Chair of Embedded Intelligence for Health Care and Wellbeing
Univ.-Prof. Dr.-Ing. habil. Björn Schuller

Bachelor Thesis

Speech Separation using Deep Clustering

Maximilian Ammann
Matrikel-Nr.: 1471541

supervised by Shuo Liu

October 14, 2019

Abstract

Speech separation is the task of separating the voices of concurrently speaking persons. Compared to traditional signal processing approaches Deep Clustering presents a data-driven method which outperforms the classical ones.
In Deep Clustering, a recurrent neural network is trained to embed each time-frequency bin of a monaural speech spectrum into a high-dimensional space. The affinity of these embedding vectors in the target space is exploited to segment the spectrogram using k-means clustering. Vectors with low distance between each other belong to the same speaker. Therefore, the segmented clusters allow to create a spectrum mask for each speaker. Additionally, the mask is applied to the mixture in order to separate the voices. Finally, the time-domain speech signal is reconstructed. The approach offers speaker-independent separation and achieves an Source to Distortion Ratio of 8.89 on the high fidelity TIMIT data set.
In the thesis, experiments were conducted to find out whether the evaluation results of the original Deep Clustering paper hold true for different data sets which were not recorded under laboratory conditions. Therefore, the algorithm has been evaluated on various data sets, namely WSJ0, TIMIT and the freely available TEDLIUM, in order to determine its adaptability to varying grades of quality. Furthermore, the thesis states the stages of Deep Clustering precisely and presents a free and open source implementation.
The findings include that the quality of separation suffers from noise in the training and evaluation data set. In contrast, unknown speakers do not affect the success to the same extent. These results illustrate how important the data set is, and hence more research is needed on preparing real-world data.
For this reason two methods for visualizing the embedding vectors were introduced. The first illustration uses the linear dependence between the vectors whereas the second selects different orders for clustering to draw a conclusion about the success of Deep Clustering.

Contents

Abbreviations

BPTT	Back-Propagation Through Time
CASA	Computational Auditory Scene Analysis
COLA	constant-overlap-add
DC	Deep Clustering
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
FFT	Fast Fourier Transform
GRU	Gated Recurrent Unit
IBM	Ideal Binary Mask
ICA	Independent Component Analysis
iDFT	inverse Discrete Fourier Transform
iFFT	inverse Fast Fourier Transform
ISA	Independent Subspace Analysis
ISR	Image to Spatial Distortion Ratio
LSTM	Long Short-term Memory
MAE	Mean Absolute Error
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NMF	Non-negative Matrix Factorization
PCA	Principal Component Analysis
RIFF	Resource Interchange File Format
RNN	Recurrent Neural Network
SAR	Source to Artifact Ratio
SDR	Source to Distortion Ratio
SIR	Source to Interference Ratio
SNR	Signal to Noise Ratio
SPHERE	Speech Header Resources
SRT	Speech Reception Threshold
STFT	short-time Fourier Transform
VAD	Voice Activation Detection
WCE	Within Cluster Error

Chapter 1

Introduction

The goal of source separation is to separate a mixture into estimated sources. There have been many approaches to the task of separating signals in a variety of areas such as automated karaoke, object-specific equalizers, hearing prostheses and robust speech recognition [72]. Algorithms can use assumptions whether or not the signal is stereo or whether certain harmonic or redundant compositions exist [53, p. 1308].
The domain discussed in this thesis is the separation of speech. In 1953 the "Cocktail Party Problem" was coined by Cherry [8]. Even for humans it can be difficult to understand other speakers during a cocktail party due to the quantity of concurrently speaking people. In most cases though the impressive human auditory system is capable of understanding a single speaker among many others without problems. Miller [42] and Rosen, Souza, Ekelund, and Majeed [55] experimentally showed that the problem of separating voices is difficult even for humans because of the similarity of spectral and temporal features. Separating speech from other kinds of noise like a high frequency tone presents a less challenging problem for humans [42, p. 128].

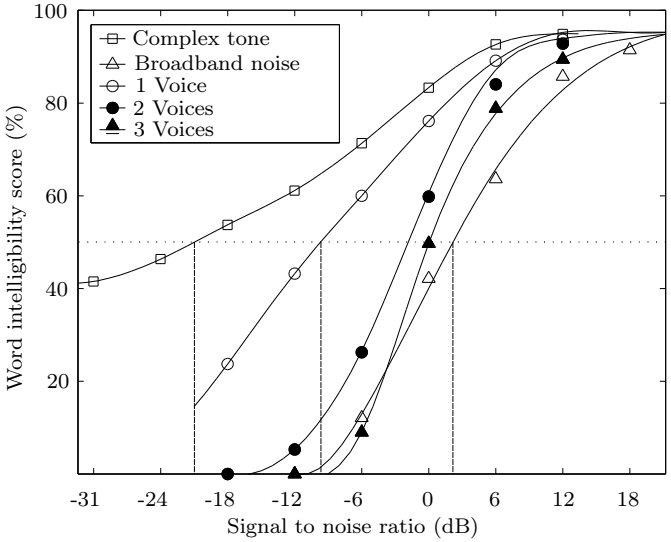


Figure 1.1: Speech intelligibility score for different kinds of noise (from [69, p. 2, 70, p. 7], redrawn from [42]). Complex tones are repetitive pulses of square waves with harmonically related frequencies. Broadband noise consists of randomly related frequencies over a broad spectrum. The properties of the 1, 2 and 8 voices signals are self-explanatory. The vertical dotted lines correspond to the -20, -10 and 3 dB Speech Reception Thresholds (SRTs).

Figure 1.1 shows the intelligibility for different kinds of noises with different Signal to Noise Ratios (SNRs). Speech inside a mixture with a 10-20 dB more powerful complex tone noise or one additional voice is still about 50% intelligible for humans. Wang and Chen [69] mention the concept of SRT which can be seen in Figure 1.1 at the intersection point of the intelligible score for each signal and the dotted line, projected onto the SNR axis. For example the interference with one voice has a SRT of about -10dB. They also highlight that there is a 23dB SRT difference between complex and broadband noise. This means that the ability of humans to understand voices when mixed with noise highly depends on the kind of noise.
Miller [42] also concludes that even though interference with high frequency signals doesn't reduce intelligibility as much as with low frequency ones, they are considered to be more annoying. The same applies to irregular interference. However, there is no evidence that annoyance reduces the intelligibility scores.
Note that the mixtures in Figure 1.1 are played in a laboratory situation with a monaural setup [42]. As we focus on monaural speech separation, this evaluation gives a good impression of what we try to achieve with an algorithmic approach to the problem. We will not make use of the "binaural advantage" (multichannel advantage) mentioned in [70, p. 7] in this thesis.
We can conclude that the separation of speech is not a trivial task for humans. Especially for hearing impaired individuals it can be even more challenging as the SRT can be expected to be 4-6 dB lower [14]. The development of an algorithm to perform the separation is also non-trivial which will be shown in Chapter 1.2 by summarizing the history of speech separation and its milestones.

1.1 Challenges in Speech Separation

Speech separation is a challenging problem because there is not a single perfect solution. Let $x_1(n_1)$ and $x_2(n_2)$ be the reference samples of two audio signals for $0 \leq n_1 < N_1$ and $0 \leq n_2 < N_2$, then the mixture signal is defined as $x = x_1 + x_2$ with the length $N = \min(N_1, N_2)$. The goal of speech separation is to split the signal $x(n)$ into the target reference signals. Unfortunately, an infinite amount of solutions exists to this problem, because this is a highly underdetermined task.
This ambiguity is even more severe if the separation is conducted blind which means that no prior knowledge about the mixture is available. This includes knowledge like the count, gender or age of the speakers.
Speech separation is called to be speaker-dependent if the speakers have to be the same for training and testing. Target-dependent means that the dominant speaker has to be the same during training and testing. Speakers who cause interference can be chosen arbitrarily. The most difficult goal for an algorithm is to be speaker-independent, which means that the algorithm also performs well if the speakers during testing are completely unknown. [69, p. 15]
Speech enhancement can be seen as a special case of speech separation. Enhancement means that a disturbing component of the audio should be removed. That component can be static noise, noise from cars or babble. Noise can be seen as an additional source in the context of speech separation. Nevertheless, it is often ignored in speech separation because the main goal is to get source estimations for speech. Therefore, a field which naturally profits from advances in speech separation is speech enhancement. Another profiting field is automatic speech recognition [51]. A signal with exactly one speaker and less noise is a better input for speech recognition than a noisy one.
In this thesis we try to achieve blind speech separation. Music or powerful background noise it not expected during training and testing of the Deep Clustering (DC) model. This is still a difficult challenge because we try to separate sources within the same domain. The next chapter focuses on the methods which have been tried in the past.

1.2 Earlier Approaches

For the past 6 decades it is true that the "Cocktail Party Problem" hasn't been solved properly [69, p. 2]. Nevertheless, recently impressive progress has been achieved by using Deep Neural Networks (DNNs). In this chapter we will go over earlier work and also cover the latest achievements in speech separation, which make use of DNNs.
The earliest methods for achieving a separation of audio used digital signal processing techniques. In [13] speech enhancement was achieved by using estimators in the time-frequency domain. According to a subjective comparison in 2007 this approach performs well compared to other methods proposed at that time [28]. It is noteworthy though, that all competing algorithms did not achieve to enhance the noisy audio significantly in the presence of multiple background speakers [28, p. 601].
Computational Auditory Scene Analysis (CASA) focuses on modeling the human auditory system in order to achieve goals like automatic speech and speaker recognition, music transcription, improving hearing prostheses or audio information retrieval [70, p. 13]. Compared to the other approaches mentioned, CASA is driven by the motivation to model the human [70, p. 29]. The other approaches are application-driven and focus on digital signal processing. An analogy for the difference between CASA and digital signal processing is given by the difference between computational vision and computer vision, where the former is concerned about modeling the human and the latter about practical processing [40]. Approaches in the field of CASA use spectrum, pitch and model based methods to separate speech [70, p. 106].
Later approaches use Independent Component Analysis (ICA) or its extension Independent Subspace Analysis [11, 7] in order to perform blind source separation. ICA exploits the assumption that the source components are statistically independent. The model of this source separation is $x = As$, where A is a $n \times \rho$ invertible mixing matrix with linearly independent columns, $s = \begin{pmatrix} s_1 & s_2 & \cdots & s_\rho \end{pmatrix}^\mathsf{T}$ is a vector which contains the sources and $x = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}^\mathsf{T}$ is a vector which describes the mixture, where $n \geq \rho$. The unmixing matrix $W \approx A^{-1}$ is calculated using x and the fact that the vectors in s are statistically independent [7]. The estimated source signals are defined as Wx. Several approaches exist in order to exploit this independence, but the model and assumptions are equal. Virtanen [68, p. 1067] notes that ICA is applicable for multichannel blind source separation, but can't be directly applied to monaural signals. Independent Subspace Analysis (ISA) does not require multichannel audio [68, p. 1067].
In 2007, Virtanen [68, p. 1071] achieved better separation performance than ICA and ISA using Non-negative Matrix Factorization (NMF). NMF approximately factorizes a non-negative matrix $V \in \mathbb{R}_+^{n \times m}$ into two non-negative matrices $W \in \mathbb{R}_+^{n \times \rho}$ and $H \in \mathbb{R}_+^{\rho \times m}$ [57, 36]. In digital signal processing NMF is used on the magnitude spectrum (see Equation 2.2). Therefore, the constraint of non-negativity is not a problem because the magnitudes in the frequency representation are already positive. The mixture is seen as a linear combination of the components W and H [57]. In [56] a supervised learning approach is presented which makes use of personalized phoneme dictionaries to prepare W.
Models like NMF and ICA are linear. Therefore, Huang, Kim, Hasegawa-Johnson, and Smaragdis [29] propose the use of more complex nonlinear DNNs and Recurrent Neural Networks (RNNs) with the goal of modeling voices better in 2014. This work represents one of the earliest uses of RNNs for speech separation. Neural networks offer multiple nonlinear layers which already proved valuable in tasks like speech enhancement. The RNN proposed in [29] takes for each time step t the spectral features as input and outputs the concatenation of two estimated sources $\hat{y}_1^{(t)}$ and $\hat{y}_2^{(t)}$ as seen in Figure 1.2. An additional step is needed such that the sum of the estimated sources is equal to the mixture. A binary or soft time-frequency mask like defined in Definition 3.3.1 must be computed from the estimated sources which can be element-wise multiplied with the mixture to get the final estimated sources. The approach achieved a 3.8-4.9dB improvement compared to NMF on the TIMIT [17] data set. Note that according to the definition in Section 1.1 this approach is not speaker independent as it uses the same speakers during training and evaluation.

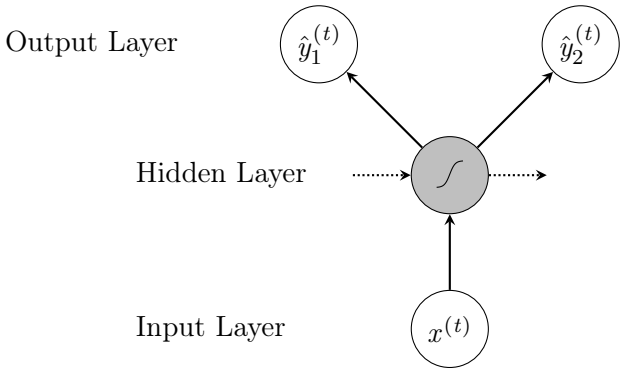


Figure 1.2: Structure of the layers in the RNN from [29]. The spectra for two speakers $\hat{y}_1^{(t)}$ and $\hat{y}_2^{(t)}$ is estimated at time step t. The input spectrum $x^{(t)}$ is the mixture with both speakers.

Instead of defining an Ideal Binary Mask (IBM), it is also possible to use the calculated magnitude spectra directly like proposed in [64]. Qian, Weng, Chang, Wang, and Yu [51] argue that the direct estimation of the spectrum is inferior because masks are more robust to changes of the input signals visible in the power spectrum.
A constraint of the so far mentioned deep learning approaches is that they are only able to separate k speakers when trained on k speakers. By introducing the DC method we will try to overcome this limitation.
Furthermore, the mentioned spectra and mask based methods suffer from the label ambiguity or permutation problem, which prevented progress for several years. The problem is that if the spectra or masks are calculated using a DNN like shown in Figure 1.2, it is unknown which of the estimated spectra $\hat{y}_1^{(t)}$ and $\hat{y}_2^{(t)}$ belongs to which speaker. It is not easily possible to specify that $\hat{y}_1^{(t)}$ always corresponds to the same speaker as the permutation of the spectra should be a different for the next time step $t + 1$. A naive approach to this problem would be to train the network that the first spectrum should always be the female speaker. This fails though if there are more speakers or same gender speakers are considered. [34, p. 4]
As we will see DC represents a state-of-the-art solution for speech separation. We will explore this approach in full detail to prove this statement in this thesis. The next section gives an outline of the whole thesis before we start to define DC properly.

1.3 Outline

The findings of this thesis include that the speech separation using Deep Clustering:

- allows to visualize the clustering and separation process
- and is speaker-independent,
- but suffers from noise in the training and evaluation data sets.

In order to make these contributions the thesis is structured in the following way. The first chapter focused on the motivation, idea and challenges behind speech separation. It also introduced the classical approaches to speech separation.
The second chapter covers the foundations for DC by discussing the discrete Fourier transform, deep neural networks (specifically recurrent neural networks) and k-means clustering.

The main chapter goes in detail about the state-of-the-art DC approach. We go over the aspects of DC in detail by splitting the method in several stages. The foundations of the previous chapter are used to describe the process of separation speech.
Chapter 4 focuses on an experimental evaluation of the implemented model. Visualizations of the clustering are presented by using Principal Component Analysis (PCA) and order selection in the sections 4.1 and 4.2 respectively. The constrains of the evaluation environment are described in detail to make the results reproducible. Practical methods are explained and evaluation results given.
Finally, a conclusion about the adaptability of deep clustering to real-world data is drawn.

Chapter 2

Foundations of Deep Clustering

As the DC approach consists of several stages like described in Chapter 3 several topics should be covered in order to provide a deeper understanding of the applied method. The following three sections provide the foundations for DC by introducing the discrete Fourier transform, neural networks and k-means clustering.

2.1 Discrete Fourier Transform

Signal processing has two categories, one of which is analog signal processing. Its objective is to process signals which are continuous in time. The other one is digital signal processing which processes discrete signals and is used throughout this thesis. A discrete signal is not a continuous waveform but a sequence of discrete values, where value corresponds to a time step. [38, p. 2]
In the context of digital audio a signal has a length N which corresponds to the amount of samples. Furthermore, there is also the sample rate which is defined as the amount of samples per second. As we are dealing with digital data each sample also has a sample size in bit and a data type. Therefore, a discrete audio signal can be characterized by Table 2.1.
The analysis, manipulation and synthesis of digital audio signals often requires to transform the signal to the frequency-domain. The Discrete Fourier Transform (DFT) is a mathematical procedure which determines the frequency content of a discrete signal. The DFT $X(m)$ of the time-domain signal $x(n)$ is defined as

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi nm/N}, \tag{2.1}$$

where m is the index of the frequency bin, N is the amount of samples, e is the base of the natural logarithm and i the imaginary number [38, p. 49-50]. The Fourier transform is a cornerstone in the analysis of signals and is used in a lot of fields. An infamous quote of William Thomson explains that "Fourier's Theorem is not only one of the most beautiful results of modern analysis, but it is

Characteristic	Symbol	Example
Amount of samples	N	16000
Sample rate	f_s	8 kHz
Sample size	-	16 bit
Sample data type	-	Float (Little-endian)

Table 2.1: Characteristics of an example discrete signal.

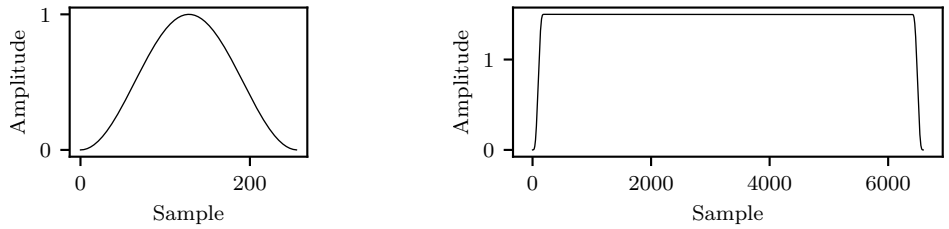
said to furnish an indispensable instrument in the treatment of nearly every recondite question in modern physics".
If $x(n)$ is a real valued signal, which means that the imaginary part is assumed to be zero, the DFT is symmetric. For simplicity, we assume that N is even as this will also be the case in the application of DC. For $m = 0$ to $m = \frac{N}{2}$ the result of $X(m)$ is redundant to the values for $m > \frac{N}{2}$. Thus for $0 \leq m < N$ the magnitudes $|X(m)|$ and $|X(N-m)|$ are equal [38, p. 63].
The non-redundant complex output of the DFT are $\frac{N}{2} + 1$ values which are called frequency bins. Unfortunately, the bins provide no information about the time interval in which the samples were taken. In order to scale the frequency bins according to an absolute frequency axis we need to scale the indices m by $\frac{f_s}{N}$. For example, if $f_s = \text{kHz}$ and $N = 16k$ then the frequency bin with an index of one corresponds to a frequency of $f = \text{kHz}$ [38, p. 67]. This scale allows determining dominant frequencies in the defined time interval. The maximum frequency in this case is $\frac{N}{2}\frac{f_s}{N} = \frac{f_s}{2}$ which is also known as the folding or Nyquist frequency [38, p. 28]. The first bin with $m = 0$ is also called the DC (direct current) bias and represents the mean of the input and can therefore be ignored in the analysis of the signal [24, p. 15]. The last bin $m = \frac{N}{2}$ can also be ignored because anti-aliasing filters should have removed the corresponding frequencies [24, p. 16]. The frequency resolution of the DFT is limited by the sample count within the time interval. [38, p. 51]
Another property of the DFT as defined by Lyons [38, p. 65] is linearity. If $x_1(t)$ and $x_2(t)$ are discrete signals and $X_1(t)$ and $X_2(t)$ their corresponding transforms then the DFT of the sum $x_{sum}(t) = x_1(t) + x_2(t)$ is $X_{sum}(t) = X_1(t) + X_2(t)$. The goal of source separation is to separate a signal like $x_{sum}(t)$ into its sources like defined at the beginning of the chapter. Because of the linearity of the DFT, it is possible to perform the separation in the frequency-domain as done in the deep clustering approach (see Section 3.2 and 3.5).
For analysis of speech we are interested in the power instead of the magnitudes which is defined as $X_{pwr}(m) = |X(m)|^2$. So the magnitudes of the signal are proportional to its power [38, p. 9]. A useful unit for the power spectrum is dB (decibel). The conversion from a linear power spectrum to a logarithmic one is defined as

$$X_{dB}(m) = 10\log_{10}(|X(m)|^2)\,\text{dB} = 20\log_{10}(|X(m)|)\,\text{dB}. \tag{2.2}$$

Using the decimal logarithm has an advantage when evaluating power differences. It allows greater resolution when the power levels are small [38, p. 487].
An effect which needs to be acknowledged is leakage. Leakage causes the results in the frequency-domain to be only an approximation to the originals signals before sampling. Frequencies which are not exactly centered in a frequency bin are spread over all other bins [38, p. 71]. In order to reduce this effect windows can be used [38, p. 80]. Instead of using a rectangular window, which is equal to using no window, a window like the Hann window can be used. Amplitudes in a time interval multiplied by this window are near zero in the begging and the end. An example for a window like this be seen in Figure 2.1a. This window will be used throughout this paper. It is noteworthy that there will be a processing gain by using a window. This means the mean of the power spectrum will be lower by using a window [38, p. 83]. In order to recover this loss a division by $\sum_{n=0}^{M} w_M(n)$ is needed [24, p. 15].
When separating speech using DC, the signal is modified in the frequency-domain. Once the signal has been modified, it is possible to convert it back to time-domain using the inverse Discrete Fourier Transform (iDFT):

$$x(n) = \frac{1}{N}\sum_{m=0}^{N-1} X(m)e^{-i2\pi mn/N}. \tag{2.3}$$

Apart from minor numerical errors the inverse is expected to be the same as the original input.
In 1965 Cooley and Tukey [10] defined the Fast Fourier Transform (FFT) which is an efficient implementation of the DFT to reduce the computational costs for a large amount of samples [38, p. 129]. Lyons [38, p. 131] makes clear that the FFT is not an approximation to the DFT but in



(a) A single Hann window.

(b) Sum of 100 overlapping windows. This corresponds to $\text{COLA}_M(n)$ where $0 \leq n < (100-1) \cdot H + M$.

Figure 2.1: Visualization of overlapping Hann windows of size $M = 256$ and hop length of $H = 64$: $w_M(n) = \frac{1}{2}[1 - cos(\frac{2\pi n}{M})]$ [22].

fact is equal. FFT and inverse Fast Fourier Transform (iFFT) are going to be used throughout the paper. Some implementations assume that $N = 2^k$, where k is a positive integer [38, p. 130], while others are more relaxed. If the input to the FFT is real valued, some efficient implementations do not calculate the negative frequencies. Therefore, it is required to scale the linear power spectrum by a factor of two [24, p. 15].
In discrete signals like speech the frequencies change over time. A usual approach to make the DFT dependent on the time is by creating short overlapping windows like demonstrated in Figure 2.1b [46, p. 714]. This short-time Fourier Transform (STFT) is defined as:

$$S(t, m) = \sum_{n=0}^{N-1} x(n) w_M^\alpha (n - tH) e^{-i2\pi mn/N} \tag{2.4}$$

[61]. Here $x(n)$ denotes the real valued discrete signal, w_M^α a window function of length M and H the hop size. The value of α is explained later. Note that for all values of the discrete signal which are outside of the window the product in the sum is zero. In conclusion, the STFT takes a discrete signal and creates a 2-dimensional function of time and frequency. In practice if $N - M$ is not a multiple of the hop size zero-padding is used to create time intervals of desired size. In order to inverse the result of the complex STFT, Equation 2.3 can be used. This yields the equation

$$x_t(n) = \begin{cases} \frac{1}{M} \sum_{m=0}^{M-1} S(t, m) e^{-i2\pi m(n-tH))/N} & \text{if } tH \leq n < tH + M \\ 0 & \text{otherwise} \end{cases}, \tag{2.5}$$

where $x_t(n) := x(n) w_M^\alpha (n - tH)$ and $\alpha > 0$. In order to reconstruct the original signal $x(n)$, overlap-add is required, which is defined by the following sum [45, p. 857, 59]:

$$\begin{aligned} \sum_{t=-\infty}^{\infty} x_t(n) w_M^\alpha (n - tH) &= \sum_{t=-\infty}^{\infty} x(n) w_M^{\alpha+1} (n - tH) \\ &= x(n) \text{COLA}_M(n) \end{aligned}, \tag{2.6}$$

where $\text{COLA}_M(n) = \sum_{t=-\infty}^{\infty} w_M^{\alpha+1}(n - tH)$. Note that the usage of infinite sums is possible here because $x_t(n)$ is defined on $\mathbb{Z}$. COLA_M is visualized in Figure 2.1b where the value of $\text{COLA}_M(n)$ is shown for $0 \leq n < 100 \cdot M$. Therefore, the reconstructed signal is given by

$$x(n) = \frac{\sum_{t=-\infty}^{\infty} x_t(n) w_M^\alpha (n - tH)}{\text{COLA}_M(n)}. \tag{2.7}$$

Window functions fulfill the constant-overlap-add (COLA) constraint exactly if and only if $\forall n \in \mathbb{Z} : \text{COLA}_M(n) = C$, where C is some constant. [61, 45, p. 857]. If the STFT $S(t, m)$ is modified

arbitrarily to create $\tilde{S}(t, m)$, then there is no discrete signal $\tilde{x}$ which represents the modified STFT. In order to mitigate this problem Griffin and Lim [21, p. 237 eq. 6] reduce the squared error between the modified $\tilde{S}(t, m)$ and the STFT of $\tilde{x}$ by using $\alpha = 1$.
Therefore, it is useful if a window function fulfills COLA since in this case $x(n) = \frac{x_t(n)}{C}$. If $C = 1$ then no division is needed. Note that it is required that $\forall n \in \mathbb{Z}$: $\text{COLA}_M(n)$ isn't zero. A periodic (DFT-even) Hann window fulfills COLA with a window size of $M + 1$ and hop size of $0.25M$ [45, p. 859]. The inverse STFT approach used in DC is the overlap-add method with $\alpha = 1$ and a three-quarter overlapping Hann window.
A close look at Figure 2.1b reveals that for samples at the beginning or near the end $\text{COLA}_M(n)$ is almost zero. Therefore, parts of the signal are basically lost. In this thesis samples of value zero are added to the start and end of signal x such that the first Hann window is at centered at tH. After reconstructing the signal, some samples have to be truncated.
One might ask why the rectangular window which obviously fulfills the COLA criterion is not a good choice. The reason is the already discussed leakage effect.
In this section the DFT and FFT were introduced and basic constrains covered. Based on this the STFT was introduced which is the main algorithm which will be used to create features and reconstruct discrete signals. Also, the main ideas behind choosing a window were covered.
In the thesis' implementation of DC the FFT implementation of the `NumPy` library is used [44]. The implementation of the STFT which uses the FFT from NumPy is found in the `librosa` library [41].

2.2 Deep Neural Networks

The basic instance of neural networks is the feedforward neural network. It is also called Multilayer Perceptrons (MLPs) and is a typical and basic model in modern machine learning. The goal of a MLP is to estimate some function f^* by learning from samples. For each sample x the function f^* gives a result $\hat{y} = f^*(x)$. The neural network tries to estimate f^* by using the model $y = f(x; \theta)$ such that $y \approx \hat{y}$, where θ are the learned weights. The notation $f(x; \theta)$ means that f is a function of x and is parameterized by θ. This notation is used throughout this section. By learning the function f^* the weights θ are adjusted incrementally. The goal is that the network does not only optimize the weights such that already seen inputs are mapped correctly, but that f also works for unseen inputs. This is also known as generalization. Usually a training set is used to optimize the weights and a testing or evaluation set is used to measure the performance. [18, p. 102]
We go shortly about the terminology which is used with neural networks. Multiple functions get chained together in order to build the model. For example the three functions f_1, f_2 and f_3 can represent a neural network if they are composed to one function $f(x) = (f_1 \circ f_2 \circ f_3)(x) = f_1(f_2(f_3(x)))$. Each of these functions represents one layer. Therefore, this network has a depth of three, where the third layers is called the output layer, the first input layer and the layer in between hidden layer. The number of the variables of each layer determines the width of it. The term "neural" is derived from neuroscience after observing patterns which reveal the representation ability of the brain. If a vector is passed to a layer, each variable in the layer is treated like a neuron, which can decide on its own what the output value should be. In the case of an artificial neural network the output value is determined by the weight and bias of the neuron. In practice, however, this can be seen as an analogy and only inspires the engineering of neural networks. [18, p. 163]
An intuitive way to introduce feedforward neural networks is to start with a linear model which is not able to properly approximate a nonlinear function like for example, x^2. In case of linear regression the model is $y = w^\mathsf{T} x + b$ [18, p. 106]. This model can be extended to use $\phi(x)$ instead of x directly, where ϕ denotes a nonlinear hidden layer. This yields the nonlinear model $y = w^\mathsf{T} \phi(x) + b$. One might question how ϕ should be constructed. The answer is that we actually leave this task to the neural network. Therefore, the model of the complete MLP is defined as $f(x; \theta, b, w) = \phi(x; \theta, b)^\mathsf{T} w$, where θ are the weights of a hidden layer and w the weights which map from the hidden layer to the output layer. At this point, engineers can also apply their domain

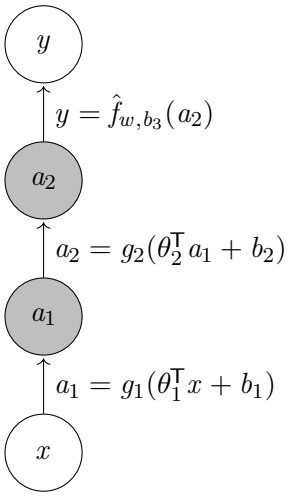


Figure 2.2: Architecture of a simple MLP with the computational steps needed in order to calculate the activations a_1, a_2 and y given the input y. In this example the weights are given by θ_1, θ_2, the biases by b_1, b_2 and the activation functions by g_1, g_2. The function $\hat{f_w}$ is not yet defined as it needs to be chosen carefully according the task of the perceptron.

knowledge to restrict the function family ϕ for each hidden layer. Nevertheless, this task is usually left completely to the neural network. [18, p. 165]
The function of a nonlinear hidden layer ϕ is defined as $\phi(x; \theta, b) = g(\theta^\mathsf{T} x + b)$ [18, p. 167]. This function involves a nonlinear activation function g. The qualification of activation functions and its utilization cases will be discussed in Section 2.2.1.
In Figure 2.2 a simple feedforward neural network is presented. In each hidden layer a specific activation function $g \in \{g_1, g_2\}$, weight matrix $\theta \in \{\theta_1, \theta_2\}$ and bias vector $b \in \{b_1, b_2\}$ is used. The computation in the output layer is indicated by a so far undefined function $\hat{f}_{w,b_3}$ which will be discussed in Section 2.2.1. Taking x as input the whole neural network outputs y, depending on the network architecture. Evaluating y is also called forward propagation, which is achieved by recursively computing the activations in each layer until the last one is reached. The process of training a model involves the optimization using back-propagation of the weights which will be discussed in Chapter 2.2.2 [18, p. 165]. Goodfellow, Bengio, and Courville [18] point out that the term back-propagation is often misunderstood. The back-propagation algorithm only computes the gradients whereas algorithms like gradient descent are responsible for updating the weights and biases.

2.2.1 Loss and Activation Functions

Machine learning algorithms require a loss function as feedback to evaluate how good an estimation is. As noted in the introduction, the goal is to approximate the function f to f^*. A loss function rates how well the model performs based on its current weights and biases. Minimizing the loss function and finding good weights and biases is an optimization problem. The difference between an optimization problem and machine learning is that the latter tries to generalize the estimated function such that it also works good on previously unseen testing data. Because the model of neural networks is nonlinear, the loss function is inherently nonlinear. This causes the loss function to be convex and therefore there is no guarantee that gradient descent will find a global minimum [18, p. 171]. One might ask how the weights and biases should be initialized. According to Goodfellow, Bengio, and Courville [18, p. 172] weights should be initialized to small random values and the biases to zero or small positive random values.

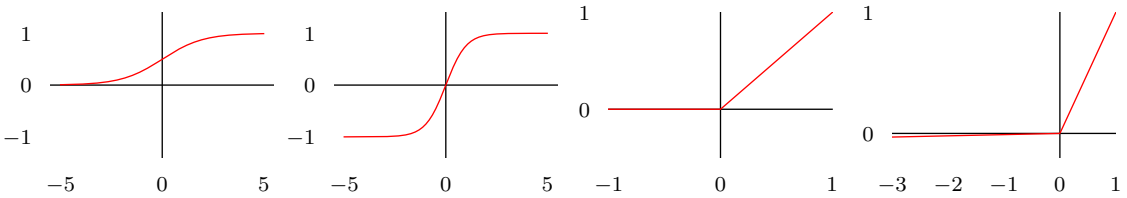
There are multiple loss functions to choose from, namely cross-entropy, Mean Squared Error (MSE) and Mean Absolute Error (MAE). Goodfellow, Bengio, and Courville [18, p. 175] notice that depending on the transformations in the output layer, the loss functions MSE and MAE can perform poorly. The commonly used cross-entropy loss is based on the maximum likelihood estimation and performs well in a lot of applications.
Depending on the task of the network the output layer has to be selected carefully because of the effect of saturation. Therefore, this chapter will find out in which way the function $\hat{f}_{w,b_3}$ in Figure 2.2 has to be chosen in order to complete the computation of y.
A function saturates if the function becomes very flat for specific input parameters. For example the derivative of the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$ is very small for positive and negative values. If a function saturates for specific values numerical errors will arise which make gradient-based algorithms perform poorly. [18, p. 173]
The first possibility is that the output layer is linear. In this case $y = \hat{f}_{w,b_3}(a_2) = w^\mathsf{T} a_2 + b_3$, which does not saturate. [18, p. 176]
If the task of the network is to perform binary classification to separate for example cats and dogs, then the logistic curve is a proper choice for the activation function when combining it with maximum likelihood. This yields the following output layer: $y = \sigma(w^\mathsf{T} a_2 + b_3)$. [18, p. 176-178]
Lastly, if the task of the network is to predict n different classes a softmax output layer should be considered the right choice. Due to the insignificance for the DC approach this will not be discussed in further detail. [18, p. 179]
The DC approach uses a fully connected layer as output layer. Therefore we don't have to deal with the effect of saturation when choosing the loss function in Section 3.3.1.
After choosing an activation function for the output layer, further functions have to be evaluated for the hidden layers. The selection of an activation function for each hidden layer is still a process of trial and error and an active research field. The usual method is to compare the efficiency of the network using different activation functions. It was already mentioned that in order to train a neural network it is necessary to determine the derivatives of the loss function and therefore also the derivatives of the activation functions. One could think that it in the context of neural networks it is required that these functions are differentiable. Nevertheless, the common rectifier activation function

$$g_{ReL}(x) = \max(0, x) \tag{2.8}$$

is not differentiable at $x_0 = 0$ because $\lim_{x \to 0} \frac{g_{ReL}(x) - g_{ReL}(x_0)}{x - x_0} = 0$ for $x < 0$ and for $x > 0$ it is 1. Goodfellow, Bengio, and Courville [18, p. 186] explain that in practice this is not required because gradients usually do not approach $x_0 = 0$ and therefore cases where the derivative is undefined are uncommon. Also when implementing these networks in software it is very likely that if the gradient is computed to be 0 that it actually is not really 0.
The linear rectifier unit as seen in 2.3c and defined in Equation 2.8 is a common activation function. The nature of this function makes the layer for the most part linear. Only negative values are clipped to 0. This linearity makes the model easier to optimize [18, p. 188]. As already mentioned in Section 2.2.1, it makes sense to initialize the biases to a small positive value such that neurons are active in the initial state. Goodfellow, Bengio, and Courville [18, p. 189] note that especially RNNs benefit from this activation function because of its linearity. The leaky linear rectifier unit as seen in Figure 2.3d

$$g_{LReL}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases} \tag{2.9}$$

represents a variation, which has a small slope if $x < 0$ with the result that there is always a non-zero small gradient [39].
Other activation functions are the logistic function and the hyperbolic tangent function. It was already mentioned that the logistic curve saturates for very positive and negative values (see Figure 2.3a). The same holds true for the hyperbolic tangent function as seen in Figure 2.3b. Therefore,



(a) Logistic Sigmoid (b) Hyperbolic Tangent (c) Linear Rectifier Unit (d) Leaky Rectifier Unit

Figure 2.3: Overview of the common activation functions used in MLPs.

the use of these is functions in MLPs is uncommon today. Nevertheless, the hyperbolic tangent activation function performs well in the DC approach. It is noteworthy that the tangent is more similar to a linear mapping compared to the logistic function because for the interval $[0, 1]$ the tangent is more similar to a linear function. [18, p. 189]

2.2.2 Back-propagation and Gradient Descent

If we want to calculate $y = f(x)$ using a trained feedforward neural network we iterate the network in forward direction. In the case of an untrained MLP the output is more or less random. Therefore, some kind of feedback is required to update the weights and biases in order to train it. In Chapter 2.2.1 the loss function was introduced which gives a metric on how well the network performs. The goal of this chapter is to give an intuition how the derivative of the loss function J in respect to the weights of each layer is determined. $J(W, B)$ gives the loss between the estimated y and the expected $\hat{y}$, where $W = (\theta_1, \theta_2, \ldots, \theta_L)$ contains the weights and $B = (b_1, b_2, \ldots, b_L)$ the biases for layer 1 to L. This allows algorithms like gradient descent to update the weights according to the derivatives. The adjustment for each weight depends on the derivative in respect to the current weights. [18, p. 198]
This chapter is structured in three parts. In the first one the back-propagation algorithm is presented for a simplified MLP. The second part notes that the algorithm can generalized for arbitrary MLPs. Last the idea behind gradient descent is explained.
A simple model which can demonstrate back-propagation is a MLP where each layer has only a single neuron. Note that in this case each layer has only a single weight and a single bias. Therefore, no multiplication of multi-element vectors is needed in the simple model. In order to visualize this, a computational graph is presented in Figure 2.4. The training of the network works by adjusting the weights and biases. Therefore, we want to calculate $\frac{dJ}{d\theta_l}$ and $\frac{dJ}{db_l}$ for each layer. A closer look at the computational graph reveals that the activation a_L of the last layer L depend on z_L and therefore also on the weight, bias and activation of the previous layer. [18, p. 201]
The core of the back-propagation algorithm is the chain rule which allows to calculate the derivative of a function $f = g \circ h$ if the derivatives of g and h are already known. Using this rule it is possible to determine the derivative of the loss in respect to the weight θ_L of the last layer L using the following equation:

$$\frac{\partial J}{\partial \theta_L} = \frac{\partial z_L}{\partial \theta_L} \frac{\partial a_L}{\partial z_L} \frac{\partial J}{\partial a_L}. \tag{2.10}$$

Let's recall that the formula of the hidden layer l is

$$a_l = g_l(z_l), \tag{2.11}$$

where

$$z_l = \theta_l^\intercal a_{l-1} + b_l \tag{2.12}$$

like shown in Figure 2.2. The term in the middle on the right side of Equation 2.10 is the derivative of the activation in respect to z_L, which is $g'_L(z_L)$. The first term on the right is equal to the

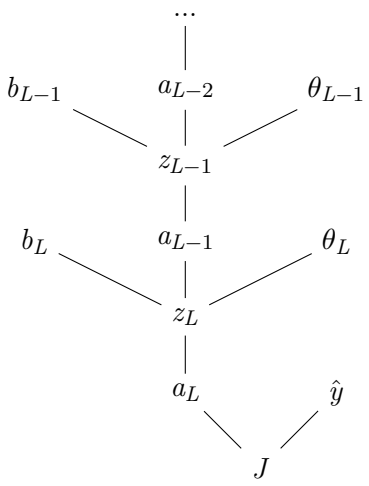


Figure 2.4: The computational graph of a multilayer neural network where each layer has exactly one neuron. L denotes the layer layer in the network. Therefore, θ_L is the weight, b_L the bias and $a_L = g(z_L)$ the activation of the last layer. J is the loss function which compares the activation with the expected results $\hat{y}$.

activation of the previous layer a_{L-1}. This is because z_L is defined as $z_L = \theta_l^\intercal a_{L-1} + b_L$ and therefore a_{L-1} is the slope.
In summary, Equation 2.10 can also be expressed as

$$\frac{\partial J}{\partial \theta_L} = a_{L-1}\delta_L, \tag{2.13}$$

where $\delta_L = \frac{\partial J}{\partial a_L} g'_L(z_L)$. Similarly if the derivative of the loss function in respect to the bias is

$$\frac{\partial J}{\partial b_L} = \delta_L \tag{2.14}$$

(The factor a_{L-1} is omitted here because $\frac{\partial z_L}{\partial b_L} = 1$).
After calculating the derivatives for the last layer we can calculate the derivatives for the hidden layers. Using the chain rule it is possible to calculate the derivative in respect to the activations of the previous layer:

$$\frac{\partial J}{\partial a_{L-1}} = \frac{\partial z_L}{\partial a_{L-1}} \frac{\partial a_L}{\partial z_L} \frac{\partial J}{\partial a_L} = \theta_L^\intercal \delta_L. \tag{2.15}$$

This equation can be used when calculating the derivative of J with respects to the weight of the layer $L-1$:

$$\begin{aligned} \frac{\partial J}{\partial \theta_{L-1}} &= \frac{\partial z_{L-1}}{\partial \theta_{L-1}} \frac{\partial a_{L-1}}{\partial z_{L-1}} \frac{\partial z_L}{\partial a_{L-1}} \frac{\partial a_L}{\partial z_L} \frac{\partial J}{\partial a_L} \\ &= \frac{\partial z_{L-1}}{\partial \theta_{L-1}} \frac{\partial a_{L-1}}{\partial z_{L-1}} \frac{\partial J}{\partial a_{L-1}} \\ &= a_{L-2}\, \theta_L^\intercal \delta_L\, g'_{L-1}(z_{L-1}). \end{aligned} \tag{2.16}$$

Using the chain rule we can recursively determine all the derivatives in the computational graph by repeating the above computation from $l = L$ until $L = 1$. This is the main idea of back-propagation. Although the algorithm was only demonstrated for scalar values it also applies to

vectors and tensors of arbitrary dimension [18, p. 199]. If dealing with vectors the chain rule can be expressed as

$$\nabla(x, z) = (\frac{\partial y}{\partial x})^{\mathsf{T}} \nabla(y, z), \tag{2.17}$$

where $x \in \mathbb{R}^m, y \in \mathbb{R}^n, g : \mathbb{R}^m \to \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}$, $\frac{\partial y}{\partial x}$ the $n \times m$ Jacobian matrix and $z = f(y)$ with $y = g(x)$ [18, p. 199]. This allows the weight vectors to have an arbitrary dimension in a generalized version of gradient-descent.
After calculating the derivatives it is possible to update the weights and biases of each layer l using gradient descent

$$\theta_l \leftarrow \theta_l - \eta \frac{\partial J}{\theta_l} \tag{2.18}$$

and biases

$$b_l \leftarrow b_l - \eta \frac{\partial J}{b_l}, \tag{2.19}$$

where η is the learning rate [43, Chapter 2]. Even though it is possible to update the weights after each input it can also be done in batches. This means the weight adjustments are collected over b_s inputs before the model is actually updated. [18, p. 272]
In this section we covered how the back-propagation works. Armed with that knowledge we can discuss how RNNs work in the next section.

2.2.3 Recurrent Neural Networks

The network architecture of DC is based on RNNs. To be more accurate a bidirectional Long Short-term Memory (LSTM) architecture was chosen in [27] which will be covered later in this section. The difference between feedforward and recurrent neural networks is that the latter allows recurrent connections between hidden neurons whereas the former doesn't. The reason for this architectural decision is that it is quite difficult for a MLP to model time-series data. The connections between neurons within a layer allows interactions between present and future inputs. [19, p. 22].
Especially for audio signals containing speech, temporal context is important. If a recurrent network is fed with audio data it is beneficial if the output of the time step $t + 1$ depends on the activations of the time step t.
The recurrent network expects a sequence of features as input. Therefore, the input is not only a vector of features, but a matrix $I = \begin{pmatrix} x^{(1)} & x^{(2)} & \dots & x^{(t)} \end{pmatrix}^{\mathsf{T}}$ where each row corresponds to a feature vector. Note the superscript in brackets is not an exponent but just another index. For each time step $t = 1$ until $t = \tau$ the vector $x^{(i)}$ is fed into the RNN.
In Figure 2.5 a visualization of a RNN can be viewed where the connections between the hidden layers represent the time-dependence between samples in the sequence of inputs. The shape of the input in each time step is still the same as in the MLP shown in Figure 2.2.
The forward-propagation equation for each layer l is

$$a_l^{(t)} = g_l(z_l^{(t)}) \tag{2.20}$$

with

$$z_l^{(t)} = U a_{l-1}^{(t)} + W a_l^{(t-1)} + b_l, \tag{2.21}$$

where g_l is an activation function, b_l the biases, θ_l input-to-hidden weights, w_l hidden-to-hidden weights, $a_{l-1}^{(t)}$ the activations of the previous layer and $a_l^{(t-1)}$ the activations of the previous time step of the same layer [18, p. 370]. Note that for each time step the initial input $a_0^{(t)}$ of the network is $x_{(t)}$. The Equation 2.20 is quite similar to Equation 2.11. The difference is that the activation of each layer depends not only on the previous layer but also on the activations in the past.
The forward-propagation algorithm starts with $t = 1$ and recursively determines the activations until $t = \tau$. For simplicity he initial value of $a_l^{(0)}$ is considered to be zero [19, p. 23]. The output of the last layer the can be computed using a fully connected layer which follows Equation 2.11.

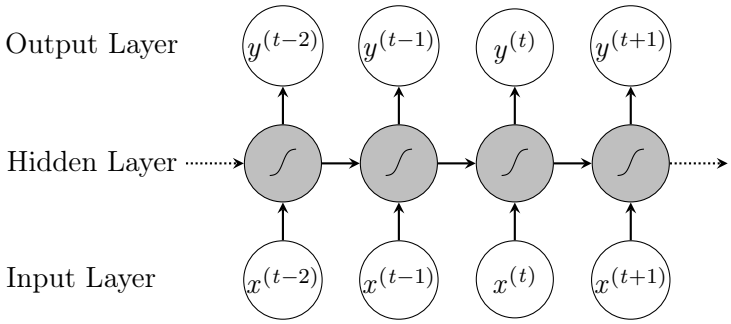


Figure 2.5: Unfolded architecture of a simple RNN over time. Each column is similar to the MLP structure of Figure 2.2 and represents a single time step. This figure shows an extract of a RNN for the time steps $t-2$ until $t+1$. Each hidden layer depends on the hidden layer one time step earlier.

A possible back-propagation algorithms for RNNs is Back-Propagation Through Time (BPTT). This algorithm works similar to the procedure described in Section 2.2.2. It applies the back-propagation algorithm for MLPs at each time step from $t = \tau$ to $t = 0$ and determines the updates for the weights in each step. [73, p. 450] Similar to the MLPs the weight updates can happen here also in batches of size b_s.

Long Short-term Memory

A common problem when dealing with RNNs is the vanishing gradient problem. The use of LSTM cells together with recurrent networks aims to solve this problem.
The vanishing gradient problem prevents the formation of long-term dependencies between samples in a sequence [19, p. 38]. This is because there is no memory and control mechanism which defines how volatile the calculated results are. This effect is best demonstrated using a graphic which can be seen in 2.6. As time progresses the influence of the input slowly decays. In the following paragraphs a cell structure is presented which counteracts this effect. The suggested cell structure controls how influential the activations of the input and hidden layers are. Furthermore, a mechanism of forgetting is required.
First we discuss how a cell "forgets". This is visualized by the red part in Figure 2.7. The input and the activations from the previous time step are combined to get values between 0 and 1 which

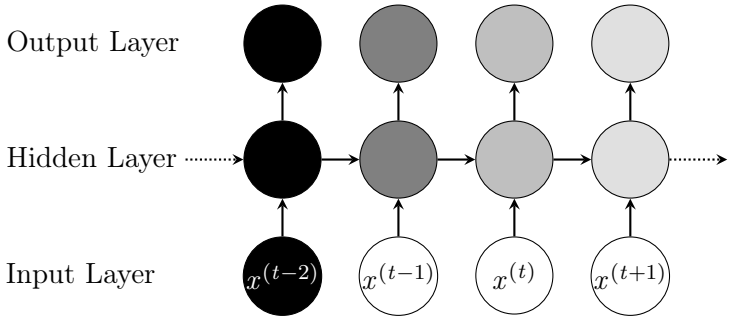


Figure 2.6: Visualization of a the vanishing gradient effect. The monochromic gradient of each layer represents the sensitivity to the inputs $x^{(t-2)}$ to $x^{(t+1)}$. The influence from the inputs vanishes slowly over time. This figure is redrawn from [19, p. 38].

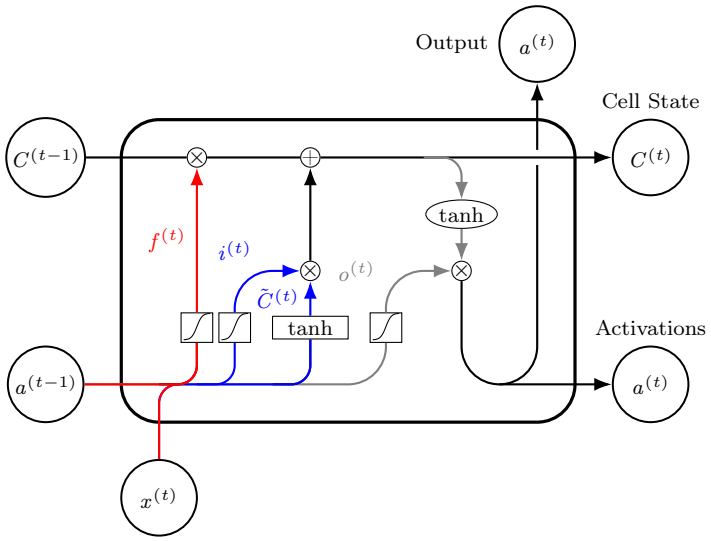


Figure 2.7: Visualization of a LSTM cell, where f is the forget gate, i is the input gate and o is the output gate.

will later be multiplied by the previous cell state $C^{(t-1)}$:

$$f^{(t)} = \sigma\Big(U^f x^{(t)} + W^f a^{(t-1)} + b^f\Big), \tag{2.22}$$

where σ is the logistic function, $x^{(t)}$ is the input vector, $a^{(t-1)}$ the activations from the previous layer and U^f, W^f and b^f the weights and biases of the forget gate. Note that these calculations can be seen as layers within a cell. [18, p. 399]
Secondly the LSTM needs a way to determine which new information from the input should be added to the cell state. This part is colored blue in Figure 2.7. A logistic and hyperbolic tangent layer are responsible for calculating the state update. The logistic layer determines which values should be updated:

$$i^{(t)} = \sigma\Big(U^i x^{(t)} + W^i a^{(t-1)} + b^i\Big). \tag{2.23}$$

The tangent layer calculates the values to be added:

$$\tilde{C}^{(t)} = \tanh\Big(U^C x^{(t)} + W^C a^{(t-1)} + b^C\Big). \tag{2.24}$$

The new state can be calculated using the previous two results:

$$C^{(t)} = f^t \odot C^{(t-1)} + i^{(t)} \odot \tilde{C}^{(t)}, \tag{2.25}$$

where $\odot$ is the element-wise product. The new state depends on the previous state $C^{(t-1)}$ multiplied by the forget gate $f^{(t)}$ and the modified inputs $\tilde{C}^{(t)}$ multiplied by the input gate $i^{(t)}$. Note that the gates basically act like a filter here. Therefore, the cell is able to remember and also forget. [18, p. 399]
Finally, the output of the cell needs to be calculated. The output gate is implemented as a logistic layer which is filtered using the hyperbolic tangent of the current state:

$$o^{(t)} = \sigma\Big(U^o x^{(t)} + W^o a^{(t-1)} + b^o\Big), \tag{2.26}$$

$$a^{(t)} = o^{(t)} \odot \tanh(C^{(t)}). \tag{2.27}$$

The resulting activations $a^{(t)}$ are the activations of the LSTM cell at time step t. [18, p. 399]
Like RNNs, LSTM networks can use the BPTT algorithm for back-propagation of the loss [19, p. 43]. A difference is that the computation of the derivatives consumes more computing resources are more terms are involved.

Bidirectional Recurrent Neural Networks

Vanilla RNNs and vanilla LSTM networks are limited in a sense that they can only look into the past. By using a bidirectional structure it is possible to overcome this limitation. In context of audio analysis with DC it is beneficial that the current output depends not only on the past but also the future activations in order to have a broader context of the features. Goodfellow, Bengio, and Courville [18] also notes that neural networks for speech recognition benefit from being bidirectional because the interpretation of current phoneme can depend on the next phonemes.
Bidirectional RNNs were discovered by Schuster and Paliwal [58] in 1997 and have proven very successful in a variety of tasks [19, p. 25].

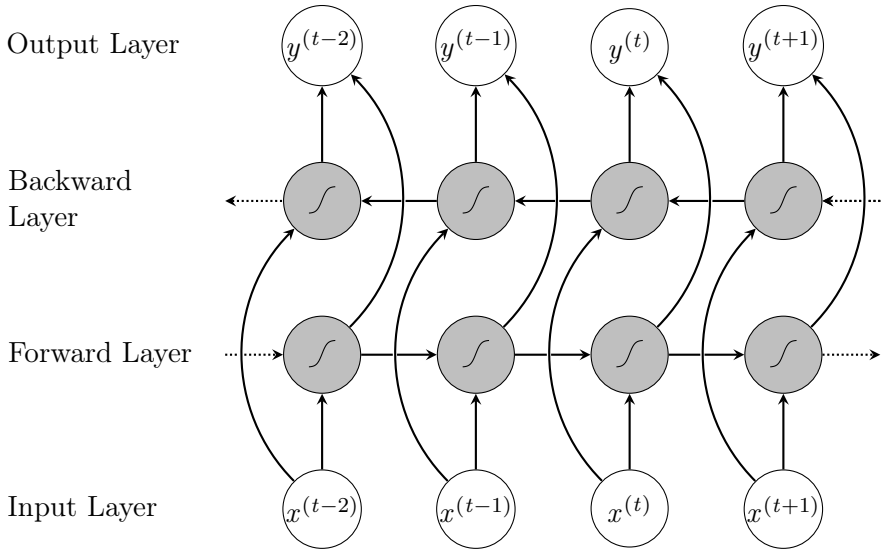


Figure 2.8: Unfolded architecture of a bidirectional RNN over time with one forward and one backward layer.

A bidirectional RNNs combines moving forward through time with moving backward through time. In the former case the sequence I is iterated from $t = 1$ until $t = \tau$ and in the latter from $t = \tau$ until $t = 1$. In summary, each input vector is processed twice and yields a separate output vector. The output layer is now able to depend on the forward an backward layer which allows the output $y^{(t)}$ at time t to depend on past and future activations. A visual representation of the architecture can be seen in 2.8 where for each time step the vector of the input layer is processed in the forward and backward layer.
The back-propagation in bidirectional RNNs networks is similar to that of ordinary RNNs. The first step is to calculate the derivatives of the output layer for each time step. After that, the derivatives for the forward and backward layer can be calculated.
The usage of LSTM cells in a bidirectional RNN produces a bidirectional LSTM network. Therefore, there are no significant differences in forward or back-propagation when using LSTM cells.

2.3 K-Means Clustering

Lloyd's algorithm is one of the most wildly used clustering algorithm in practice according to a report of 2002 [6]. The algorithm does not offer quality assurance but it is simple. In this section we are going to define the algorithm which is usually termed k-means.
The goal of k-means is to cluster the vector set $\mathcal{X} \subset \mathbb{R}^d$ of n vectors given an integer k which denotes the amount of expected partitions. The result of this operation is a set $\mathcal{C} \subset \mathbb{R}^d$ of k centers. Like neural networks this algorithm also includes a loss function which is the target for optimization:

$$\gamma = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2. \tag{2.28}$$

The objective is to minimize for each $x \in X$ the distance to its nearest center. Note that even for $k = 2$ this problem is NP-hard [3]. As the perfect optimum γ_{min} is not needed in practice the Lloyd's algorithm only gives a reasonable good approximation. [4]
The algorithm works by first selecting k vectors as initial centers: $\mathcal{C} = \{c_0, c_1, \ldots, c_{k-1}\}$. The set of clusters $\mathcal{K} = \{\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_{k-1}\}$ holds for each $0 \leq i < k$ a set of vectors $\mathcal{K}_i$. Each vector $x_j \in \mathcal{X}$, where $0 \leq j < n$ is now assigned to its nearest center c_i in $\mathcal{C}$ by adding it to the set $\mathcal{K}_i$. Thereafter, for each $0 \leq i < k$ the center c_i is updated according to its center of mass:

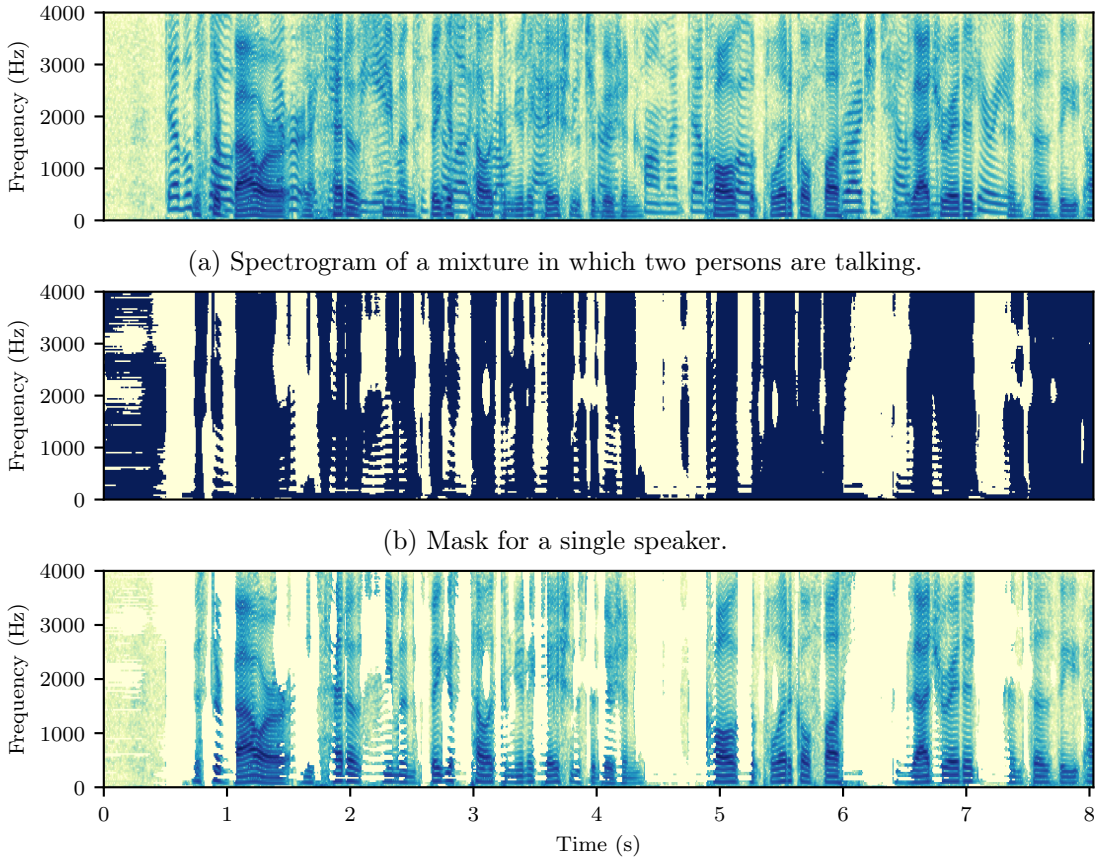
$$c_i = \frac{1}{|\mathcal{K}_i|} \sum_{x \in \mathcal{K}_i} x. \tag{2.29}$$

The process of calculating $\mathcal{K}$ and updating the centers $\mathcal{C}$ accordingly is repeated until $\mathcal{C}$ no longer changes. γ is decreasing in each step monotonically. [4, p. 2]
$\mathcal{C}$ no longer changes if the difference over time is less than a specified tolerance. It makes also sense to define a maximum amount of steps in order to define an upper boundary for the execution time which is lower than the maximum of k^n steps [4, p. 2].
It is not guaranteed that the algorithm will approach a global minima given the randomly selected initial centers. This is due to the fact that the algorithm is repeated with different centers and the results with the smallest loss are chosen.
Although the naive approach to select the initial centers randomly according to the uniform distribution works well, more sophisticated methods like in k-means++ [4] can be used to determine the initial centers in a wiser way. K-means++ works by randomly selecting vectors one after the other as initial centers. The difference is that each vector has a different probability based on how far it is to the closest center already determined. Empirical results show that the initialization in k-means++ yields better results in less time for specific data sets.

Chapter 3

Deep Clustering Algorithm

The Deep Clustering approach was first introduced by Hershey, Chen, Roux, and Watanabe [27]. There exists another publication of the same paper [26], which will not be discussed as they share the same results. Instead of solving the problem directly by estimating a spectrum or mask, it is treated as a clustering problem. Each time-frequency bin belongs to exactly one of k speakers. The clustering can be visualized by assigning each time-frequency bin a specific color which denotes the belonging of the speaker as shown in Figure 3.1.



(a) Spectrogram of a mixture in which two persons are talking.

(b) Mask for a single speaker.

(c) Separated spectrogram, which was created by applying the mask in Figure 3.1b.

Figure 3.1: Spectrogram of a mixture and a binary mask. The product of the two is shown in the last spectrogram. Dark blue is equal to one whereas the yellow parts correspond to zero. Each point corresponds to the power of the given frequency and time.

A formal definition of the speech separation using DC is the following: Let $x(n) \in \mathbb{R}$ be the samples of a mixture with k concurrent speakers, where $0 \leq n < N$. Deep Clustering computes the estimated sources $(x_i)_{i=0}^{i=k}$ such that $\sum_{i=1}^{k} x_i = x$. The constraint that the sum of the estimated sources must be equal to the mixture is fulfilled by using a mask like the IBM (see 3.3.1).
A RNN calculates for each time-frequency bin a vector in a lower dimensional space. This mapping is also called an embedding. These vectors can be clustered such that the vectors of the same speakers are close together. Using the labels of the clusters it is possible to create an IBM. Because vectors of the same speaker should be assigned to the same cluster, the permutation problem is circumvented. As seen in Figure 3.2, the DC approach can be structured into multiple stages. We will shortly go over these stages to give an overview of the method according to [27].

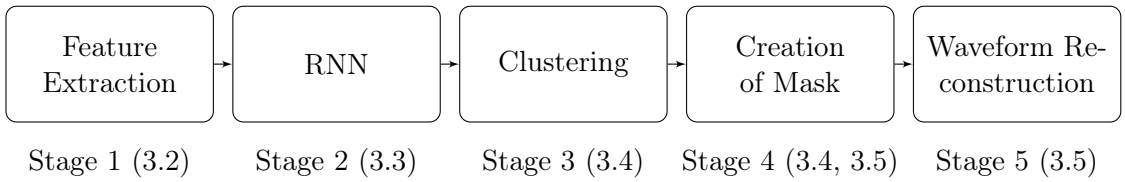


Figure 3.2: The five stages of Deep Clustering with references to the sections where they are discussed.

The **Feature Extraction** stage is responsible for creating the input features for the neural network. The input mixture is transformed to the time-dependent frequency representations using STFT. For a given discrete input signal $x(n)$ the logarithmic power spectrum is calculated. At this point it is possible to adjust the features using feature engineering. More details on how to further modify the input can be found in Section 3.2.
The **RNN** stage calculates for each time-frequency bin a d-dimensional vector using a RNN. The space of these vectors is also called an embedding space. The input of the network is a matrix which contains the time-frequency information over a specific time frame. The output of the network is exactly one d-dimensional vector for each time-frequency bin.
The **Clustering** stage determines the centers of the resulting clusters in the embedding space. Lloyd's algorithm is used in order to perform k-means clustering [37], as it represents one of the most wildly used algorithm for this task [6]. The centers are initialized using k-means++ [4]. Clustering can be performed locally or globally. The former means that clustering is performed on the output of the neural network of a single iteration. The latter means that the clustering happens after all embedding vectors of all time-frequency bins have been collected. The difference between these methods will be discussed further in Section 3.4.
In the **Creation of Mask** stage a mask is calculated by using the label information retrieved from clustering. The DC method assumes that in each time-frequency bin exactly one voice is dominant. Therefore, each bin can be assigned to one of k sources. For each speaker $0 \leq i < k$ a mask M_i is constructed in such a way that its multiplication with the spectrum of the mixture yields the estimated speech signal of the i-th speaker. One might say that calculating the mask is like adding another dimension to the time-frequency representation. In addition to the time and frequency dimension, a new dimension is added which denotes the belonging to a specific source.
The final stage is the **Waveform Reconstruction** of the separated spectra. The DC method uses only the amplitude information of the STFT. In the case of speech the amplitudes are more important than the phase. Because of that, it is sufficient to use the phase information from the mixture, even though reconstructing the phase could improve the overall performance [69, p.11]. To get a discrete signal from the time-frequency representation the inverse STFT is used. Apart from numerical errors, there is no error introduced from the inverse STFT of the unmodified frequency representation from the mixture. Because the window which was used when extracting the features

fulfills the COLA criterion, the numerical errors are negligible. In conclusion, the quality of the mask is responsible for the success of the source separation.

3.1 Expectations of Deep Clustering

We will shortly highlight the expectations of the Deep Clustering speech separation method. As described Section 1.1 the label permutation problem should be solved. Furthermore, the approach should be able to separate more than 2 speakers. The DC approach promises to separate a variable amount of voices from each other. This holds true even when the network is trained on mixtures of two speakers and tested on mixtures with three speakers. The separation also does not depend on knowing the speakers before training. [27]
In addition to the mentioned goals the separation should also work on real-world data. This will be evaluated in Chapter 4 by using different data sets. Mixtures with broadband noise could be difficult to separate because of its presence in a great range of frequency bins. Especially if the noise overlaps with speech in the 4 kHz range, the task becomes more difficult.
As the goals and expectations for the DC approach are set we will discuss the method in detail in the following sections. The sections 3.2, 3.3, 3.4 and 3.5 deal with the inference of the separated audio signals and assume a trained network. In Section 4.6 we will discuss how the network should be trained to achieve the expected results.

3.2 Pre-processing and Feature Extraction

Before we define the exact model in Section 3.3 the input features should be discussed. Recall that the input of the network should be a mixture and the output embeddings. In this section we go over the process of preparing a mixture such that it can be fed into the recurrent neural network. The first step is to load the samples $x(n)$ of the monaural discrete mixture signal for $0 < n < N$. The DC method only uses a single channel as described in Chapter 1.1. In order to determine the duration of the audio it is required to know the sampling rate f_s. The sampling rate is fixed in this paper to $f_s = 8$kHz. According to the Nyquist sampling theorem this allows a maximum frequency of 4 kHz to be sampled. This maximum frequency offers sufficient intelligibility because the resulting bandwidth is acceptable for vowels spoken by most people [23]. Furthermore, it also limits the amount of computational resources needed [27, p. 33]. The duration in seconds of the signal x is given by $N \cdot f_s$.
Next the STFT needs to be determined which expects the window length $M = 256$, hop size $H = 0.25M$ and a window function as parameters [27]. In this paper the Hann window function is used like presented in Figure 2.1a[1]. The notes of Section 2.1 apply in order to allow a reconstruction using the overlap-add method. This window length corresponds to a duration of 32 ms and a hop duration of 8 ms. The spectrum $S(t, m)$ is defined for $0 \leq t \leq \frac{N-M}{H}$ and $0 \leq m \leq \frac{M}{2}$. It is assumed x has at least M samples.
At this point it is required to make sure that the spectrum does not contain zeroes by calculating $S_+(t, m) = \max(|S(t, m)|, \epsilon)$, where $\epsilon = 1e{-}10$ is a constant which denotes the minimal magnitude. This allows the definition of the power spectrum according to Equation 2.2:

$$S_{pwr}(t, m) = 20 \log_{10}(S_+(t, m)) \,\text{dB}. \tag{3.1}$$

In Section 2.1 it was noted that the signal should be scaled in order to recover energy which is lost due to windowing or missing negative frequencies. This is not required here because the scaling of the input does not influence the performance of the network.

[1] It is assumed that the original paper also uses the Hann window function even though it specifies a squared Hann window. This is due to the common parameter $\alpha = 1$ which squares the window function.

A beneficial step is to apply Voice Activation Detection (VAD), which allows discarding quiet segments which are not important for the separation procedure. VAD is a common pre-processing step and can be done in time-domain and frequency domain [2]. Because we already operate in the frequency domain, it is straight forward to do it in this domain. A simple approach is to define a binary mask VAD(t, m) which defines which time-frequency bin is active and which is not. Active means that the power of that bin is above a certain threshold. The threshold can be chosen to be 40 dB below the maximum value of the power spectrum.
Furthermore, normalization of the mixture is a beneficial step. The neural network input is normalized according the mean μ and standard deviation σ of the training data set which is determined according to the process in Section 4.5. Using the definition of the standard score [35, p. 1018]

$$\hat{X}(t, m) = \frac{X(t, m) - \mu}{\sigma} \tag{3.2}$$

it is possible to normalize the values of the time-frequency spectrum $S_{pwr}(t, m)$.
Last the time-frequency power spectrum is segmented into frames where each frame has a specific maximum amount of windows $\#_w$. This is due to the limited amount of data a RNN is able to process at once. In Section 2.5 the input matrix I was mentioned which contains the feature vectors. The matrix for the first frame is

$$I = \begin{pmatrix} S_{pwr}(0,0) & S_{pwr}(0,1) & \dots & S_{pwr}(0, \frac{M}{2}) \\ S_{pwr}(1,0) & S_{pwr}(1,1) & \dots & S_{pwr}(1, \frac{M}{2}) \\ \dots & \dots & \dots & \dots \\ S_{pwr}(\#_w,0) & S_{pwr}(\#_w,1) & \dots & S_{pwr}(\#_w, \frac{M}{2}) \end{pmatrix}.$$

An important constant is the batch size b_s which was already mentioned in 2.2.2. During training b_s matrices are fed into the neural network at once. During the testing phase the batch size is expected to be one, as we want the outputs for a single frame.

3.3 Network Architecture

As already described in the beginning of the chapter the goal of the recurrent network in DC is to determine a d-dimensional vector for each time-frequency bin. The network architecture makes use of 2 bidirectional LSTM layers and a single feedforward layer with tanh as activation function. The paper by Hershey, Chen, Roux, and Watanabe [27, p. 33] also evaluates the logistic function but no significant difference was found in the performance of the method. In contrast to LSTM cells, there is also the possibility to use Gated Recurrent Unit (GRU) cells. They offer a simpler model and lacks for example an output gate [9]. Chung, Gülçehre, Cho, and Bengio [9] showed empirically that the performance in on pair for specific tasks. The results in Section 4.6 show that using an LSTM network is beneficial for DC. Furthermore, dropouts are applied to the forward and backward layers of the RNN like discussed in Section 4.6.
The actual input I_{b_s} of the neural network is a $b_s \times \#_w \times \frac{M}{2} + 1$ matrix. Each batch has b_s frames where each frame has $\#_w$ windows according to Equation 3.2. The duration of such a frame is approximately 824 ms which is about the length of a spoken word [27, p. 33]. Each window holds the $\frac{M}{2} + 1$ frequency bins. The matrix I_{b_s} has the correct shape to be an input of a recurrent LSTM layer.
The output of the bidirectional layers are concatenated and reshaped such that they can be input for the final feedforward layer. After concatenating and reshaping the output of the bidirectional layer it has the shape $q \times 2 \cdot 600$, where $q = b_s \#_w$. The output of the feedforward layer has the shape $q \times (\frac{M}{2} + 1)d$ which is reshaped to a $p \times d$ matrix, where $p = q(\frac{M}{2} + 1)$. For $0 \leq i < p$ each d-dimensional vector v_i is normalized in a last step such that:

$$|v_i|^2 = \sum_{i=0}^{d-1} v_i^2 = 1, \tag{3.3}$$

where $|.|$ is the Euclidean norm.
Therefore, the final matrix $V \in \mathbb{R}^{p \times d}$ assigns each of the p time-frequency bins a normalized d-dimensional vector. The matrix $VV^\mathsf{T} \in \mathbb{R}^{p \times p}$ is called the estimated affinity matrix. When training the network, the weights will be optimized such that the estimated affinity matrix is most similar to the ideal binary affinity matrix YY^T. What the training target YY^T means and why it is important for the success of the clustering is explained in the next section.

3.3.1 Loss Function

The training target of the neural network is the ideal affinity matrix $YY^\mathsf{T} \in \mathbb{R}^{p \times p}$, where p is the amount of frequency bins which should be assigned to one of k speakers.
This matrix is created from a target partition $Y \in \mathbb{R}^{p \times k}$ which is known during training. For $0 \leq i < p$ and $0 \leq j < k$ the value of y_{ij} is defined to be the j-th element of the i-th row of Y. That value belongs to the cluster j and therefore to the speaker j if and only if y_{ij} is equal to 1. The vector $y_i = (\, y_{i,0}\ y_{i,1}\ \cdots\ y_{i,k-1}\,)$ is a one-hot vector which has a 1 at the index which denotes the cluster belonging. The multiplication with its transpose yields an affinity matrix with an interesting property. For $0 \leq i, j < p$ the element $(YY^\mathsf{T})_{ij}$ is equal to 1 if the bins i and k belong to the same cluster and 0 if otherwise. Therefore, the affinity matrix is a symmetric square matrix of size p, where the diagonal consists only of ones.
With that knowledge it is possible to define a loss function J which reaches zero when the $p \times p$ estimated and ideal affinity matrices VV^T and YY^T are equal:

$$\begin{aligned} J(Y, V) &= ||\, VV^\mathsf{T} - YY^\mathsf{T}||_F^2 \\ &= \sum_{\substack{i=0,j=0 \\ y_i = y_j}} \big(|v_i - v_j|^2 - 1)\big) + \sum_{i=0,j=0} (v_i^\mathsf{T} v_j)^2, \end{aligned} \tag{3.4}$$

where $||.||_F^2$ is the squared Frobenius norm which sums over the squared difference between the entries of the matrices. The transformation into two terms can be reviewed in Appendix A.1. In Equation 3.4 the first term is directly related with the k-means objective from Equation 2.28 and pulls the embeddings of the same speakers together. The second term spreads the vectors apart and thus avoids trivial solutions [27, p. 32].
This means if the loss is small when the k-means algorithm is able to reach a small γ on the embeddings in V.
To summarize, if the network is trained properly (see Chapter 4.6) such that the difference between VV^T and YY^T is small then the calculated embeddings v_i and v_j have a small distance if they belong to the same speaker.
It is noteworthy that the $p \times p$ matrices VV^T and YY^T are constructed in two different ways. In the former case the matrix V consists of p embeddings. In the latter case Y consists of p one-hot vectors which denote the speaker belonging. In each case the multiplication by its transpose yields two matrices which can be compared.
The label permutation problem which was mentioned in Section 1.2 is circumvented here because embeddings from the same speaker will always have a small distance to each other. By using the presented loss function the neural network doesn't learn which frequency bin belongs to which speaker but only learns to put vectors from the same speaker close together in the embedding space.
The loss function in Equation 3.4 consumes quite a lot of memory because $p \times p$ matrices are constructed. A matrix which contains the time-frequency bins for 100 windows would already have $166 \cdot 10^6$ entries. Fortunately, an efficient implementation exists which exploits the fact that the objective has a low rank:

$$J(Y, V) = ||\, V^\mathsf{T} V||_F^2 - 2||\, V^\mathsf{T} Y||_F^2 + ||\, Y^\mathsf{T} Y||_F^2 \tag{3.5}$$

where $V^\mathsf{T}V$ is a $d \times d$, $V^\mathsf{T}Y$ a $d \times k$ and $Y^\mathsf{T}Y$ a $k \times k$ matrix [27, p. 32]. A proof for this transformation can be found in Appendix A.2. It's derivative is given by:

$$\frac{\partial J}{\partial V^\mathsf{T}} = 4V(V^\mathsf{T}V) - 4Y(Y^\mathsf{T}V). \tag{3.6}$$

In conclusion, the neural network is able to generate an embedding for each time-frequency bin, which has the property that other embeddings originating from the same speaker have a small Euclidean distance. This fact enables us to create clusters from these embeddings, which will be exploited in Section 3.4.

3.3.2 Frequency Mask as Training Target

When training a neural network in a supervised manner it is essential to define a proper training target. In the last section a training target was introduced which maps each time-frequency bin to a vector in embedding space. We will refer to this training target as being embedding-based.
There are two other categories of training targets, namely mapping-based and masking-based targets [69]. In the first case the spectral time-frequency representation of the separated sources is directly used. A masking-based target in contrast, calculates the belongings of each time-frequency bin to a specific estimated source. The Definition 3.3.1 will be used in this thesis whenever we speak about a mask.

Definition 3.3.1. Ideal Binary Mask The Ideal Binary Mask defines which time-frequency bin belongs to which source in a mixture. For the power spectra X_{pwr} and Y_{pwr} the mask is defined as [71, p. 187]:

$$\text{IBM}(t,m) = \begin{cases} 1 & \text{if } X_{pwr}(t,m) > Y_{pwr}(t,m) \\ 0 & \text{otherwise} \end{cases}.$$

The IBM is 1 for each time-frequency bin in which the power in the spectrum X dominates the power in Y. One might also say that the IBM is 1 for each time-frequency bin where the SNR defined as $\frac{X_{pwr}}{Y_{pwr}}$ is above the 0dB threshold.
Amongst other masks Wang and Chen [69, p. 5] also introduce the ideal ratio mask and its complex version as a training target. They can be seen as a soft version of the IBM. The complex mask allows the neural network to estimate the phases as well [74]. This motivation arises from an evaluation which claims that the phase has an impact on the quality of speech [47]. Nonetheless, the IBM is seen as sufficient for the needs of DC [27].
We pointed out two other training targets shortly in this section in order to introduce the IBM. The target of the next section is to create a similar mask using a clustering algorithm which is able to separate more than two sources.

3.4 Clustering in the Embedding Space

As already described in Figure 3.2 the Deep Clustering method has multiple stages. In the last two sections the pre-processing and neural network model were covered. These are responsible for the term *Deep*. We explore now the second major part which is the *Clustering*.
The k-means algorithm which was introduced in Section 2.3 is used to perform the clustering. The properties of the embedding vectors in V have been discussed in the last section. The clustering of these vectors will lead to a result which determines which time frequency bin belongs to which speaker. In this section the process of k-means clustering is formalized in a way that allows a formal definition using matrices..

Hershey, Chen, Roux, and Watanabe [27] introduce the clustering as a matrix factorization problem [5]. The objective function of Equation 2.28 can be reshaped as:

$$\gamma = \sum_{i=0}^{p} \sum_{j=0}^{k} z_{ij} ||v_i - c_j||^2, \tag{3.7}$$

where $v_i \in \mathbb{R}^d$ is an embedding vector of V, $c_j \in \mathbb{R}^d$ is a centroid vector of the $k \times d$ matrix M and

$$z_{ij} = \begin{cases} 1 & \text{if } x_i \in \mathcal{K}_j \\ 0 & \text{otherwise} \end{cases}$$

is a binary entry of the $p \times k$ matrix Z. The Equation 3.7 is in fact equal to Equation 2.28 because the cluster $\mathcal{K}_j$ is defined to be a set of data points which are closest to the center c_j, where $0 \leq j < k$ is the index of the cluster. Equation 3.7 can be expressed using the Frobenius norm:

$$\gamma = || V - ZM||_F^2, \tag{3.8}$$

where $M = (Y^\mathsf{T} Y)^{-1} Y^\mathsf{T} V$ [5]. The objective remains to minimize γ. The notation using matrices allows a more compact way of defining the problem. Furthermore, it allows to calculate the success of the clustering after applying the k-means algorithm. The squared error between the estimated frequency cluster assignments Z and the true labels Y, which both have the shape $p \times k$, can be defined in the following way [27, p. 32]:

$$d(Y, Z) = || Y(Y^\mathsf{T} Y)^{-1} Y^\mathsf{T} - Z(Z^\mathsf{T} Z)^{-1} Z^\mathsf{T}||_F^2. \tag{3.9}$$

The process from estimating the embedding vectors to clustering the vectors using k-means is determined by the several objectives. The training objective $J(Y, V)$, the k-means objective γ and the clustering error $d(Y, Z)$ are small, if $VV^\mathsf{T} \approx YY^\mathsf{T}$ which leads to $Z \approx Y$[27, p. 33].
In the introduction of this thesis the challenge of separating an arbitrary count of different voices was mentioned (see Section 1.1). As shown in experiments the DC approach is able to generalize to more than one voice, even when trained on only two [27, p. 34]. The proposed structure allows this by using a parameter $k > 2$ when clustering with k-means, without changing the loss function J.
The dimension of the matrix $Z \in \mathbb{R}^{p \times k}$ reminds of the already mentioned concept of a binary mask. And indeed, the matrix is nothing else than a binary mask which is defined as:

$$\text{IBM}(t, m, j) = z_{t(\frac{M}{2}+1)+m,j}, \tag{3.10}$$

where i corresponds to the speaker $0 \leq j < k$. In this thesis we will focus on the separation of two speakers such that $i \in \{0, 1\}$. This mask will be used in Section 3.5 to construct the estimated sources of the mixture.
Lastly, there are two different ways of doing the clustering. The way it was described so far is called local clustering because the output V is clustered directly. The other way is called global clustering. The outputs of the neural network are concatenated and stored in order to do the clustering, once the end of the input signal x has been reached. Experimental evaluations show that the local clustering which is used in this thesis performs slightly better [27, p. 34].
The next section covers the final stage which deals with the reconstruction of estimated source signals in the time-domain.

3.5 Reconstructing the Waveform

The previous sections explained in detail how to compute a binary mask staring from the time domain representation of the discrete signal x. This process added another dimension to the STFT.

Apart from the two dimensions for time and frequency a third one was added which denotes the speaker belonging.
Finally it is possible to create the estimated output signal $\tilde{x}^i$, where $0 \leq i < k$ is the index of the speaker we wish to isolate. As defined in Section 3.2 the complex STFT of the mixture is $S(t, f)$. In Equation 3.10 the final IBM is defined. The first step is to create a modification $\tilde{S}^i(t, f)$ of the STFT of the mixture for each speaker i:

$$\tilde{S}^i(t, f) = \text{IBM}(t, f, i) \cdot S^i(t, f), \tag{3.11}$$

where $\cdot$ multiplies the real part of $S^i(t, f) \in \mathbb{C}$ by $\text{IBM}(t, f, i) \in \mathbb{R}$ and leaves the phases untouched. The DC approach only reconstructs the magnitudes for the estimated signals. Therefore, the phases are taken from the mixture. The next step is to inverse the STFT $\tilde{S}^i(t, f)$ using the overlap-add approach. As described in Section 2.1 a padding at the start and end is required such that no information is ignored because of windowing as indicated by Figure 2.1b. Because of the interaction between the Hann window function, overlapping and padding in the overlap-add method nearly perfect reconstruction is possible. The inverse STFT creates a time discrete signal $\tilde{x}^i$ for each speaker i.
We finished now the fifth stage of the DC approach and acquired estimated source signals which are close to the reference signals. The next chapter describes how estimated sources can be compared to the references in order to measure performance when training the network in different ways.

Chapter 4

Experiments

Chapter 3 covered how the separation works given that the neural network performs well. In this chapter a visualization of the embedding space is shown in order to illustrate the DC approach. Furthermore, the amount of concurrently speaking persons is determined experimentally in Section 4.2. Lastly, the training of the network is discussed in order to achieve good results. The training of a neural network adjusts a great amount of parameters. Nonetheless, hyper-parameters exist which require manual adjustment. The assembly of the data set also determines the performance of the separation approach and requires discussion.

4.1 Visualization of Vectors in the Embedding Space

As defined in Section 2.3, the output of the k-means algorithm is a set of k clusters $\mathcal{K} = \{\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_{k-1}\}$. The vectors in the clusters are d-dimensional. As in most cases the dimension of the embeddings is greater than 10 [27, p. 34], it is impracticable to visualize the clustering directly.
This "curse of dimensionality" [32] led to algorithms which reduce the dimensionality of the data. There are mainly two ways of performing the reduction. Firstly, there is feature selection which selects the "best" subset of the d dimensions. The best subset depends on the task and data. On the contrary, feature extraction transforms the original vector to create new feature vector with a lower dimensionality. [49, p. 103]
We will focus on the visualization using PCA which transforms the vectors in the rows of V to vectors of dimension L. PCA is restricted in a sense such that the computed features are a linear combination of its basis vectors. Therefore, PCA basically changes the basis of the input data. [60, p. 3]
Apart from being pretty the visualization can help to identify implementation errors. For example if the center of the cluster for "Alexis" in Figure 4.1 is shown on the right instead of the left for some time steps, then the cluster labels are confused.
In essence such a visualization gives a good intuition what the output of the neural network means. It can also identify a malfunctioning network if the PCA fails and no clusters are identifiable.

4.2 Clustering Order Selection

When dealing with long recordings of multiple speakers talking at the same time, it is likely that speakers appear and disappear. In this section we try to estimate at which time there are two speakers or only a single speaker.
The resolution of the estimation is one frame which corresponds to 100 overlapping windows. For each approximately 0.8 s frame, k-means clustering is performed with different values for k. The example audio is the concatenation of several utterances. In the first part only one person is

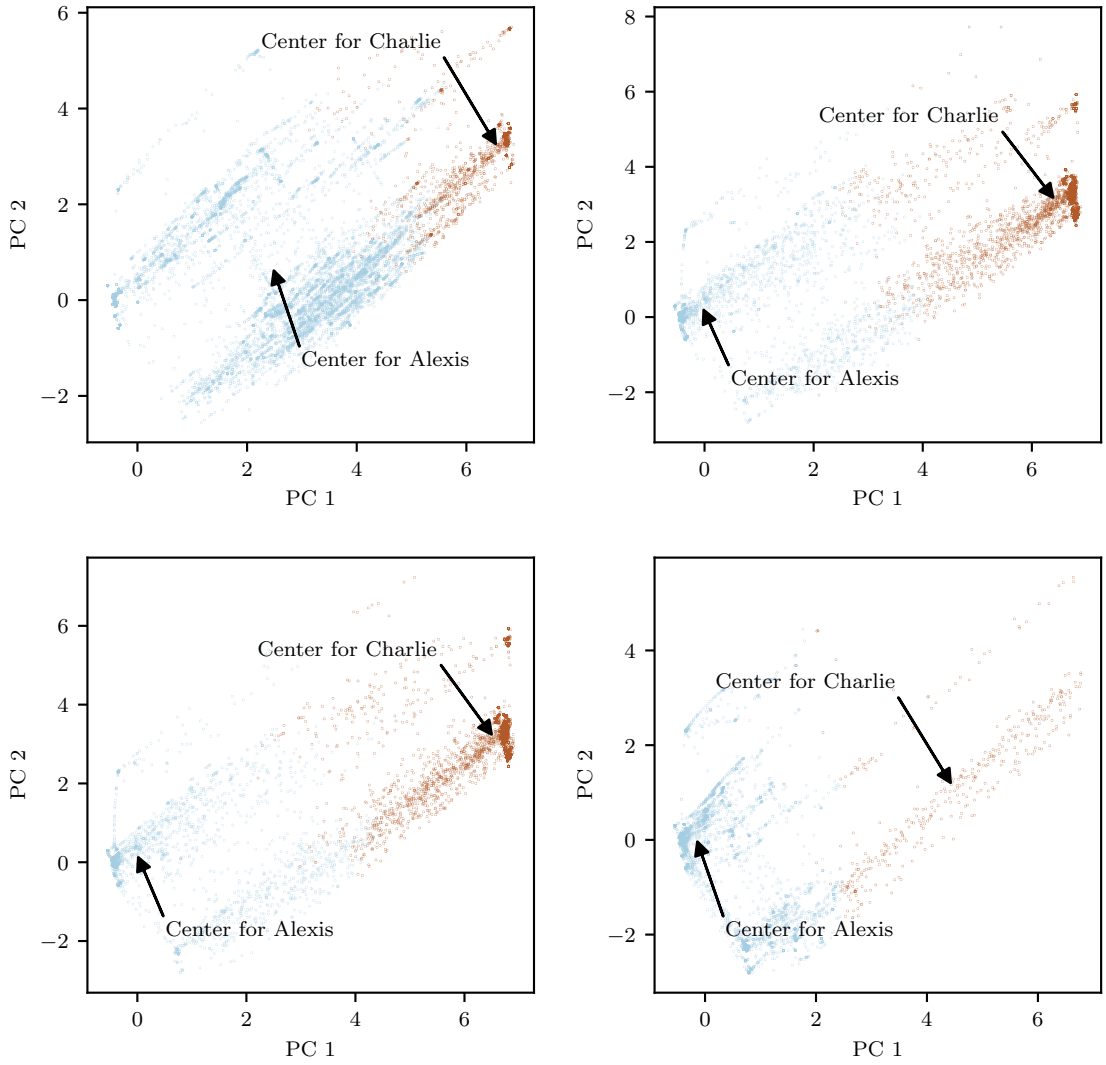


Figure 4.1: PCA of four frames with each $\#_w(\frac{M}{2}+1)$ data points using two components. Each point represents a single embedding vector. The arrows point to the center of mass of the two clusters. Each center correspond to the voice of a specific speaker like Alexis or Charlie. It is clearly visible that the loss function tries to pull certain embeddings closer together. According to Equation 3.4 vectors which are close together belong to the same speaker. The frames are selected from a signal which is longer than four frames. The first and the last frame stand out as Alexis and Charlie just started talking or already stopped.

speaking, in the second two, in the third again one and in the last two. This signal is investigated using a metric of distance and clustering score.
In Figure 4.2 the fact that the loss objective (see Section 3.3.1) pushes embeddings apart which belong to different speakers is exploited. For $k = 1$ the plot shows the Euclidean distance from the origin to the center of mass of the embeddings. For $k = 2$ the red plot shows the Euclidean distance between the two centers. It is possible to observe that the distance between the centers is low if only a single person is talking. As soon as a speaker appears the centers are pushed apart. Furthermore, when clustering with $k = 1$ the center is further apart from the origin if the k matches the count of speakers.

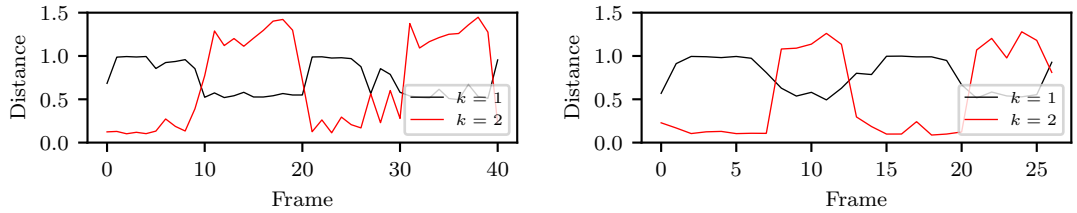


Figure 4.2: The plots show the Euclidean distance between the origin and the center for $k = 1$, as well as the distance between the two clusters for $k = 2$. In each audio signal the first segment contains one voice, the second two, the third again one and the last two.

Another way to look at the clustering is by using the objective of k-means which tries to minimize the Within Cluster Error (WCE). Figure 4.3 shows the error for each frame and for $k = 1$ and $k = 2$. In the second and fourth segment in which two voices are present the error for $k = 2$ is lower than that compared to $k = 1$. This means using k-means clustering with $k = 2$ yields a lower error and therefore is the correct choice. The observations hold true for various speakers and voices.

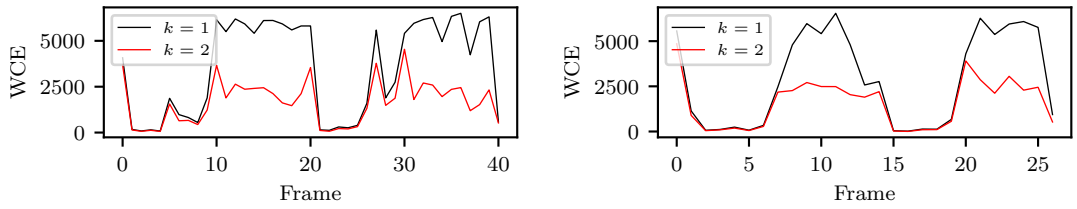


Figure 4.3: The WCE of two audio signals consisting of four segments over time. The signals consists of four segments like described in Figure 4.2.

Unfortunately this the two proposed methods only showed success for differentiating between a single and multiple speakers. As soon as it is attempted to distinguish between two and three speakers the proposed methods fail. The average distance between the centers for $k = 3$ is very similar to the distance between the centers for $k = 2$. Furthermore, the WCE is also very similar for $k = 2$ and $k = 3$.
This section showed another way of visualizing the results of the clustering by comparing the results over time for different orders. In summary we can be confident that the algorithm achieves its goal of separating voices. In order to prove this experimentally, the next step is to establish a performance criterion to compare the approaches and data sets.

4.3 Evaluation metrics

When comparing earlier approaches like those mentioned in Section 1.2 or new methods with each other an objective metric for the success of the separation is required. Even though subjective studies like in Figure 1.1 can evaluate the intelligibility of the separated speech, they are expensive to conduct. Furthermore, a subjective evaluation does not offer good comparability. Therefore, in 2006 Vincent, Gribonval, and Fevotte introduced a method for evaluating the performance of speech separation. This measurement is further adopted in [67] and [66]. The metric is actually defined for audio with more than one channel like stereo recordings. Because of that the term spatial image is introduced. In this case each channel

$$x_i(t) = \sum_{j=0}^{k} s_{ij}^{img}(t), \tag{4.1}$$

yields multiple spatial images which denote the reference sources [67, p. 3]: In this Equation $x_i(t)$ is the signal of the mixture for channel i, k the amount of sources and s_{ij}^{img} the spatial images. As we separate monaural audio this can be simplified to:

$$x(t) = \sum_{j=0}^{k} s_{j}^{img}(t), \tag{4.2}$$

where $x(t)$ is the mixture and $s_j^{img}(t)$ the signal for the reference source of speaker j. It is noteworthy, that compared to earlier performance measurement methods the proposed method offers multiple criteria, namely the Source to Interference Ratio (SIR), Source to Artifact Ratio (SAR), Image to Spatial Distortion Ratio (ISR) and the Source to Distortion Ratio (SDR) which combines the others.
In order to apply the metric the reference sources and the estimated sources are required. The metric decomposes the estimated source for speaker j as:

$$\hat{s}_j^{img}(t) = s_j^{img}(t) + e_j^{spat}(t) + e_j^{interf}(t) + e_j^{artif}(t), \tag{4.3}$$

where the error terms $e_j^{spat}(t)$, $e_j^{interf}(t)$ and $e_j^{artif}(t)$ represent the spatial distortion, interference and artifacts, $s_j^{img}(t)$ the reference source as introduced in Equation 4.2 and $\hat{s}_j^{img}(t)$ the estimated source [66]. The final metric in terms of ratios is defined as:

$$\mathrm{ISR}_j = 10\log_{10}\frac{\sum_t s_j^{img}(t)^2}{\sum_t e_j^{spat}(t)^2}, \tag{4.4}$$

$$\mathrm{SIR}_j = 10\log_{10}\frac{\sum_t (s_j^{img}(t) + e_j^{spat}(t))^2}{\sum_t e_j^{interf}(t)^2}, \tag{4.5}$$

and:

$$\mathrm{SAR}_j = 10\log_{10}\frac{\sum_t (s_j^{img}(t) + e_j^{spat}(t) + e_j^{interf}(t))^2}{\sum_t e_j^{artif}(t)^2}. \tag{4.6}$$

The combined metric is defined as:

$$\mathrm{SDR}_j = 10\log_{10}\frac{\sum_t s_j^{img}(t)^2}{\sum_t (e_j^{spat}(t) + e_j^{interf}(t) + e_j^{artif}(t))^2}. \tag{4.7}$$

In most cases it is not possible to know which estimated source corresponds to which reference source as their order is arbitrary due to the label permutation problem. Therefore the estimated

sources have to be matched to their reference. This is done by testing the each permutation and taking the result which yields the lowest ISR [66, p. 5].
Vincent, Sawada, Bofill, Makino, and Rosca [67, p. 5] note that depending on the application the error should be weighted differently. For example in hearing aids some interference is acceptable as long as the SAR is low. For other applications the interference error could be the most important. Therefore the measurement is arbitrary in a sense that it treats all errors equally.
Although other metrics like PESQ [54], PEASS [12] and STOI [63] exist, the proposed metric is used throughout this chapter in order to evaluate the performance of the DC method. The reason for this is that the implementation in the `BSS Eval` toolkit [15] is used in a lot of papers and therefore offers comparability. The next sections cover the data and which knobs can be adjusted in order to perform experiments.

4.4 Data Sets: TIMIT, WSJ0 and TEDLIUM

The choice of the data set and the way how the features are prepared matters a lot in data-driven applications like DC. The latter is also known as feature engineering. The training and also evaluation of the approach requires on the one hand a mixtures and on the other hand clean reference signals which yield the mixed signal. This is achieved by using raw speech data sets which consist of recordings of single persons talking and mixing them together.
This advantage is unique to the separation of speech. The separation of lead and companion in music raises a greater issue because of the unavailability of reference data. Another noteworthy fact is that the recordings of the individual tracks as well as the mixture undergo post-processing which makes it difficult to create data sets for that use case. [53, p. 14]
In this section three raw data sets are presented which are used to train, adjust and evaluate the RNN. Whereas TIMIT [17] and WJS0 [16] are not freely available the last one TEDLIUM3 [25] is. After that the preparation and mixing of the raw data sets is described in Section 4.5.
The TIMIT data set consists of utterances spoken by 630 English-speaking individuals with diverse dialects of which 30% are female and 70% male. Ten unique sentences are recorded from each speaker which yields 6300 utterances. This data set was recorded under laboratory conditions and therefore offers clean speech signals. [17, Documentation]
The summed duration of the utterances is approximately 5.5 hours which makes this a rather small but diverse set because of the great amount of speakers.
The WSJ0 data set is distributed on several disks. The first 12 compact disks contain training and development material and the last three test data and documentation. Furthermore, the data is subdivided into sets with specific characteristics. For training and adjusting the network the set `SI_TR_S` (Speaker-Independent Training Spontaneous Dictation) is used, whereas for evaluation the sets `SI_ET_05` (Speaker-Independent Evaluation 5K Vocabulary) and `SI_DT_05` (Speaker-Independent Development 5K Vocabulary) are used. This distribution is the same as in [27]. The recording under laboratory conditions was performed using different kinds of microphones. For the training mostly a close-talking microphone is utilized while for the other sets primarily microphones of varying types are used. The evaluation data and development set is spoken by 11 men and 9 females. The training set contains utterances from 64 men and 65 women. All speakers are English [16, Documentation and speaker information]
The sets used for training and adjustment offer 25 h of audio and the set for evaluation approximately 5 h.
The TEDLIUM data set is a free[1] audio collection of 2351 TED talks. The total duration of the audio is 452 h of which 69.9% is from males and 29.6% from females. [25]
Unfortunately, no official gender data for individual talks is available for the data set. Furthermore, as the audio is from public talks, background noise from the audience or audio-visual presentations is expected.

[1] Licensed under CC BY-NC-ND 3.0

id	data set	source	duration in h		reference
			T	E	
A	TIMIT-mix	TIMIT	30	5	[17]
B	WSJ0-mix	WSJ0	30	5	[16]
C	TEDLIUM-mix	TEDLIUM	30	5	[25]

Table 4.1: Overview of the data sets used for training. The duration column specifies the amount of training and evaluation data respectively.

All three data sets only offer monaural audio with a sampling rate of 16 kHz which is enough for this application because the separation only uses a single channel and the audio is resampled to a rate of 8 kHz (see 4.5). The waveforms of the data sets are compressed and packed using the NIST Speech Header Resources (SPHERE) toolkit. Several tools exist in order to convert them to Resource Interchange File Format (RIFF) waveform audio files in order to allow easy and universal playback and processing. The file header of the SPHERE files offer benefits like a specification of the microphone, recording location or recording session which are not required in the use case of audio separation.

4.5 Preparing the Data Sets

The creation of the data set is one of the main tasks in data-driven applications. In this section the process of creating the data sets TIMIT-mix, WSJ0-mix and TEDLIUM-mix like specified in Table 4.1 is presented. Every data set is split into two subsets for specific applications. The training set is used to adjust the weights of the neural network. In contrast, the evaluation set, which is also called the test set, is used to measure the performance of the network and its final accuracy. The training and evaluation using different data sets gives insight whether the hyper-parameterized algorithm adapts to different data.
Depending on the format of the raw data sets specific tasks are needed to create the mixture sets. The first step is to resample and convert the SPHERE files using FFMPEG version 4.1.3 to the rate already mentioned in the last section.
Thereafter, the sets are split into subsets randomly. For TIMIT the whole set is divided into a training and evaluation set which hold 90% and 10% of the data respectively. For TEDLIUM only the upper 20% of the longest talks are used. Those 236 talks are split like the TIMIT set. For WSJ0 `SI_TR_S` is used for training whereas `SI_ET_05` and `SI_DR_05` are used for evaluation. Within each split speakers are selected such that half of them are female in order to have a diversity in speech. In the case of TEDLIUM, which has no gender data available, speakers are selected randomly and assigned to a split until the percentages of the split match with the constraint that half of the speakers are female.
The 236 talks of the test and evaluation set from TEDLIUM have a mean duration of approximately 22 min. Therefore, it is required to create short audio clips from the long TED talks. This is done by randomly selecting 200 5 s audio clips for each speaker from the training set. For the evaluation set only 100 utterances per speaker are created. The first and last 20 s are excluded because they usually contain noise from the audience. The other two data sets already contain short utterances and therefore no action is required.
To summarize, we have now three data sets which contain a subset for training and evaluation. Each subset contains utterances with a fair gender distribution. For each training set the mean μ and standard deviation σ is determined in order to calculate the standard score like defined in Equation 3.2. This is used in Section 3.2 to scale the input features of the neural network. The

last step it to create mixtures from the utterances as well as a matrix Y for each mixture which are required during the training of DC (see Section 3.3.1).
Finally, to create the data sets TIMIT-mix, WSJ0-mix and TEDLIUM-mix, mixtures are generated randomly. This means a random utterance from a random speaker is selected and mixed with a random utterance from a different speaker. The mixing works by adding two utterances $y(n)$ and $z(n)$ up using a random SNR between -3 and 3 dB. In order to create the masks for the training and evaluation set the mixture $x(n) = g(n) + h(n)$ is transformed to the frequency domain using the STFT. As already noted in Section 3.4, Y describes which time-frequency bin belongs to which speaker. This can be calculated by comparing the spectra $G(t, m)$ and $H(t, m)$ of the signals $g(n)$ and $h(n)$ respectively. The element y_{ij} of the matrix Y is equal to one if the i-th time-frequency bin belongs to the signal $g(n)$. Otherwise it is zero and the bin belongs to the signal $h(n)$. This mask can be used directly in the loss function of Equation 3.5.
In conclusion, we have created the three data sets, namely TIMIT-mix, WSJ0-mix and TEDLIUM-mix which consist of small utterances. For the training and evaluation set each utterance has a binary mask Y which is able to separate the mixture in two speech signals of different speakers.

4.6 Training the Model and Adjusting Hyper-Parameters

The training of the network works by learning how to minimize the loss described in Section 3.3.1. There already have been attempts in tuning the hyper-parameters of the original DC approach [31]. This knowledge is used in order to decide what values of the hyper-parameters should be tested. The selection of hyper-parameters is based on guidelines. Therefore, the parameters have to be determined experimentally because a lot of choices are based on intuitions [20]. Table 4.2 shows the hyper-parameters discussed in this section and gives an initial guess for good parameters based on the original paper [27]. The optimization ranges are based on [31] and [20].

Parameter	Symbol	Initial Guess	Optimization Range
learning rate	η	10^{-5}	$(10^{-n})_{n\in\{3,5,7\}}$
train data	-	WSJ0-mix	A, B, C, D (see Table 4.1)
hidden units	n_{hidden}	600	50, 300, 600, 900
hidden layers	$n_{recurrent}$	4	2, 4
batch size	b_s	128	16, 32, 64, 128
embedding dimension	d	40	5, 35, 40, 45
dropout (rc ff)	-	(0.5 0.2)	(0 0), (0.5 0.2)

Table 4.2: Overview of hyper-parameters which can be adjusted for training. Initial guesses are based on [27] and ranges on [31] and [20]. The optimization range is the subject for optimization of the method. The dropout is given as a tuple where the first parameter is the recurrent dropout and the second the feedforward dropout.

The baseline for the evaluation uses the hyper-parameters mentioned in the original paper [27] with the change of using four layers instead of two like proposed in [31]. As in this thesis does not present a fundamental new method, it is adequate to use this as baseline instead of ICA or NMF. By tuning the hyper-parameters a we expect to improve the performance of the baseline model shown in Table 4.3. First of all the time it took to train the model on the hardware described in Section 4.7 is presented. Furthermore, the table shows the mean training and evaluation loss

over the whole data set respectively. When training a network the training data set is processed over and over. Each iteration is called an epoch. This means that the loss values in the tables denote the mean loss over one epoch. In order to measure the performance of the network the metric specified in 4.3 is used. The SDRs show the performance when the model is evaluated on different data sets. For example the model 1a is trained on the WSJ0-mix training data set and also performs best on it. The worst performance of the network is given by evaluating on the TEDLIUM-mix evaluation data set.
Each model has been trained for 40000 steps which means that the network has seen the same amount of batches. This corresponds to about 39 epochs. It is possible to observe that the training time varies between the proposed configurations. This is due to the different network complexity. Throughout the experiments there are cases where the evaluation loss is below the training loss, like when training on the WSJ0-mix data set in Table 4.4. This can be explained by the used data sets. It is possible that the evaluation data set doesn't offer the same challenge as the training set. Although the distribution of female and male voices are equal it is still possible that the evaluation set contains mixtures which are easier to separate. For example in the WSJ0-mix data set microphones of varying recording quality are used. This is most visible in the loss values and not the final performance measurement by the SDRs. A selection of plots which show the training and validation loss can be viewed in Appendix B.

Metric		baseline
	train time	19:18h
loss	train	1759
	eval	1590
SDR in dB	A (train)	6.03
	A (eval)	2.68
	B (train)	3.99
	B (eval)	4.32
	C (train)	1.49
	C (eval)	1.27

Table 4.3: Evaluation results the baseline model. The initial guesses from Table 4.2 are used as hyper-parameters for the baseline.

Greff, Srivastava, Koutnik, Steunebrink, and Schmidhuber [20] investigated the impact of hyper-parameters on the performance on various problems and data sets including TIMIT. The learning rate and network structure are the most important parameters according to them [20, p. 9].
The experiments of Greff, Srivastava, Koutnik, Steunebrink, and Schmidhuber [20, p. 7 Fig. 4] revealed that the learning rate depends on the data set. As we have the opportunity to train on different data sets, namely TIMIT-mix, WSJ0-mix and TEDLIUM-mix we will investigate the impact of the learning rate in Table 4.4. In contrast to the vanilla gradient descent described in Section 2.2.2, the Adam optimizer [33] is used in this thesis, which uses adaptive learning rates. Its name is derived from adaptive moment estimation. This means the learning rate is not fixed but adapts over time. The range for the learning rates mentioned in Table 4.2 corresponds to possible starting points for the learning rate. Choosing a learning rate can be done by starting with $\eta = 1$ and repeatedly decreasing it by a factor of $\frac{1}{10}$ until the performance starts to degrade [20, p. 7].
The evaluation in Table 4.4 reveals that a lower starting learning rate of $\eta = 10^{-3}$ achieved the best performance. The training on the WSJ0-mix data set achieved the lowest loss when evaluated against WSJ0-mix. This correlation holds true for each training set. When evaluated on the same set as the model is trained on, then the SDR is high. This does not mean that the model is speaker

dependent as described in Section 1.1. The model 1a which was trained on WSJ0-mix set also performs good on the TIMIT-mix data set.

Parameter/Metric		1a	1c	1d	1e	1f	1g	1h	1i
	η	10^{-3}	10^{-7}	10^{-3}	10^{-5}	10^{-7}	10^{-3}	10^{-5}	10^{-7}
	train data	A	A	B	B	B	C	C	C
	train time	19:25h	19:15h	18:12h	18:10h	18:04h	18:10h	18:01h	18:03h
loss	train	**1275**	4280	1384	1985	4452	2520	3301	7410
	eval	**1137**	2682	1389	2033	3108	3059	3669	4800
SDR in dB	A (train)	**8.59**	1.29	7.89	4.57	1.57	4.43	1.96	2.36
	A (eval)	**3.60**	1.43	2.55	2.93	1.08	2.49	1.52	1.74
	B (train)	7.50	0.38	**8.83**	6.12	0.69	3.74	1.19	1.34
	B (eval)	7.34	0.74	**8.89**	5.69	1.29	3.50	1.50	2.03
	C (train)	2.59	0.74	2.30	0.81	1.07	**5.37**	1.63	1.40
	C (eval)	2.66	0.35	2.45	1.17	0.68	**4.67**	1.28	0.48

Table 4.4: Evaluation results for adjusting the learning rate. 1b is left out as it corresponds to the baseline in Table 4.3.

The amount of hidden units influences the performance of the network significantly. According to [20, p. 7] the performance of a LSTM network increases with the number of hidden units. Isik, Roux, Chen, Watanabe, and Hershey [31] showed experimentally that increasing the amount of hidden recurrent LSTM layers increased the performance of DC. Therefore, in Table 4.5 the amount of hidden units and hidden layers are adjusted to determine which network architecture in beneficial for the DC approach. The hyper-parameter configuration 1a shows that a low count of 50 units is not sufficient because the evaluation shows that the separation fails. A higher count of hidden units shows better performance. Although the configuration 2d with 2 hidden layers and 900 hidden units performs better than the other approaches 2a and 2b, the baseline shows better SDRs. In conclusion, the architecture of the baseline is superior to the other experiments shown in Table 4.5.

As promised in Section 3.3 an evaluation using GRU cells was conducted. When training on the WSJ0-mix data set and using a learning rate $\eta = 1e - 03$ only a maximum SDR of 2.84 was achieved when evaluating on the TIMIT-mix data set. Therefore it is safe to conclude that the usage of LSTM cells is beneficial to the DC method.

The batch size b_s determines the number of frames which are processed before the weights of the network are updated. A greater batch size allows parallelization of the algorithm using multiple computing units and thus reducing training time. One goal of the this evaluation is to determine whether adjusting the batch size has an impact on the performance.

Table 4.6 shows experimental results using different batch sizes. Generally a greater batch size affects the performance in a positive way. A even higher batch size of 128 as it is used in the baseline achieves the best results. When using a single computing unit like a GPU, memory is a constraint which limits the batch size.

There are further parameters of the DC approach which affect the final performance of the separation and are unique to the approach.

One such parameter is the dimension of the embeddings the neural network produces. Hershey, Chen, Roux, and Watanabe [27] conclude that the separation fails for a small embeddings with a dimension of 5. They also notice that the best separation is achieved by a dimension $d = 40$. Therefore, we will try to further fine adjust the parameters by evaluating the performance in small

Parameter/Metric		2a	2b	2d
	n_{hidden}	50	300	900
	$n_{recurrent}$	4	4	2
	train data	A	A	A
	train time	13:38h	13:47h	15:52h
loss	train	2295	2014	**1865**
	eval	2073	1749	**1659**
SDR in dB	A (train)	1.60	4.42	**5.79**
	A (eval)	-0.12	1.68	**2.78**
	B (train)	1.35	3.11	**4.28**
	B (eval)	1.35	3.22	**4.48**
	C (train)	0.35	0.67	**1.26**
	C (eval)	0.36	0.77	**1.60**

Table 4.5: Evaluation results for adjusting the amount of hidden units. 2c is left out as it would correspond to the baseline in Table 4.3.

steps close to the recommended dimension of 40 in Table 4.7. Choosing a dimension of 45 improves the performance by 0.19 dB compared to the baseline when evaluated on the WSJ0-mix training set, whereas a dimension of 35 is slightly worse. The evaluations on the other data sets show an improvement for a dimension of 35. As the differences in the SRDs are tiny and depend on the evaluation data set, it is reasonable to conclude that optimization does not offer great performance gains.

A further parameter for the DC approach is the length of the input frame which provides the network with temporal context. There have been experiments which conclude that shorter frames increase the diversity within one batch and thus make the training faster in the beginning. This strategy can be combined with curriculum training which trains first on shorter frames and later on longer. An example of this process would be to pre-train a network with frames of 100 windows before training it with frames of 400. This has been successfully applied in order to increase the separation performance slightly [31]. As this optimization does not offer a great leap in performance [31] and is not further followed in [72], it is not further evaluated.

A common regularization technique is used in order to improve the network. As in [31] dropouts are applied to the feedforward and recurrent layers of the bidirectional LSTM structure. The feedforward dropout probability is set to 0.5, whereas the probability for the recurrent dropout is set to 0.2 in the baseline. In Table 4.8 the usage of dropouts is shown for different data sets. It is not possible to show the improvement of several decibel like demonstrated on the WSJ0 data set in the paper by [31]. The reason for this can be the interaction between hyper-parameters. As the learning rate in Table 4.4 affects the performance greatly, it is possible to assume that the learning rate of 10^-5 is responsible that there is no performance gain when using dropouts.

In conclusion, tuning the hyper-parameters offers an improvement in performance. The learning rate affected SDR notably, whereas the dropout did not. In closed conditions, which means when evaluating on the same set of speakers as the network was trained on, the best performance is achieved. When training and evaluation on a clean data set with little noise, then the separation is speaker independent. This fact can be viewed in Table 4.4. The SDRs of configuration 1a which was trained on WSJ0-mix is comparable to that of configuration 1d which was trained on TIMIT-mix. None of the configurations perform similar well as the configuration 1g when evaluated on the TEDLIUM data set. As the TEDLIUM data set contains noise like described in Section 4.4 it

Parameter/Metric		3a	3b	3d
	b_s	32	64	16
	η	10^{-5}	10^{-5}	10^{-5}
	train data	A	A	A
	train time	12:24h	15:30h	11:52h
loss	train	1964	**1851**	2046
	eval	1794	**1645**	1908
SDR in dB	A (train)	4.70	**5.72**	4.34
	A (eval)	1.73	**2.28**	1.50
	B (train)	3.42	**3.89**	2.54
	B (eval)	3.68	**3.90**	2.77
	C (train)	0.96	**1.24**	0.80
	C (eval)	**1.18**	1.10	0.98

Table 4.6: Evaluation results for adjusting the batch size.

is reasonable to assume that noise in training and evaluation data reduces the performance most. Furthermore, the kind of microphone is already an indicator for the success of the deep clustering method. As seen in Table 4.3 the discrepancy between the evaluation results on the WSJ0-mix training and evaluation data set which use different microphones shows the importance of the recording quality.
Therefore the DC approach overcomes the challenge of speaker dependence but is very sensitive to noise. This represents a challenge for further work on this topic.

4.7 Environment Setup

Reproducibility of separation results is an important aspect of deep learning. Therefore, the essential properties of the environment should be given in this section.
The implementation which is used to create the evaluation uses `Python` 3.7.3 [50]. The `NumPy` [44] library version 1.16.3 was used for fundamental scientific computing operations. The library which is used to implement the neural network is `TensorFlow` [1] 1.13.1, which uses `NVIDIA cuDNN` 7.3.1 and `NVIDIA CUDA` 10.0. As already mentioned in Section 2.1 the `librosa` 0.6.3 [41] library is used for audio and signal processing tasks. In Section 3.4 Lloyd's algorithm is applied to perform the clustering. The implementation of this algorithm can be found in the `scikit-learn` [48] library version 0.20.3. Plots which show data generated by software implemented in context of this thesis were created using `Matplotlib` in version 3.0.3 [30].
For evaluation the 4$^{\text{th}}$ `BSS Eval` toolbox[2] [62] is used (see Section 4.3). This implementation is based on the third and second version implemented in Python, which can be found in the `mir_eval` [52] project. Originally the evaluation toolkit was developed as a `MATLAB` toolbox by Févotte, Gribonval, and Vincent [15].
In order to train the model a NVIDIA GeForce GTX 1080 GPU with 8GB memory was used.
The source code of the implementation of this thesis is publicly available under the name `grog` on `https://github.com/maxammann/speech-separation-thesis`.

[2]The current home of the toolbox is `https://github.com/sigsep/bsseval`

Parameter/Metric		4a	4b
	d	35	45
	train data	A	A
	train time	19:10h	19:30h
loss	train	1780	**1765**
	eval	**1597**	1616
SDR in dB	A (train)	6.01	**6.22**
	A (eval)	**2.74**	2.70
	B (train)	**4.31**	4.25
	B (eval)	**4.62**	4.36
	C (train)	**1.42**	1.39
	C (eval)	**1.54**	1.41

Table 4.7: Evaluation results for adjusting the embedding dimension.

Parameter/Metric		5a	5b	5c
	dropout	(0 0)	(0 0)	(0 0)
	train data	A	B	C
	train time	19:13h	18:02h	17:59h
loss	train	**1909**	2152	3411
	eval	**1722**	2180	3668
SDR in dB	A (train)	**5.59**	4.48	0.77
	A (eval)	2.67	**2.71**	-0.16
	B (train)	4.08	**5.95**	0.24
	B (eval)	4.43	**5.71**	0.51
	C (train)	1.07	1.21	**1.81**
	C (eval)	1.74	**1.94**	1.63

Table 4.8: Evaluation results for adjusting the dropout.

Chapter 5

Conclusion

In this paper, we apply the Deep Clustering algorithm which consists of several stages. After extracting the features of the speech in the frequency-domain a RNN is trained to assign each time-frequency bin a high-dimensional embedding vector. The k-means clustering of these vectors yields a segmented speech mixture spectrum which is used to reconstruct the speech of the individual speakers.
By analyzing the state-of-the-art speech separation method, this thesis has demonstrated its adaptability to various data sets. This research clearly illustrates that the approach is speaker independent, but it also raises the question whether it generalizes to recordings of variable quality. While the performance on the TIMIT-mix and WSJ0-mix data sets is similar, the evaluation on the TEDLIUM-mix data set shows the significance of the recording method. Furthermore, already the choice of the microphone and recording setup influences the performance significantly. Further research is needed in order to model broadband and complex tone noise in data sets and thus make the approach generalize to real-world data and recording scenarios. The sensitivity to noise presents a great challenge.
To better understand the output of the neural network, visualizations of the clustering have been introduced. Whereas the first visualization shows the individual embeddings and cluster centers by applying PCA, the second one shows the behavior of the centers and the clustering error over time using order selection. This offers more insight and a more intuitive understanding of the proposed method and can support the research about more noise-robust methods.
The thesis started by introducing the fundamentals of discrete Fourier transform, neural networks and k-means clustering in order to create an understanding of the used concepts. Furthermore, the stages of Deep Clustering have been explained in detail. Finally, the experiments visualize the results and evaluate the success of separation. Together with the publication of the source code for the implementation, this allows reproducibility and comparability of the results on the freely available TEDLIUM data set.
Apart from the contributions of this thesis, further research is needed to make the results of speech separation more comparable. Undertakings like the Community-Based Signal Separation Evaluation Campaign (SiSEC) or CHiME Speech Separation and Recognition Challenge are desirable as the data and procedures for evaluation are defined. These challenges provide a great advantage in terms of comparability as the algorithms can compete on the same data. Acquiring data sets which offers the same characteristics and difficulty as a previously used one can be difficult.
Another aspect which is left open for future work, is the real-world and real-time application of Deep Clustering. The current network architecture is not yet suitable for hearing aids as it requires a significant amount of computational resources. Simplifying the architecture or pruning unimportant neurons while keeping similar performance is a field of research which deserves attention.
In conclusion the Deep Clustering approach offers state-of-the-art speech separation which creates a pleasant listening experience as long as the constraint of high quality and noiseless audio is met.

Appendices

Appendix A

Equivalent transformations of the Loss Function

A.1 Descriptive form

The following derivations transform the loss objective to a more descriptive form. In Section 3.3 it is noted that the embedding row vectors of V are normalized such that:

$$\sum_{i=0}^{d-1} v_i^2 = v_i^\mathsf{T} v_i = \langle v_i, v_i \rangle = 1, \tag{A.1}$$

where $\langle , \rangle$ denotes the dot product, i is the index of a frequency bin and d the dimension of the embedding. Hence given two normalized vectors $v_i \in \mathbb{R}^d$ and $v_j \in \mathbb{R}^d$, the dot product can be expressed as:

$$\begin{aligned} v_i^\mathsf{T} v_j = \langle v_i, v_j \rangle &= 1 - \frac{1}{2}\langle v_i, v_i \rangle - \frac{1}{2}\langle v_j, v_j \rangle + \langle v_i, v_j \rangle \\ &= 1 - \frac{1}{2}\langle v_i - v_j, v_i - v_j \rangle, \\ &= 1 - \frac{1}{2}|v_i - v_j|^2, \end{aligned} \tag{A.2}$$

where $|.|$ is the Euclidean norm.
The loss function introduced in Section 3.3.1 can be described with the k-means objective:

$$J(Y, V) = ||VV^\mathsf{T} - YY^\mathsf{T}||_F^2 = \sum_{i=0,j=0} ((VV^\mathsf{T})_{ij} - (YY^\mathsf{T})_{ij})^2 = \sum_{i=0,j=0} (v_i^\mathsf{T} v_j - y_i^\mathsf{T} y_j)^2. \tag{A.3}$$

Due to the fact that $y_i^\mathsf{T} y_j$ is 1 if y_i and y_j belong to the same speaker and 0 if speaker i and j are different ones:

$$\begin{aligned} &= \sum_{\substack{i=0,j=0 \\ y_i = y_j}} (v_i^\mathsf{T} v_j - 1)^2 + \sum_{\substack{i=0,j=0 \\ y_i \neq y_j}} (v_i^\mathsf{T} v_j - 0)^2 \\ &= \sum_{\substack{i=0,j=0 \\ y_i = y_j}} \big((v_i^\mathsf{T} v_j)^2 - 2(v_i^\mathsf{T} v_j) + 1\big) + \sum_{\substack{i=0,j=0 \\ y_i \neq y_j}} (v_i^\mathsf{T} v_j)^2 \\ &= \sum_{\substack{i=0,j=0 \\ y_i = y_j}} \big(1 - 2(v_i^\mathsf{T} v_j)\big) + \sum_{i=0,j=0} (v_i^\mathsf{T} v_j)^2. \end{aligned} \tag{A.4}$$

With Equation A.2:

$$\overset{A.2}{=} \sum_{\substack{i=0,j=0 \\ y_i = y_j}} \big(|v_i - v_j|^2 - 1)\big) + \sum_{i=0,j=0} (v_i^\mathsf{T} v_j)^2. \tag{A.5}$$

The descriptive form includes now the term $|v_i - v_j|^2$ which sets the goal of the loss function to pull embeddings of the same speaker closer together.

A.2 Efficient Implementation

As the matrices VV^T and YY^T are expensive to compute, Hershey, Chen, Roux, and Watanabe [27] introduced an efficient implementation of the loss function. In order to show that the proposed transformation is valid, a proof is presented. Firstly, two helping theorems are introduced here of which the first defines how the Frobenius matrix of the low rank matrix $V^\mathsf{T}V$ can be created:

$$\begin{aligned}\sum_{i,j}(v_i^\mathsf{T}v_j)^2 &= \sum_{i,j}\sum_{l} v_{il}v_{jl} = \sum_{i,j}\sum_{l}(V)_{il}(V)_{jl} \\ &= \sum_{i,j}\sum_{l}(V^\mathsf{T})_{li}(V)_{jl} \\ &= \sum_{i,j}\sum_{l}(V^{\mathsf{T}^\mathsf{T}})_{il}(V^\mathsf{T})_{lj} = \sum_{i,j}\sum_{l}(V^\mathsf{T})_{lj}(V)_{il} \\ &= \sum_{i,j}(V^\mathsf{T}V)^2_{ji}.\end{aligned} \tag{A.6}$$

As the indices i and j iterate over the same interval, they can be interchanged:

$$= \sum_{i,j}(V^\mathsf{T}V)^2_{ij} = ||\,V^\mathsf{T}V||^2_F. \tag{A.7}$$

Similarly the second helping equation defines Frobenius matrix of the low rank matrix $V^\mathsf{T}Y$:

$$\begin{aligned}\sum_{i,j}(v_i^\mathsf{T}v_j)(y_i^\mathsf{T}y_j) &= \sum_{ij}\left(\sum_{l} v_{il}v_{jl}\right)\left(\sum_{k} y_{ik}y_{jk}\right) \\ &= \sum_{i,j}\sum_{k,l}(v_{il}v_{jl}y_{ik}y_{jk}). \\ &= \sum_{k,l}\sum_{i,j}(v_{il}v_{jl}y_{ik}y_{jk}) \\ &= \sum_{k,l}\left(\sum_{i,j} v_{il}y_{ik}\right)\left(\sum_{i,j} v_{jl}y_{jk}\right) \\ &= \sum_{k,l}(V^\mathsf{T}Y)^2_{kl} \\ &= ||\,V^\mathsf{T}Y||^2_F.\end{aligned} \tag{A.8}$$

The equations A.7 and A.8 can be used to formulate the efficient implementation:

$$
\begin{aligned}
J(Y, V) &= ||VV^{\mathsf{T}} - YY^{\mathsf{T}}||_F^2 = \sum_{i,j}((VV^{\mathsf{T}})_{ij} - (YY^{\mathsf{T}})_{ij})^2 \\
&= \sum_{i,j}(v_i^{\mathsf{T}} v_j - y_i^{\mathsf{T}} y_j)^2 \\
&= \sum_{i,j}(v_i^{\mathsf{T}} v_j)^2 - 2(v_i^{\mathsf{T}} v_j)(y_i^{\mathsf{T}} y_j) + (y_i^{\mathsf{T}} y_j)^2 \\
&= \sum_{i,j}(v_i^{\mathsf{T}} v_j)^2 - 2\sum_{i,j}(v_i^{\mathsf{T}} v_j)(y_i^{\mathsf{T}} y_j) + \sum_{i,j}(y_i^{\mathsf{T}} y_j)^2. \\
&\overset{(A.7)}{\underset{(A.8)}{=}} ||V^{\mathsf{T}} V||_F^2 - 2||V^{\mathsf{T}} Y||_F^2 + ||Y^{\mathsf{T}} Y||_F^2.
\end{aligned}
\tag{A.9}
$$

Appendix B

Loss Plots of Some Selected Experiments

When training a model it is possible to calculate the loss over time for the training and evaluation data set. The following figures show the plot for the loss over time the some experiments from Section 4.6. The gray area around the lines denote the standard deviation. In summary, it is possible to see that a high learning rate leads to a low loss in a short amount of time. It also seems to avoid local minima as it reaches an overall lower loss.

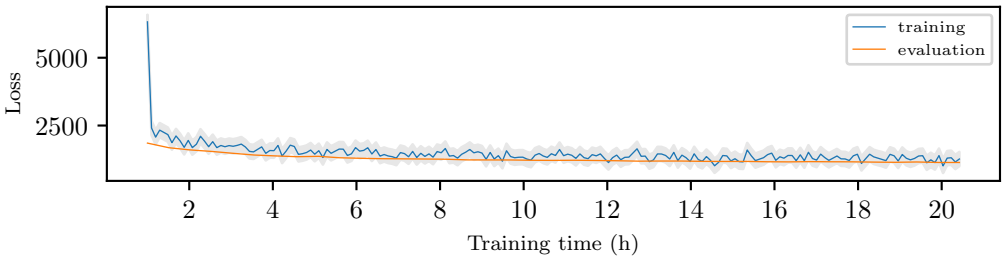


Figure B.1: Loss plot for the experiment 1a from Table 4.4 with learning rate $\eta = 10^{-3}$.

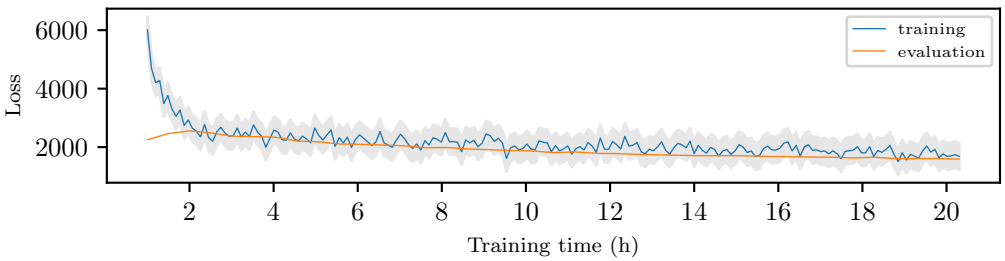


Figure B.2: Loss plot for the experiment 0 from Table 4.3 with learning rate $\eta = 10^{-5}$.

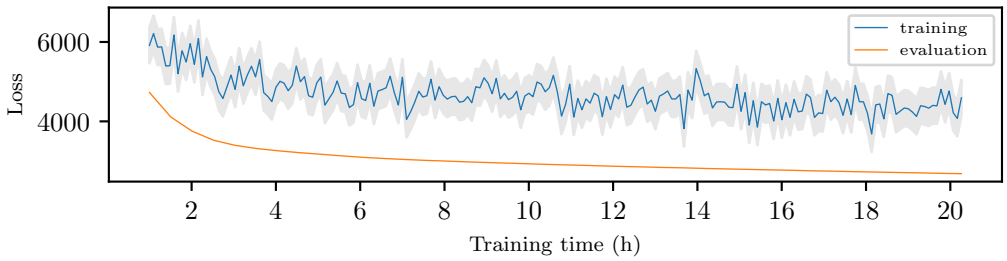


Figure B.3: Loss plot for the experiment 1c from Table 4.4 with learning rate $\eta = 10^{-7}$.

Furthermore, a higher batch size of $b_s = 64$ leads to a smoother graph and lower loss compared to a batch size of $b_s = 16$.

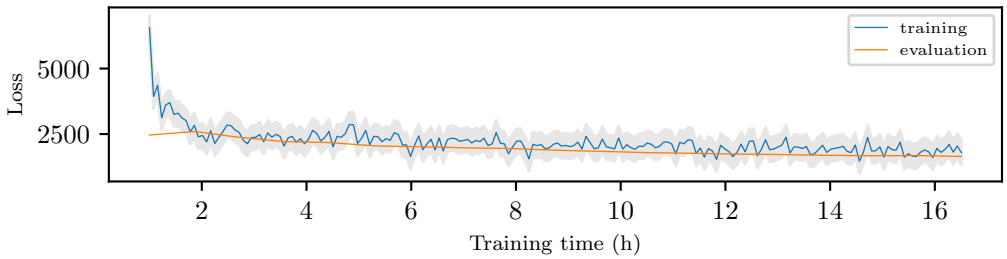


Figure B.4: Loss plot for the experiment 3b from Table 4.6 with batch size $b_s = 64$.

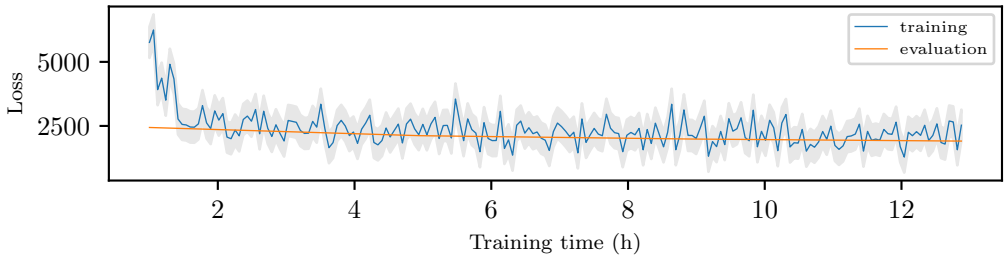


Figure B.5: Loss plot for the experiment 3d from Table 4.6 with batch size $b_s = 16$.

List of Figures

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[2] M. T. Adiga and R. Bhandarkar. Improving single frequency filtering based voice activity detection (VAD) using spectral subtraction based noise cancellation. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. IEEE, Oct. 2016. DOI: 10.1109/scopes.2016.7955823.

[3] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, Jan. 2009. DOI: 10.1007/s10994-009-5103-0.

[4] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, New Orleans, Louisiana. Society for Industrial and Applied Mathematics, 2007. ISBN: 978-0-898716-24-5.

[5] C. Bauckhage. k-Means Clustering Is Matrix Factorization, Dec. 2015. arXiv: 1512.07548 [stat.ML].

[6] P. Berkhin. Survey Of Clustering Data Mining Techniques. Technical report, 2002.

[7] M. A. Casey and A. Westner. Separation of mixed audio sources by independent subspace analysis. In *ICMC*, 2000.

[8] E. C. Cherry. Some experiments on the recognition of speech, with one and with two ears. *The Journal of the Acoustical Society of America*, 25(5):975–979, 1953. DOI: 10.1121/1.1907229.

[9] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, Dec. 2014. arXiv: 1412.3555 [cs.NE].

[10] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–297, May 1965. DOI: 10.1090/s0025-5718-1965-0178586-1.

[11] S. Dubnov. Extracting sound objects by independent subspace analysis. In *Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio*, June 2002.

[12] V. Emiya, E. Vincent, N. Harlander, and V. Hohmann. Subjective and objective quality assessment of audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(7):2046–2057, Sept. 2011. DOI: 10.1109/tasl.2011.2109381.

[13] Y. Ephraim and D. Malah. Speech enhancement using a minimum mean-square error log-spectral amplitude estimator. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(2):443–445, Apr. 1985. DOI: 10.1109/tassp.1985.1164550.

[14] J. M. Festen and R. Plomp. Effects of fluctuating noise and interfering speech on the speech-reception threshold for impaired and normal hearing. *The Journal of the Acoustical Society of America*, 88(4):1725–1736, Oct. 1990. DOI: 10.1121/1.400247.

[15] C. Févotte, R. Gribonval, and E. Vincent. BSS_EVAL Toolbox User Guide – Revision 2.0. Technical Report, 2005, page 19. URL: https://hal.inria.fr/inria-00564760.

[16] J. S. Garofolo, D. Graff, D. Paul, and D. S. Pallett. ARPA CSR-WSJ0 - Continuous Speech Recognition Wall Street Journal. NIST Speech Discs 11-1.1 - 11-12.1, 11-13.1, 11-14.1, and 11-15.1, 1994.

[17] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. DARPA TIMIT - Acoustic-Phonetic Continuous Speech Corpus. NIST Speech Disc 1-1.1, Feb. 1993.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (visited on 07/17/2019).

[19] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012. DOI: 10.1007/978-3-642-24797-2.

[20] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber. LSTM: a search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct. 2017. DOI: 10.1109/tnnls.2016.2582924.

[21] D. Griffin and J. Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, Apr. 1984. DOI: 10.1109/tassp.1984.1164317.

[22] F. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978. DOI: 10.1109/proc.1978.10837.

[23] D. A. Heide and G. S. Kang. Speech enhancement for bandlimited speech. In *IEEE International Conferenceon Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 393–396, May 1998. DOI: 10.1109/ICASSP.1998.674450.

[24] G. Heinzel, A. Ruediger, and R. Schilling. Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new at-top windows. Feb. 2002.

[25] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Estève. Ted-lium 3: twice as much data and corpus repartition for experiments on speaker adaptation. In A. Karpov, O. Jokisch, and R. Potapova, editors, *Speech and Computer*, pages 198–208, Cham. Springer International Publishing, 2018. ISBN: 978-3-319-99579-3. DOI: 10.1007/978-3-319-99579-3_21.

[26] J. R. Hershey, Z. Chen, J. L. Roux, and S. Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation, 2015. arXiv: 1508.04306 [cs.NE].

[27] J. R. Hershey, Z. Chen, J. L. Roux, and S. Watanabe. Deep clustering: discriminative embeddings for segmentation and separation. In *IEEE International Conferenceon Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Mar. 2016. DOI: 10.1109/icassp.2016.7471631.

[28] Y. Hu and P. C. Loizou. Subjective comparison and evaluation of speech enhancement algorithms. *Speech Communication*, 49(7-8):588–601, July 2007. DOI: 10.1016/j.specom.2006.12.006.

[29] P. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis. Deep learning for monaural speech separation. In *IEEE International Conferenceon Acoustics, Speech and Signal Processing (ICASSP)*, pages 1562–1566, May 2014. DOI: 10.1109/ICASSP.2014.6853860.

[30] J. D. Hunter. Matplotlib: a 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. DOI: 10.1109/mcse.2007.55.

[31] Y. Isik, J. L. Roux, Z. Chen, S. Watanabe, and J. R. Hershey. Single-channel multi-speaker separation using deep clustering. In *Interspeech*. ISCA, Sept. 2016. DOI: 10.21437/interspeech.2016-1176.

[32] E. Keogh and A. Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning and Data Mining*, pages 314–315. Springer US, 2017. DOI: 10.1007/978-1-4899-7687-1_192.

[33] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*:arXiv:1412.6980, arXiv:1412.6980, Dec. 2014. arXiv: 1412.6980 [cs.LG].

[34] M. Kolbaek, D. Yu, Z.-H. Tan, and J. Jensen. Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(10):1901–1913, Oct. 2017. DOI: 10.1109/taslp.2017.2726762.

[35] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley, 9th edition, 2006. ISBN: 978-0-471-72897-9.

[36] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, Oct. 1999. DOI: 10.1038/44565.

[37] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, Mar. 1982. DOI: 10.1109/tit.1982.1056489.

[38] R. G. Lyons. *Understanding Digital Signal Processing*. Addison Wesley Pub. Co, 1st edition, 1997. ISBN: 9780201634679.

[39] A. Maas, A. Hannun, and A. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning*, 2013.

[40] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., 1982. ISBN: 0716715678.

[41] B. McFee, M. McVicar, S. Balke, V. Lostanlen, C. Thomé, C. Raffel, D. Lee, Kyungyun Lee, O. Nieto, F. Zalkow, D. Ellis, E. Battenberg, R. Yamamoto, J. Moore, Ziyao Wei, R. Bittner, Keunwoo Choi, Nullmightybofo, P. Friesch, Fabian-Robert Stöter, , Thassilo, M. Vollrath, Siddhartha Kumar Golu, Nehz, S. Waloschek, , Seth, R. Naktinis, D. Repetto, C. Hawthorne, and CJ Carr. Librosa/librosa: 0.6.3, 2019. DOI: 10.5281/zenodo.2564164.

[42] G. A. Miller. The masking of speech. *Psychological Bulletin*, 44(2):105–129, 1947. DOI: 10.1037/h0055960.

[43] M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com/ (visited on 07/17/2019).

[44] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing, 2006.

[45] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Pearson Education, 3rd edition, 2014. ISBN: 1-292-02572-7, 978-1-292-02572-8.

[46] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time Signal Processing*. Prentice-Hall, Inc., 2nd edition, 1999. ISBN: 0-13-754920-2.

[47] K. Paliwal, K. Wójcicki, and B. Shannon. The importance of phase in speech enhancement. *Speech Communication*, 53(4):465–494, Apr. 2011. DOI: 10.1016/j.specom.2010.12.003.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] P. Pudil and J. Novovičová. Novel methods for feature subset selection with respect to problem knowledge. In *Feature Extraction, Construction and Selection*, pages 101–116. Springer, 1998. DOI: 10.1007/978-1-4615-5725-8_7.

[50] Python Software Foundation. The python language reference, version 3.7.3. URL: https://docs.python.org/3/reference/ (visited on 07/24/2019).

[51] Y.-m. Qian, C. Weng, X.-k. Chang, S. Wang, and D. Yu. Past review, current progress, and challenges ahead on the cocktail party problem. *Frontiers of Information Technology & Electronic Engineering*, 19(1):40–63, Jan. 2018. ISSN: 2095-9230. DOI: 10.1631/FITEE.1700814.

[52] C. Raffel, B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, and D. P. W. Ellis. mir_eval: A Transparent Implementation of Common MIR Metrics. *Proceedings of the 15th International Conference on Music Information Retrieval*, 2014.

[53] Z. Rafii, A. Liutkus, F.-R. Stoter, S. I. Mimilakis, D. FitzGerald, and B. Pardo. An overview of lead and accompaniment separation in music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(8):1307–1335, Aug. 2018. DOI: 10.1109/taslp.2018.2825440.

[54] A. Rix, J. Beerends, M. Hollier, and A. Hekstra. Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. In *IEEE International Conferenceon Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2001. DOI: 10.1109/icassp.2001.941023.

[55] S. Rosen, P. Souza, C. Ekelund, and A. A. Majeed. Listening to speech in a background of other talkers: effects of talker number and noise vocoding. *The Journal of the Acoustical Society of America*, 133(4):2431–2443, Apr. 2013. DOI: 10.1121/1.4794379.

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

[57] B. Schuller, F. Weninger, M. Wollmer, Y. Sun, and G. Rigoll. Non-negative matrix factorization as noise-robust feature extractor for speech recognition. In *IEEE International Conferenceon Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Mar. 2010. DOI: 10.1109/icassp.2010.5495567.

[58] M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. DOI: 10.1109/78.650093.

[59] B. Sharpe. Invertibility of overlap-add processing, 2018. URL: https://gauss256.github.io/blog/cola.html (visited on 07/23/2019).

[60] J. Shlens. A tutorial on principal component analysis, 2014. arXiv: 1404.1100 [cs.LG].

[61] J. O. Smith. Overlap-add stft processing. In *Spectral Audio Signal Processing*. 2011. URL: https://ccrma.stanford.edu/~jos/sasp/Overlap_Add_Decomposition.html (visited on 07/16/2019).

[62] F.-R. Stöter, A. Liutkus, and N. Ito. The 2018 Signal Separation Evaluation Campaign, Apr. 2018. arXiv: 1804.06267 [eess.AS].

[63] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen. A short-time objective intelligibility measure for time-frequency weighted noisy speech. In *IEEE International Conferenceon Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2010. DOI: 10.1109/icassp.2010.5495701.

[64] Y. Tu, J. Du, Y. Xu, L. Dai, and C.-H. Lee. Deep neural network based speech separation for robust speech recognition. In *12th IEEE International Conference on Signal Processing (ICSP)*. IEEE, Oct. 2014. DOI: 10.1109/icosp.2014.7015061.

[65] E. Vincent, R. Gribonval, and C. Fevotte. Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech and Language Processing*, 14(4):1462–1469, July 2006. DOI: 10.1109/tsa.2005.858005.

[66] E. Vincent, S. Araki, F. Theis, G. Nolte, P. Bofill, H. Sawada, A. Ozerov, V. Gowreesunker, D. Lutter, and N. Duong. The Signal Separation Evaluation Campaign (2007–2010): Achievements and Remaining Challenges. *Signal Processing*, 92(8):1928–1936, Aug. 2012. DOI: 10.1016/j.sigpro.2011.10.007.

[67] E. Vincent, H. Sawada, P. Bofill, S. Makino, and J. P. Rosca. First Stereo Audio Source Separation Evaluation Campaign: Data, Algorithms and Results. In M. E. Davies, C. J. James, S. A. Abdallah, and M. D. Plumbley, editors, *Independent Component Analysis and Signal Separation*, pages 552–559. Springer, 2007. ISBN: 978-3-540-74494-8.

[68] T. Virtanen. Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria. *IEEE Transactions on Audio, Speech and Language Processing*, 15(3):1066–1074, Mar. 2007. DOI: 10.1109/tasl.2006.885253.

[69] D. Wang and J. Chen. Supervised speech separation based on deep learning: an overview. *IEEE/ACM Transactions on Acoustics, Speech, and Signal Processing*, 26(10):1702–1726, Oct. 2018. DOI: 10.1109/TASLP.2018.2842159.

[70] D. Wang and G. Brown. *Computational Auditory Scene Analysis: Principles, Algorithms, and Applications*. Wiley, 2006. ISBN: 9780471741091.

[71] D. Wang. *On Ideal Binary Mask As the Computational Goal of Auditory Scene Analysis*. In *Speech Separation by Humans and Machines*. P. Divenyi, editor. Springer, 2005, pages 181–197. ISBN: 978-0-387-22794-8. DOI: 10.1007/0-387-22794-6_12.

[72] Z.-Q. Wang, J. L. Roux, and J. R. Hershey. Alternative objective functions for deep clustering. In *IEEE International Conferenceon Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Apr. 2018. DOI: 10.1109/icassp.2018.8462507.

[73] R. J. Williams and D. Zipser. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity, 1995.

[74] D. S. Williamson, Y. Wang, and D. Wang. Complex ratio masking for monaural speech separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(3):483–492, Mar. 2016. DOI: 10.1109/taslp.2015.2512042.