolutional layers followed by some fully connected layers as well [34]. As mentioned in the previous

chapter, the l-th layer of a network's general form of processing for an input $\mathbf{x}^{(l)}$ is :

$$\mathbf{x}^{(l+1)} = f^{(l)}\left[g(\mathbf{w}^{(l)}, \mathbf{x}^{(l)})\right] \tag{4.1}$$

where for a fully connected network $g(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ and for a convolutive network $g(\mathbf{w}, \mathbf{x}) = \text{maxpool}(\mathbf{w} * \mathbf{x})$, where the $*$ operator denotes convolution and the maxpool$(\cdot)$ function is described below. The parameter $\mathbf{w}^{(l)}$ is referred to as the layer weights and in the case of a fully connected layer it is a matrix, and in the case of the convolutive network it contains a set of filters which are also often referred to as the kernels. Finally the function $f^{(l)}(\cdot)$ is the activation function which can take various forms.

In fully-connected layers, we have the weight matrix w^l that has as many feature vectors as the space we are projecting the data into. The convolutional layers have a different architecture from this. Instead of multiplying, we convolve their input with the kernels represented by $\mathbf{w}^l$ of a given size and produce fm number of feature maps. The number of feature maps determines the number of inputs we pass on to the next layer. Similar to fully connected layers, we learn multi-dimensional weights that connect feature maps from the current convolutional layer to the next convolutional or fully connected layer. This corresponds to choosing the number of hidden nodes in a fully connected layer. We also have the choice to set the size of the kernels according to our needs. We will discuss the implications of that later in the experiments section.

At the end of every convolutional layer, we perform sub-sampling in order to reduce dimensionality of the output passed on to the next layer. This is a process of taking a non-overlapping rectangular region over the output from convolution and either taking the mean of the values or just passing on the max value of that region. This helps reduce computation as it leads to dimensionality reduction at every layer [35]. This sub-sampling leads to position invariance in the taken rectangular regions and also downsamples it.

4.2.1 TRAINING

During the training phase we estimate the appropriate weights and biases to enable the desired prediction. In case of convolutional network, these weights are higher dimensional as they have to connect every feature map

in the previous layer to every feature map in the next layer [35]. Obviously, using convolutional layers increases the complexity of the network and the amount of computation required for training.

The training process is very similar to the one for FNN algorithm in the previous chapter. The training process of learning these parameters can be done using various methods of gradient descent. This is done over a large number of iterations with a variable learning rate. The basic equations for gradient descent and parameter update are:

$$\frac{\partial e}{\partial \mathbf{w}^l} = \delta_n^l \cdot x_n^{l-1} \tag{4.2}$$

$$\mathbf{w}^l = \mathbf{w}^l - \eta \frac{\partial e}{\partial \mathbf{w}^l} \tag{4.3}$$

The overall algorithm follows the same steps as in chapter 3. The only difference is that we have convolution of the kernels with the input, and the weight matrix $\mathbf{w}^l$ for the convolution layers is more than two-dimensional.

After every convolutional layer, we need a function to downsample the output to reduce redundancy and also computation. Hence we can use either max pooling or take the mean of the window of a variable size. This size can also be tested for getting the best performance. Other parameters that can be varied for source separation quality and good speech perceptibility are the kernel size, number of feature maps in the convolutional layer and the number of hidden nodes in each of the fully connected layers. Also the number of layers L and how they are divided between being convolutional and fully connected can also be experimented with. Finally using error back-propagation, some of the commonly used loss functions are softmax loss and Euclidean loss. The latter gave better performance than some of the other loss functions for our case of speech based source separation.

4.3 ARCHITECTURE

For our purpose, we use an absolute valued spectrogram as a single image. This spectrogram generally contains multiple utterances of the speaker. Theoretically we have one big spectrogram as the input, but in Keras we divide this spectrogram so that we have multiple inputs to the network. Since the

output in the test phase should have the same dimensions as the input, we do not have any fully-connected layers in between. The regression case needs a two-dimensional image-like output of the clean speech spectrogram. Such a network was used in [36] for human hand tracking in images. Our network architecture is similar to this and outputs a clean speech spectrogram which is two-dimensional.

As shown in figure 4.1, we slide kernels or windows of size a x b across the given input and model fm feature maps in the hidden layer. The final layer then uses a similar or different window to project these 20 feature maps back into the single spectrogram space. These kernels learned during training form the **w** matrices for each layer. The convolution and window stride has to be adjusted such that we do not change the size of the output spectrogram. It has to have the same dimensions as the input spectrogram. In the training process, we use the Eucledian distance measure to compute the error at each iteration.

The input to this network is a noisy speech mixture containing speech and noise types depending on the three training cases. The label comparison is the corresponding clean speech which we use to measure the error at every iteration of training.

4.4 RESULTS

The training data used for this is similar to that for KL-NMF and FNN based denoising. In the training process, we achieved convergence after some iterations as shown in figure 4.1. But the performance of denoising was not as good as in the previous two chapters. We experimented with various window sizes, number of feature maps and layers, but we got the best SIR of about 10 dB and SDR/SAR of about 8 dB as shown in figure 4.2. Similar to the validation tests for FNN in chapter 3, the performance of the CNN algorithm on the test signal is the same after a few iterations. The largest kernel or window size that we used was 64 x 20. We could not experiment further with bigger windows due to limitations of Keras and the GPU used. There is a good chance that the performance will improve for bigger window sizes, especially those with a bigger dimension in the time domain. These might be able to model the non-stationary and semi-stationary noise types better

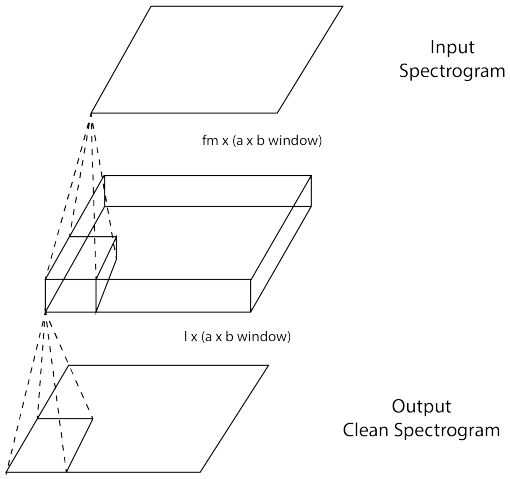


Figure 4.1: This is a representation of the architecture used for the CNN. The input is two-dimensional given as one input and we convolve using some windows of size a x b and form fm number of two-dimensional feature maps. We then project the data back into the original two dimensions to give a spectrogram back.

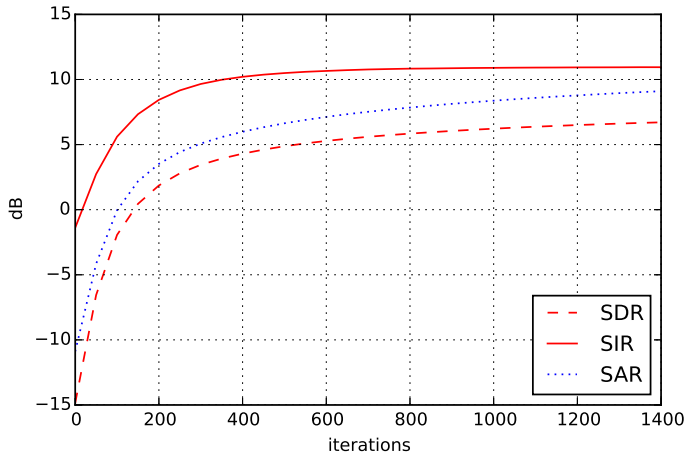


Figure 4.2: This is a validation test performed on a test signal in the fully-supervised case. The algorithm does not improve the denoising performance after about 1000 iterations.

due to their ability to capture the pattern better.

4.5 CONCLUSION

This method was promising in theory but we could not complete the experiments on time. It has potential due to its ability to model two-dimensional feature maps that can model the noise and even the speech better than one-dimensional feature vectors like in KL-NMF and FNN.

CHAPTER 5

CONCLUSION

This thesis has addressed various methods we used for solving the problem of speech denoising. Chapter 1 discusses the signal processing aspect as we do all the denoising in the frequency domain. We introduce the theory behind taking the STFT of the signal and explain the various parameters used like sampling rate and window size. We also explain the three cases of having different training information: fully-supervised, semi-supervised and unsupervised cases. We have generated results for the NMF and neural network methods for all these cases. We also explain the various measures and tools used for comparing the quality of source separation and its trade-off factor of degrading speech quality. Chapters 2-4 talk about the various algorithms used for denoising in detail.

Chapter 2 investigates the BKL-NMF algorithm in detail. We have first explained the theory behind the working of NMF and how it can be applied for audio spectrograms. We then talk about how we modified this so that we can have models that do the denoising when they have training information of only the speaker or just speech in general. We proposed a model that does the speech denoising in real-time and independently of the speaker and noise-type. We compared the cases of having different training information and showed that the unsupervised case has comparable performance to the other two cases. Also, the performance of this algorithm in real-time degraded very little. This shows that we have developed this robust tool for denoising that can be used in real-world speech recognition systems.

Chapter 3 discusses the use of neural networks for performing the denoising process. We use a fully-connected neural network to do this regression task and explain the theory behind it. We have shown results to prove why the parameters used were optimal and also to explain the trade-off between the source separation and speech quality. Similar to the NMF case, we have shown how this algorithm with unsupervised training has denoising perfor-

mance comparable to that of the fully-supervised case. Also this method is basically an online model since we deal with each sample of the input individually. We have shown that the FNN beats the NMF algorithm in denoising by a considerable amount because of its ability to model multiple layers.

In chapter 4 we have proposed an algorithm that uses convolution in the various layers of the neural network. We experimented with this approach as we thought that the non-stationary and semi-stationary noise types can be modeled better by learning some two-dimensional features. This algorithm does perform some denoising, but not as well as the ones in chapter 2 and 3.

REFERENCES

[1] "CMU Sphinx: Open source speech recognition toolkit." [Online]. Available: http://cmusphinx.sourceforge.net/

[2] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Neural Information Processing Systems (NIPS)*, 2012.

[3] M. Bay, A. Ehmann, J. Beauchamp, P. Smaragdis, and J. Downie, "Second fiddle is important too: Pitch tracking individual voices," in *Polyphonic Music Proceedings of the 13th International Society for Music Information Retrieval Conference*, Porto, Portugal, Oct. 2012.

[4] I. Sutskever, O. Vinyals, and V. Le, "Sequence to sequence learning with neural networks," in *Neural Information Processing Systems (NIPS)*, 2014.

[5] J. Garofolo, "Timit acoustic-phonetic continuous speech corpus ldc93s1. web download," in *Philadelphia: Linguistic Data Consortium*, 1993.

[6] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*, 1st ed. Prentice Hall Signal Processing Series, 1989.

[7] Z. Duan, G. Mysore, and P. Smaragdis, "Online PLCA for real-time semi-supervised source separation," in *International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA), LNCS 7191*, 2012, pp. 34–41.

[8] E. Vincent, C. Fevotte, and R. Gribonval, "Performance measurement in blind audio source separation," *IEEE Trans. on Audio Speech Lang. Process.*, vol. 14(4), pp. 1462–1469, 2006.

[9] C. Fevotte, R. Gribonval, and E. Vincent, "BSS EVAL toolbox user guide," 2005.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[11] "Keras: Theano-based deep learning library." [Online]. Available: http://keras.io/

[12] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[13] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010, oral presentation.

[14] M. Kim, J. Yoo, K. Kang, and S. Choi, "Nonnegative matrix partial co-factorization for spectral and temporal drum source separation," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, No.6, pp. 1192–1204, 2011.

[15] N. Mohammadiha, P. Smaragdis, and A. Leijon, "Low-artifact source separation using probabilistic latent component analysis," in *Workshop for Applications of Signal Processing in Audio and Acoustics, New Paltz, NY*. IEEE, Oct 2013.

[16] D. Lee and H. Sueng, "Algorithms for non-negative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.

[17] P. Smaragdis and J. Brown, "Non-negative matrix factorization for polyphonic music transcription," in *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 2003.

[18] A. Lefévre, F. Bach, and C. Févotte, "Itakura-saito nonnegative matrix factorization with group sparsity," in *International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 2011.

[19] P. Smaragdis, B. Raj, and M. Shashanka, "Supervised and semi-supervised separation of sounds from single-channel mixtures," *Proc. ICA 2007, LNCS, 4666*, pp. 414–421, 2007.

[20] M. Kim and P. Smaragdis, "Collaborative audio enhancement using probabilistic latent component sharing," in *International Conference on Acoustics, Speech, and Signal Processing*. IEEE, May 2013.

[21] T. Virtanen, J. Gemmeke, B. Raj, and P. Smaragdis, "Compositional models for audio processing, accepted for publication," *IEEE Signal Processing Magazine*, March 2015.

[22] D. Sun and G. Mysore, "Universal speech models for speaker independent single channel source separation," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vancouver, Canada.

[23] Z. Duan, G. Mysore, and P. Smaragdis, "Speech enhancement by online non-negative spectrogram decomposition in non-stationary noise environments," in *Interspeech*, Portland, OR, USA, September 2012.

[24] C. Ding, T. Li, and W. Peng, "On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing," *Computational Statistics and Data Analysis*, vol. 52, pp. 3913–3927, 2008.

[25] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Singing-voice separation from monaural recordings using deep recurrent neural networks," in *International Symposium of Music Information Retrieval*, Taipei, Taiwan.

[26] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Deep learning for monaural speech separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Florence, Italy, 2014.

[27] P. Smaragdis, "CS598PS lecture 9 - Decision theory and simple classifiers," class notes for CS 598, University of Illinois, fall 2014.

[28] P. Smaragdis, "CS598PS lecture 10 - Nonlinear classifiers," class notes for CS 598, University of Illinois, fall 2014.

[29] M. Hasegawa-Johnson, "ECE544NA lecture - From linear classifiers to neural network," class notes for ECE 544, University of Illinois, fall 2014.

[30] M. Hasegawa-Johnson, "ECE544NA lecture - Training a neural network," class notes for ECE 544, University of Illinois, fall 2014.

[31] D. Liu, P. Smaragdis, and M. Kim, "Experiments on deep learning for speech denoising," in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Singapore, 2014.

[32] S. Lawrence, C. Giles, A. Tsoi, and A. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no.1, Jan. 1997.

[33] A. Simpson, G. Roma, and M. Plumbley, "Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network," in *Proceedings of the International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, Liberec, Czech Republic, 2015, pp. 429–436.

[34] "Deep learning tutorial on convolutional neural networks, Stanford University." [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[35] "Deep learning: Convolutional neural networks." [Online]. Available: http://deeplearning.net/tutorial/lenet.html

[36] S. Nowlan and J. Platt, "A convolutional neural network hand tracker," in *Neural Information Processing Systems Foundation*, January 1995.