Karlsruhe Institute of Technology

Neural Network Architectures for Reverberated Lecture Speech Recognition

Master Thesis of

Marvin Ritter

at the Department of Informatics
Institute for Anthropomatics and Robotics

Reviewers:	Prof. Dr. Alexander Waibel
	Prof. Dr. Rainer Stiefelhagen
Advisors:	Prof. Dr. Florian Metze
	Markus Müller
	Dr. Sebastian Stüker

Time Period: 22th December 2015 – 6th July 2016

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 1st March 2017

Acknowledgement

I would like to express my gratitude to my supervisor Alex Waibel for the useful comments, ideas and for listening to my problems. This also goes to my advisors Markus Müller and Florian Metze, who helped out with system setups and explanation where needed. Makrus was also the one who told me to try bottle-neck features. The engineering of the Python Interface done in this thesis would not have been possible without Florians simple decoder prototype and Sebastian Stükers support for the idea. Thank you!

I would also like to thank the CLICS team and sponsors for making it possible to write this thesis at the Carnegie Mellon University in Pittsburgh. It was a great place to study and exchange knowledge with many great minds. There I also meet Nils Holzenberger, who I want to thank for the wonderful discussions, even so not all were related to this work.

Finally I would like to thank my loved ones, who have supported me throughout entire process.

Abstract

Speech technology has left the research laboratory and is making its way into end user products. The new applications include voice search, personal assistants, interpreters and control of home entertainment systems. Many of those still require close-talk microphones to achieve good recognition accuracy. A low error rate is crucial for language understanding and the resulting actions on top of it, and hand-held or head mounted microphones create an additional hurdle for the user. Therefore almost all applications of speech recognition would benefit from distant-talking speech capturing, where talkers are able to speak at some distance from the microphone. Due to the distance environmental sounds mix in and disturb the speech signal. One of these disturbances is reverberation.

This thesis shows the effect of reverberation on speech recognition and techniques that can help to avoid low recognition rates. It gives an overview of state of the art acoustic modeling and shows how training data can be transformed to train reverberation robust models. This is cheaper than recording new multi-conditional training data. We describe how room impulse responses can be used to add reverberation to audio signals. The approach it verified for large-vocabulary continuous speech recognition using recordings from lectures. It lowers the word error rate by 30 % for a mix of reverberant environments.

The size of temporal context is investigated and found that increasing it for reverberant speech improves the system performance. Finally the approach was evaluated for three different feed-forward neural networks architectures. This provided only small gains.

For the experiments in this work extensive implementation work was done. A newly created Python module was used to train the acoustic models and run decoding tests. The new pipeline is flexible and allows the use of arbitrary neural network structure. The design of the module is described in this thesis.

Deutsche Zusammenfassung

Die Automatische Spracherkennung hat die Forschungslabore verlassen und ist auf den Weg zum Endnutzer. Neue Anwendungen wie persönliche Assisstenten, Übersetzer and die Kontrolle von Home Entertainment Systemen per Sprache werden unseren Umgang mit Computern für immer verändern. Noch benötigen viele dieser Anwendungen Nahsprechmikrofone um gute Erkennungsraten zu erreichen. Eine geringe Fehlerrate ist entscheidend für das auf der Spracherkennung aufbauende Sprachverständnis und die Reaktion des Systems. Dauerhaft ein Mikrofon in der Hand zu halten oder aufwändig am Körper zu montieren ist jedoch aufwändigt und schafft eine zusätzlich Hürde für den Benutzer. Daher könnten fast alle Anwendungen von automatischer Spracherkennung durch entfernte Mikrofone profitieren. Dabei mischen sich Umgebungsgeräusche in das Signal und stören das Sprachsignal. Eine dieser Störungen ist Nachhall.

Die vorliegende Thesis zeigt den Effekt von Hall auf die Spracherkennung sowie Techniken um die negativen Auswirkungen zu verhindern. Dabei wird mit einer Übersicht über moderne akustische Modelle begonnen. Es wird gezeigt dass verhallte Trainingsdaten die Erkennungsrate verbessern. Neue (verhallte) Daten zu sammeln und zu transkribieren ist aufwändig und teuer. Daher wird beschrieben wie Raumimpulsantworten genutzt werden können um die dem Raum eigenen akustischen Eigenschaften einem Signal hinzuzufügen. Die Wirksamkeit der Methode wird mit Experimenten belegt.

Anschließend wird die Größe des temporalen Kontext untersucht and es stellt sich heraus das eine Vergrößerung die Qualität der Erkennung erhöht. Die Vorgehensweise wird an drei verschieden Sturukturen für neurale Netze getestet. Insgesamt werden Verbesserung um 30 % der Wortfehlerrate auf verhallter Sprache erzielt.

Für die Experimente in dieser Arbeit waren umpfangreiche Neuimplementierungen notwendig. Das neu erstellte Python Modul erlaubt das Trainieren und Evaluieren von Spracherkennern. Python besonders anfängerfreundlich und erlaubt, über zahlreiche Tools, das Trainieren und Nutzen von beliebigen Strukturen für neurale Netze. Grundlegende Aspkete des Moduls werden in der Arbeit beschrieben.

Contents

List of Figures

List of Tables

1. Introduction

Automatic speech recognition (ASR) is the process of converting an acoustic speech signal into its corresponding sequence of words or other linguistic entities. Recently speech recognition has left the research laboratory and is increasingly coming into practical use. The wide range of products ranges from simple dictation over searching the world wide web via voice (e. g. Siri on iPhone, Google Now on Android) to control of the home entertainment system (e. g. Kinect on Xbox, Amazon Echo). On view are personal assistants that support the user by keeping track of things, providing information and undertake little tasks. And while automatic speech recognition (ASR) has advanced considerably during the last decades - state of the art systems are now able to transcribe read newspaper with near human performance - most of the systems today still require a microphone located near the talker. This might not always be possible and forcing hand-held or body-worn equipment is always a burden to usability. Thus these applications would benefit from distant-talking speech capturing.

The challenge of distant-talking speech is that with increasing distance the target speech signal becomes less dominant and sounds from the environment start mixing in. These environmental sounds can be ambient noises (air conditioner, traffic etc.), competing speakers (other conversations), music and reverberation. The later are reflections of the source signal from walls and obstacles that arrive with a delay at the microphone. The recorded signal only contains the sum of the original signal and all reflections. Even without the other distortions of far-field recordings reverberation can cause dramatic increases in the error rate (see table 1.1).

Building speech recognition systems that are robust to reverberation is therefore important for future applications. It will improve the usability and widen the area of use of natural human-machine interfaces. To achieve this goal we will use artificial neural networks. Neural network (NN) for short are a family of models inspired by biological neural networks. They learn from examples, transcribed audio in case of speech recognition, and given large amount of examples achieve the best results today. Unfortunately most of the transcribed recordings are close-talking speech. Without enough examples from various environments neural networks (NNs) will not learn how to recognize reverberant speech. Therefore this thesis focuses a) on creating artificial reverberant training data and b) tune NNs to learn from it.

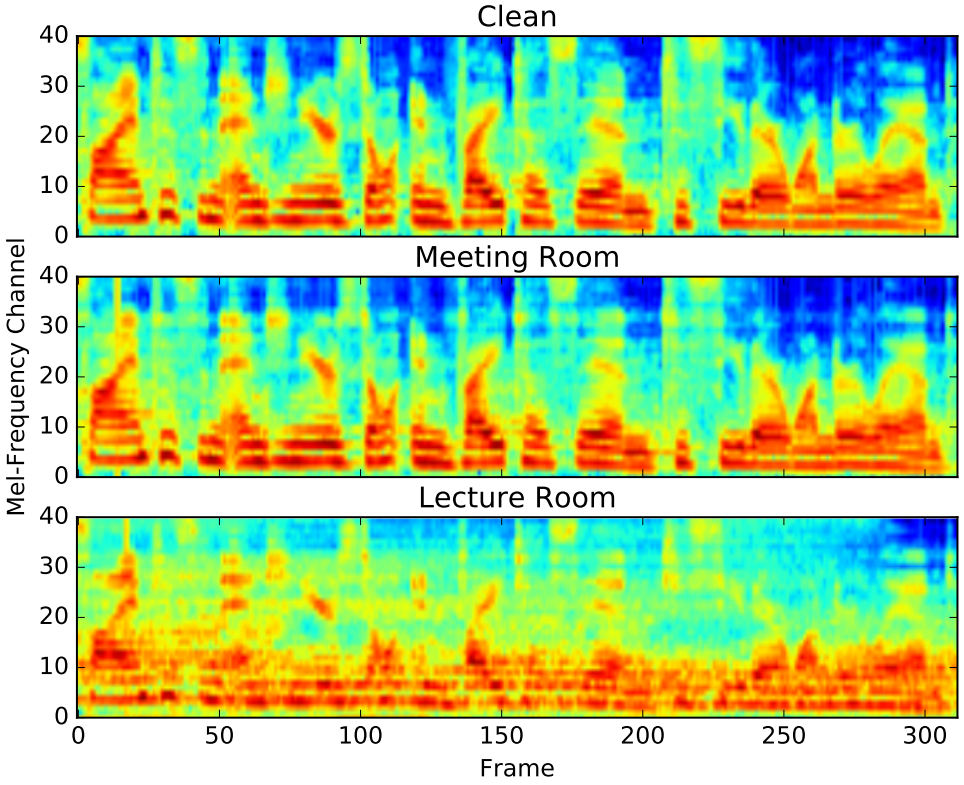

Figure 1.1.: Mel frequency spectrum for the same sentence spoken into a close-talk microphone and two distant microphones a few meters into big rooms. The spectrum and especially the formats get smeared in time.

environment	word error rate
Close-talk	8.3 %
Meeting Room	13.3 %
Lecture Room	27.7 %
Stairway	58.8 %
Auditorium	62.3 %

Table 1.1.: Room size and reflection properties greatly influence of the recognition rate. The signal constructed for the measurement does not include ambient noises that are normal for those environments. Even without these noises the high error rate is to high for productive application.
In very reverberant environments, like big cathedrals, even humans will have difficulties understanding speech, but the rooms above don't cause serious problems for use for humans.

1.1. Contribution

Many existing techniques for dealing with reverberation were developed. Most of the signal processing approaches try to estimate the reflections and remove them from the recorded signal. While this works in case of multiple microphone and non-moving speakers, it requires basic knowledge about the environment. Also there are cases were the original signal cannot be recovered. Newer approaches therefore focus on training systems with reverberated data to learn deep features that are robust to new environments. This has been done for hidden Markov model (HMM) systems [1] and for dereverberating autencoders [2, 3]. It was also used successfully to train a NN on reverberant features directly [4, 5], but mixed with other techniques for various noise types. Building on that we investigate the use of room impulse responses (RIRs) to generate reverberant training data (see figure 1.1 for examples). We compare the use of measured room impulse responses with generated ones and explain the method step by step. We present different neural network architectures, namely a simple deep neural network (DNN), a combination of a DNN and Bottle-neck features (BNF) and the advanced time delayed neural network (TDNN) architecture. Finally we show that a greater temporal context is helpful to deal with reverberant speech.

1.2. Structure of this work

Chapter 2 will provide background knowledge about speech recognition, neural networks and how both can be used together. These are prerequisites for understanding core concepts of recent research on the topic. The chapter will close with a review of room impulse responses. Since the approaches investigated in this thesis are inspired by dereverberating autoencoders and results of the ASpIRE challenge, chapter 3 will introduce the related work. In chapter 4 the comprehensive changes to the speech recognition framework are described. An overview of the newly created Python module is also given. Chapter 5 will start with an overview of our experimental setup and the training data used. Different models are evaluated. Finally, we summarize our work in chapter 6, providing a short discussion of the inferred knowledge and showing research opportunities for future works.

2. Background

This chapters explains the basic concepts of modern ASR. It will first give an overview of systems without neural networks, then introduce artificial neural networks and how they can be used for acoustic models. State-of-the-art speech recognition systems can have many different components (e. g. multiple language models) and use many different machine learning algorithms, we will therefore focus on those relevant for this work.

2.1. Automatic Speech Recognition

Automatic speech recognition (ASR), sometimes also referred to as speech to text (STT), targets the problem of transforming an audio signal containing speech into a textual representation (e.g. sequence of words) with the same meaning. It is a complex task and no single algorithm, in form of a function mapping from speech to text, is known to solve it. This is due to the high variability of speech caused by the interaction of speaker characteristics (e. g. gender, mood), and rate, environment noise, side talks and many more.

Instead of functions, written by humans, signal processing techniques are combined with statistical models to find the most likely transcription. The basic components are shown in figure 2.1. First the audio signal is recoded using a microphone and the analogous signal is sampled, both in time and amplitude, to get a discrete digital signal. The preprocessing will then perform several transformations to normalize the data and extract salient features describing properties of the human speech. A detailed list of steps can be found in [6, chapter 5.2]. The result will be a sequence of real valued vectors X (feature vectors).

Now the decoder will search the space of possible sentences (hypotheses) for the sentence $\hat{W}$ with highest probability given the features X. We can write this as:

$$\hat{W} = \arg\max_{W} P(W|X) \tag{2.1}$$

Using Bayes' rule we can rewrite the equation to [7, p. 135]:

$$\hat{W} = \arg\max_{W} P(W|X) \tag{2.2}$$

$$= \arg\max_{W} \frac{P(X|W)P(W)}{P(X)} \tag{2.3}$$

$$= \arg\max_{W} P(X|W)P(W) \tag{2.4}$$

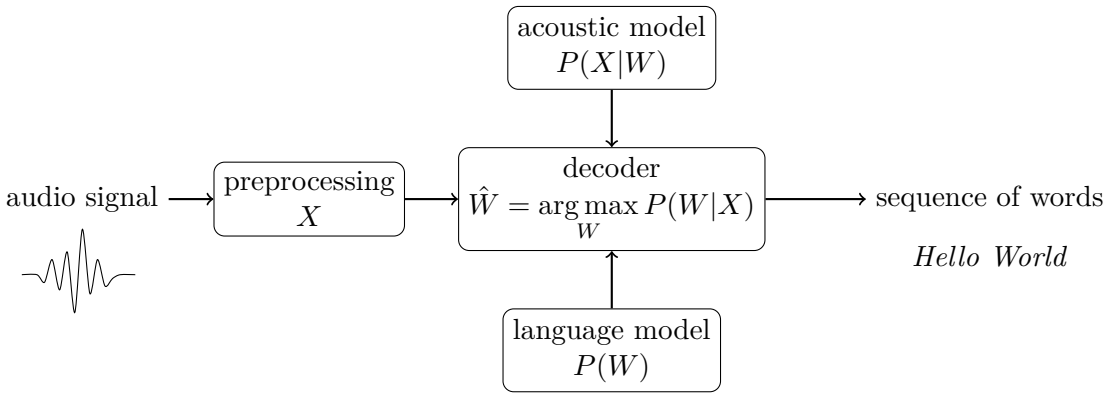

Figure 2.1.: Flowchart of a speech recognition system

This is also known as the fundamental formula of ASR. $P(X|W)$ is the probability of seeing sequence of features vectors X given a hypothesized word sequence W. We call this the acoustic model (AM). It integrates knowledge of acoustics and phonetics. $P(W)$ is a-priori probability of a hypothesized sentence W. Typical learned from large text corpora it provides knowledge of grammar and language style and is therefore called the language model (LM). $P(X)$ is the a-priori probability of seeing features X and not relevant for the search of best hypothesis.

To achieve the goals of this thesis only the acoustic model (AM) (and the training data) was altered and will be described in more detail in the next sections.

2.1.1. Acoustic Model (AM)

The task of the AM is to assign probabilities to sequences of phonemes given the sequence of feature vectors from the preprocessing. Phonemes can vary in time range (e. g. 30 ms to 200 ms for Polish [8]) and frequency depending on the speaker, context and environment. To deal with the variance many systems use hidden Markov model (HMM).

Hidden Markov Model (HMM)

A HMM is a natural extension of the Markov chain [7, p. 380] and adds a non-deterministic process that generates output observation symbols in any given state. The state sequence itself is hidden and cannot be observed directly. Formally a HMM is defined as 5-tuple $\lambda = (S, \pi, A, B, V)$:

- $S = \{q_1, q_2, \ldots, q_n\}$ - A set of states
- $\pi = \{\pi_i\}$ - A initial state distribution, where π_i is equal to the probability of being in state i at time $t = 0$
- $V = \{o_1, o_2, \ldots, o_m\}$ - An alphabet of observable symbols
- $A = \{a_{ij}\}$ - A transition probability matrix, where a_{ij} is the probability of going from state i into state j.
- $B = \{b_i(k)\}$ - An output probability matrix, where $b_i(k)$ ii the probability of emitting symbol o_k while being in state i.

In speech recognition it is common to have one or three states for each phoneme. In the later case, each phone is divided into three parts (begin, middle and end) for a better recognition and time invariance. Given a dictionary with pronunciations we can build a HMM for each word in our vocabulary (see figure 2.2 for an example).

When working with HMM three problems are encountered:

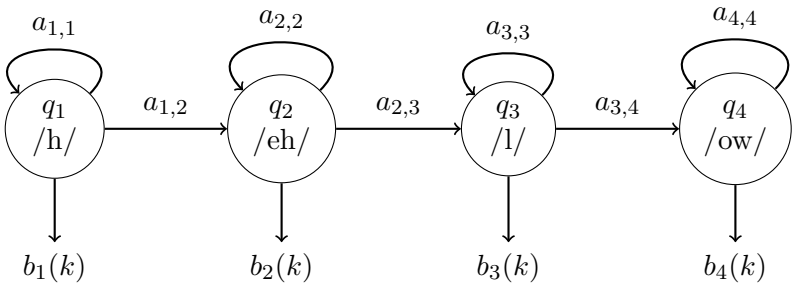

Figure 2.2.: Simple HMM for the word *hello*. In general each state can be connected with any other state, but for the application in speech recognition all backward transition probabilities are set to 0.

1. Evaluation: Given a model $\lambda = (A, B, \pi)$ and the observation sequence $O = o_1, o_2, \ldots, o_T$, how to efficiently compute $P(X|\lambda)$, the probability of the observation sequence, given the model?
2. Decoding: Given a model $\lambda = (A, B, \pi)$ and the observation sequence $O = o_1, o_2, \ldots, o_T$, how to determine the state sequence $Q = q_1, q_2, \ldots, q_T$ that most likely produced our observation sequence?
3. Learning: How to update the model parameters $\{A, B, \pi\}$ to maximize $P(O|\lambda)$

Using dynamic programming the three problems can be solved efficiently. The solution for the evaluation problem is the *Forward-Backward Procedure*. It can be used to recognize isolated words. The decoding problems targets (word) segmentation and continuous speech recognition and done using the *Viterbi Algorithm*. Finally, learning the parameters (i. e. for a subsequent recognition task) can be done using the Baum-Welch-Algorithm, an instance of the expectation-maximization (EM) algorithm.

Gaussian Mixture Model (GMM)

The normal (or Gaussian) distribution is a widely used continuous probability distribution and well suited for fitting noisy data. The formula for a d-dimensional Gaussian distribution $\mathcal{N}$ is given in 2.6. μ refers to the mean vector and $|\Sigma|$ is the determinant of the covariance matrix. An example curve is shown in 2.3.

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \cdot \exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)) \tag{2.5}$$

μ : mean vector

d : number of dimensions

Σ : covariance matrix

Gaussian mixture model (GMM) are weighted sums of Gaussian distributions defined by:

$$p(x) = \sum_{i=1}^{N} w_i \mathcal{N}(x|\mu_i, \Sigma_i) \tag{2.6}$$

$$\sum_{i=1}^{N} w_i = 1 \quad \text{and} \quad 0 \leq w_i \leq 1 \quad \forall i \in 1, \ldots, N$$

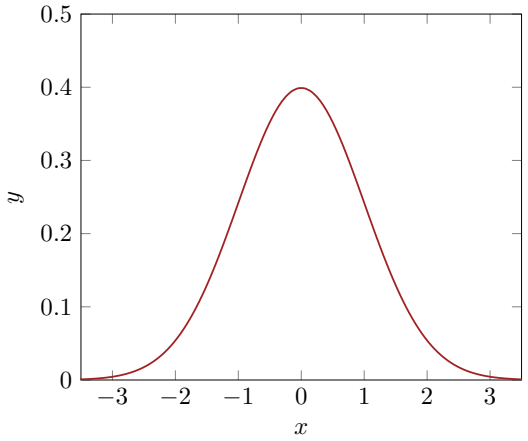

Figure 2.3.: Gaussian distribution with $\mu = 0$ and standard deviation $\sigma = 1$

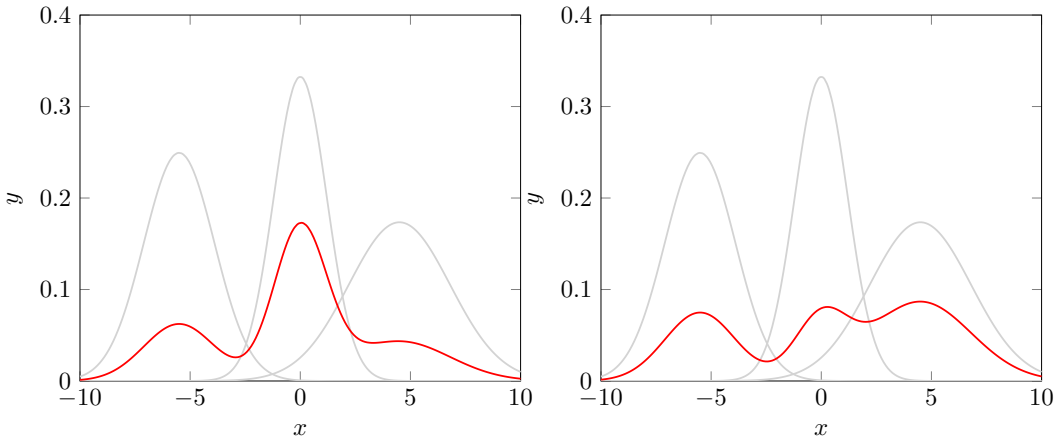

Figure 2.4.: Two different mixtures of 3 Gaussian distributions (weights left: $[0.25, 0.5, 0.25]$, weights right: $[0.3, 0.2, 0.5]$)

Gaussian mixture models (GMMs) can be used to replace the probability matrix B in a HMM to calculate probabilities for states given the continuous vector of features from the preprocessing step. Each state has its own GMM with a fixed number of distributions. And each distribution has its own weight, mean vector and covariance matrix. The ideal number of distributions depend on the amount of training data and might be limited by the computational resources available. In general more distributions allow the model to better fit the structure of the data and become more accurate. For training a GMM-HMM model the expectation-maximization (EM) algorithm can be used [9].

2.1.2. Word Error Rate (WER)

The goal of ASR is to transform speech to text, where the text matches the words spoken. Often both sequences will not be identical and might not even have the same number of words. For a given utterance, the correct sequence is our reference and the output of our system will be our hypothesis. In speech recognition it is common to define the performance of the system as an edit distance between the reference and the output. Thus counting the minimum number of operations is required to transform one into the other. Depending on the task the distance can be calculated on different levels (e. g. words, characters, phonemes). Operations are usually substitutions, deletions and insertions and can have

different penalties, but most common is simply setting all to 1. This is the case for the Levenshtein distance of two sequences a and b, defined in 2.7.

$$\text{levenshtein}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \text{levenshtein}_{a,b}(i-1,j) + 1 \\ \text{levenshtein}_{a,b}(i,j-1) + 1 \\ \text{levenshtein}_{a,b}(i-1,j-1) + 1 - \delta_{a_i,b_j} \end{cases} & \text{otherwise.} \end{cases} \tag{2.7}$$

Here δ_{ij} is the Kronecker delta given by $\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$

Transferring the Levenshtein distance to word level leads to the widely used word error rate (WER):

$$\text{WER} = \frac{\#\text{Substitutions} + \#\text{Deletions} + \#\text{Insertions}}{\#\text{Words in reference}} \tag{2.8}$$

We will use this measure to compare our systems. In case that the hypothesis is longer than the reference it can be greater than 1, otherwise it will be between 0 and 1 (or 0 and 100 % respectively).

2.2. Neural Network

Artificial neural networks, or neural networks (NNs) for short, are a group of models inspired by the human brain. They were proven to be able to approximate every function [10] and can deal with very different input and output (e. g. pictures, audio features, text, categories). This makes them ideal for machines learning. During the last decade they have outperformed many other models in various machine learning tasks, including speech recognition, natural language processing and image classification [11, 12, 13]. Neural networks scale up well to millions of learnable parameters and provide the learning capacity required for new big data sets (i. e. the ImageNet data set has about 15 million images [14]). The recent success of NN is also due to the availability of increasing performance of computers. This is even further accelerated by the use of graphics processing units (GPUs) that are well suited for the training algorithms.

This sections will give a short definition of a feed-forward deep neural network (DNN) and then explain two variations, denoising autoencoders (DAEs) and time delayed neural networks (TDNNs), and some advanced methods that were used for the experiments in this work. The explanations will assume familiarity with the basics of machine learning. More extensive descriptions and extensions of NNs can be found in [15] and [16].

2.2.1. Deep Neural Network (DNN)

A deep neural network (DNN) is a feed-forward, artificial neural network that has more than one hidden layer of neurons. An example is in figure 2.6. Each hidden unit j typically uses the logistic function (equation 2.10 to map from the total input from the layer below, z_j to a scalar value y_j (also see figure 2.5). The weights w_{ij} and the bias b_j in each layer have to be learned. This can be done by backpropagating the derivatives of a cost function that measures discrepancy between the target outputs t_j and the actual outputs o_j of the top layer (output layer) [17]. As we require reference outputs t_j, we need labeled training data.

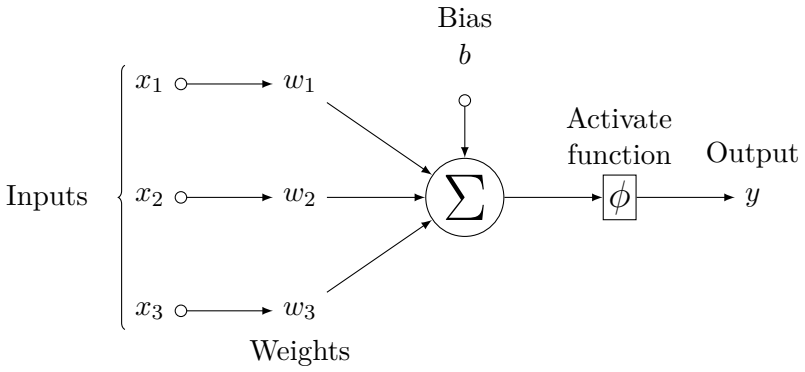

Figure 2.5.: Outline of an artificial neuron. The Σ node computes $z_j = b_j + \sum_{i=1}^{n} w_{ij} * x_i$ and ϕ computes the activation $y_j = \phi(z_j)$. There exist more powerful models of neurons like long short-term memory (LSTM) and the spiking neurons that are closer to biological neurons.

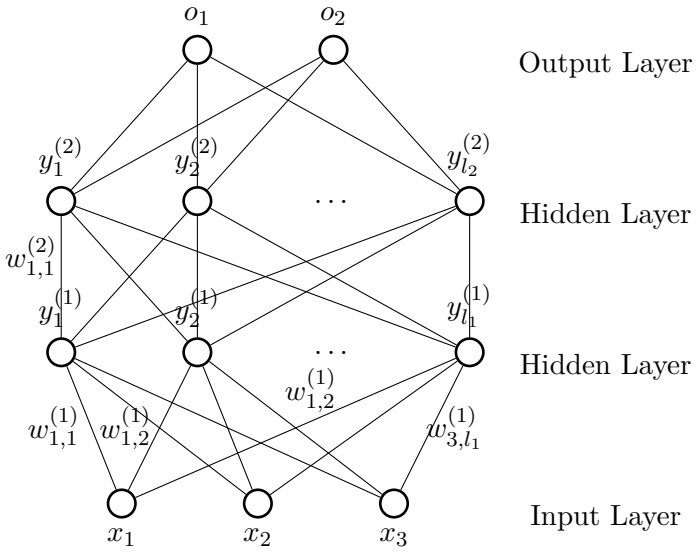

Figure 2.6.: A small DNN with 2 hidden layers. The input layer has 3 neurons that are fixed to the input values, two hidden layers and an output layer with 2 neurons.

2.2.2. Autoencoder

An autoencoder is a artifial neural network that is trained to learn a representation of its input in order to attempt to output a copy of it. The output is usually not of interest, but the hidden representation. If a layer with fewer neurons than the input size is inserted, as in figure 2.7, the autoencoder will learn a compressed encoding of the data. Similar to a principal component analysis (PCA) the encoding can be used for purposes of dimensionality reduction. In fact, if the network uses a linear activation function and is trained with mean-square error (MSE) it learns to span the same subspace as a PCA [16]. The k hidden units would correspond to the top k components of the PCA. Autoencoders with nonlinear activation function can learn a more powerful nonlinear encoding. Hinton et al. showed that they can be superior to PCA components [18].

If the hidden representation dimensionality is greater than the input dimensionality, the encoding is not a compression any more. Assuming that non-linearity and training will prevent learning the identity function the hidden representation learns complex features. Using multiple hidden layers leads to deep features that can be used as initialization for bigger models (see 2.2.4).

Denoising Autoencoder (DAE)

Denoising autoencoder (DAE) is a specific kind of autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point. Training autoencoders in this way will make them more robust to noisy data. It also increases the variance in the training and can prevent overfitting. The noise signal can be something that matches the domain (i. e. salt-and-pepper noise for images) or generic Gaussian noise. The input can also be masked by setting random random feature to zero.

Training DAE is very similar to undirected restricted Boltzmann machine (RBM) with Gaussian visible units [16].

2.2.3. Time-Delay Neural Network (TDNN)

As shown in 2.1 speech recognition is a sequence to sequence problem. The input is typical the features of multiple time steps. A normal DNN will be fully connected between the layers and thus learn an affine transformation of the entire temporal context. But intuitively there are transformations to be learned on narrow context. Deeper layers can then learn the wider context from the activations of the hidden activations. To achieve this, Waibel et al. introduced the time delayed neural network (TDNN) [19]. It layers are not fully connected, but each neuron only receives input from a small temporal context. Additionally the assumption is made that narrow features are time invariant and weights between neurons in the same layer but different time steps are shared. *Shared weights* are forced to the have the same value by averaging their weight updates. The transformation can be formulated as a convolution in the time domain:

$$z[n] = f[n] * w[n] = \sum_{t=-\infty}^{\infty} f[t]\, w[n-t] \tag{2.9}$$

where f corresponds to the input, w to the weights and z to the weighted sums of the neurons in the hidden layer. In speech recognition each input will be a feature vector, so $f[t]$ and $w[t]$ would be vectors as well. The similarity to a convolution also gave the network the name convolutional neural network (CNN).

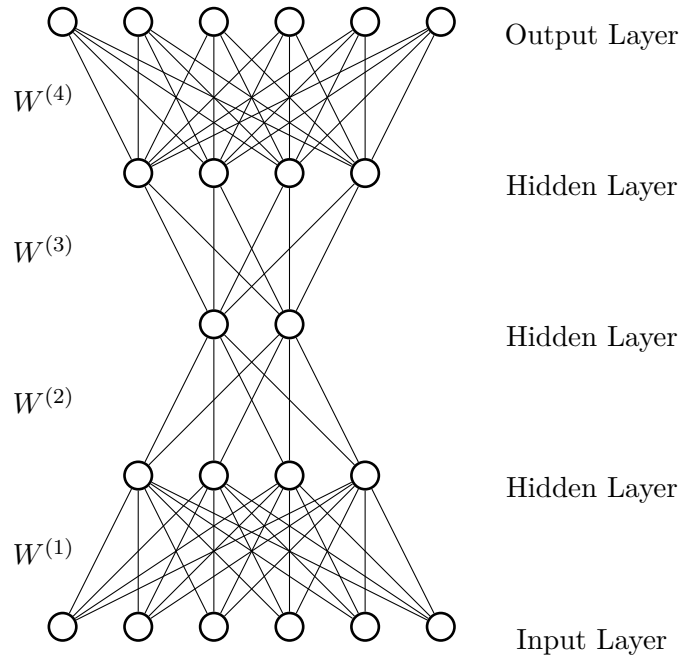

Figure 2.7.: An autoencoder with 3 hidden layers. The representation in the second hidden layer has only 2 neurons compared to the 6-dimensional input. None the less the NN is trained to output the original input. If we force $W^{(1)} = W^{(4)\intercal}$ and $W^{(2)} = W^{(3)\intercal}$ we say the **weights are tied**.
The representation in the second hidden layer could be used as compressed representation as show in 2.3.1

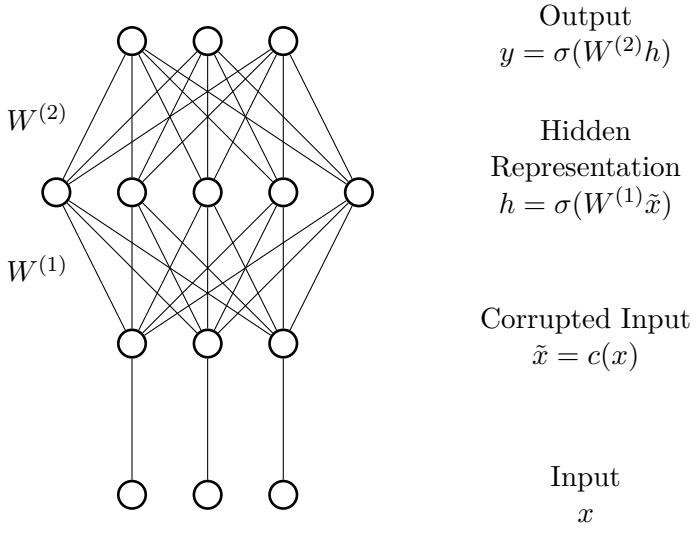

Figure 2.8.: Denoising autoencoder with a single hidden layer. The input x is corrupted using $c(x)$. The error is calculating between the original input x and the output of the NN y. Again weights can be tied, $W^{(2)} = W^{(1)\intercal}$

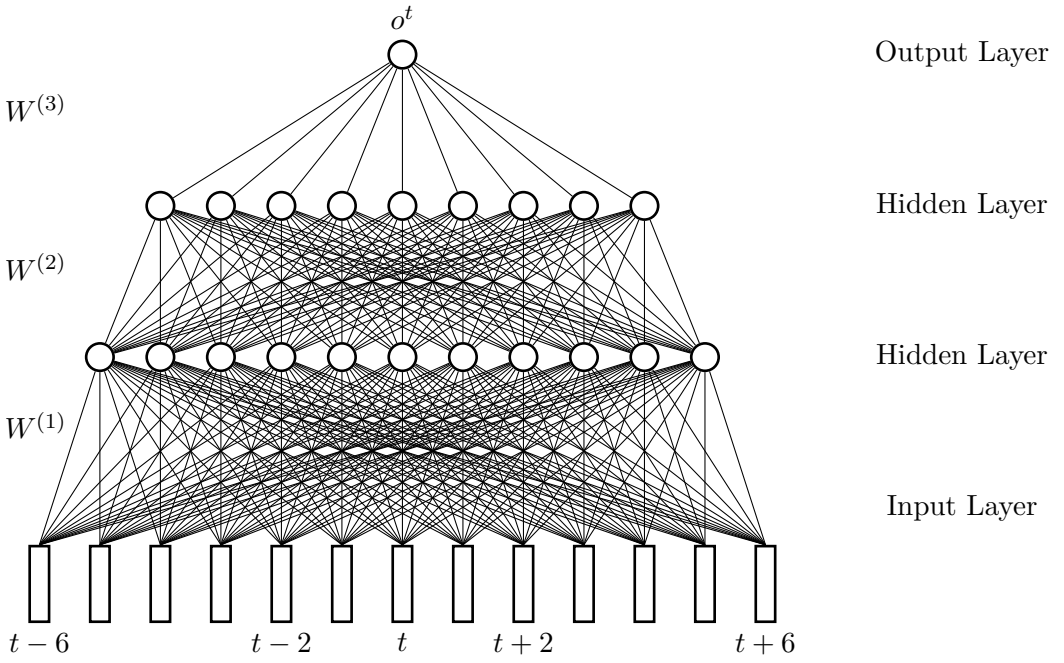

Figure 2.9.: A fully connected DNN with input context $[-6, 6]$ (see 2.3.1)

While the weight sharing leads to fewer parameters and thus a better generalization of the model, it is also more complex and will slow down the training. [20] reported a factor 10 in training time compared to a DNN with the same number of parameters. To deal with this the authors used subsampling.

Subsampling

Looking at 2.10 the filters overlap in each layer. The idea of subsampling is to apply the filter on fewer time points. This is undersampling the input signal. The natural subsampling would only be done to the input signal, but [20] showed that it is also possible to use it in the hidden layers. They report a speedup of factor 5 if subsampling is used in all layers.

The TDNN approach can be generalized to multiple dimensions. For speech recognition the frequency domain in the spectrum is the second dimension and for visual tasks many systems will use x and y coordinates and the color channel.

2.2.4. Advanced methods for training Neural Networks

As powerful as NN can be all their knowledge is inside the weights. Finding good weights can be tricky and the followings sections will list a few optimizations.

Activation Functions

Very important for the learning process is the activation function $\phi(x)$ of a neuron. Typically it will look similar to a heaviside step function and output values in the range $[-1, 1]$ or $[0, 1]$. If $\phi(x)$ is non-linear it can be shown that networks using it with at least one hidden layer can approximate any function, making them very powerful [10]. Further layers with linear activation function can be optimized away by changing weights and bias respectively and are therefore of little use.

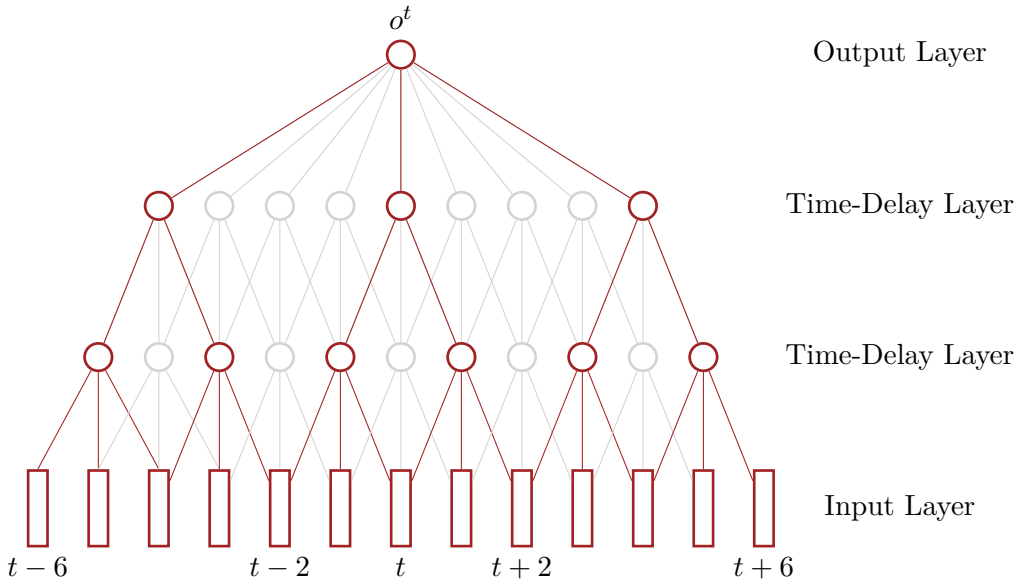

Figure 2.10.: Computation in a TDNN with subsampling (black) and without subsampling (black+gray). Both networks have the same number of parameters in the two time-delay layers.

The **Sigmoid** function is a very common activation function. The output lies between 0 and 1 allowing it to be interpreted as probability. However, using the Sigmoid-activation in deep networks can worsen the vanishing gradient problem [16].

$$\sigma(x) = \frac{1}{1 + e^{-\beta x}} \tag{2.10}$$

(β influences the slope, but is usually set to 1.)

The **Hyperbolic tangent** produces outputs in the interval $[-1, 1]$. While it is basically just a rescaled version of the Sigmoid function it's derivative is also bigger in around $x = 0$.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 1 - \frac{2}{e^{2x} + 1} \tag{2.11}$$

The **rectified linear unit (ReLU)** [21] is biology more plausible as the previous two. Its computation is also very simple making the training a bit faster[1]. The linear part has always a derivative of 1 and does not reinforce the vanishing gradient problem. For negative inputs the output and the derivative are always 0. Unlucky weight updates cause this for all samples in the training data. The neuron is then effectively dead.

$$\text{relu}(x) = \max(0, x) \tag{2.12}$$

Another important activation function is the **Softmax** function. It is a generalization of the Sigmoid function and operates on a vector of real values instead of a single scale. The input vector can be the weighted activations of the previous layer. The output will be a vector that can be interpreted as probability distribution. All output values will sum up to

[1] An equivalent definition of the Rectifier Linear Unit is $\text{relu}(x) = 0.5(x + |x|)$. This will not introduce a branch in code execution and was tested to run slightly faster on GPUs.

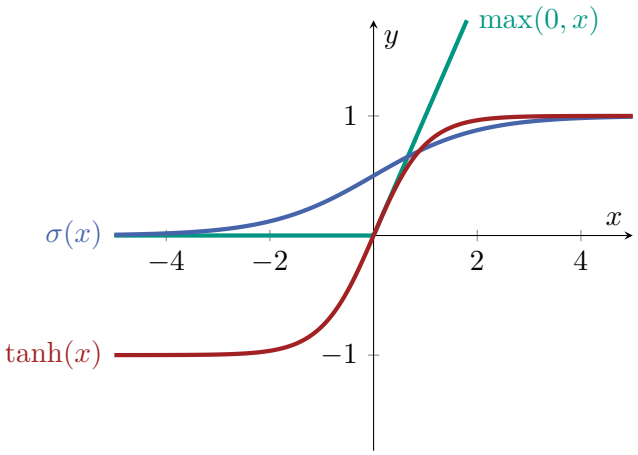

Figure 2.11.: Popular activation functions: Sigmoid function $\sigma(x)$, Hyperbolic Tangent $\tanh(x)$ and Rectifier Linear Unit $\text{relu}(x)$

1. Due the exponential terms values will be close to either 1 ore 0. In examples for inputs $[2, 5, 3, 1]$ the result will be $[0.04, 0.83, 0.11, 0.02]$.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.13}$$

More activation functions and derivatives can be found in [22].

Minibatch Gradient Descent

To train NN the Backpropagation algorithm, a generalization of the delta rule, is used [15, chapter 6.3]. After a forward pass of the NN errors are computed by applying an error function on the output values. Going backwards through the NN the errors for individual edges are computed. By deriving the error function we can then use gradient descent to update the weights and minimize the error. After that we start over with a new forward pass. If this is done for each training example, one at a time, it is called **stochastic gradient descent (SGD)**. While this can work, single training examples are often noisy resulting in sequences of inconsistent weight updates. On the opposite one could calculate errors and derivatives for all training examples and perform the best update for the whole data set by taking the average. This is called **batch gradient descent** and it will take the steepest route to the next minimum for the distribution of the training data. Unfortunately todays data sets are very large and a single iteration over all training examples can take up to several hours of calculations making batch gradient descent very inefficient.

Minibatch Gradient descent combines the benefits of both approaches. Instead of taking all or a single example a small number of training examples is used, usually 64-512 samples. The assumption is that each minibatch has a similar distribution as the whole data set and is less effected by noise than single examples. The data set is then divided into minibatches and weight updates are performed after each minibatch. The use of minibatches gives additional performance improvements. If the size is set correctly all required values will fit in the machines memory and calculations can be rewritten as big matrix multiplication that are easier to parallelize.

All experiments in this work use minibatches of 256 examples.

Generative Pretraining

Training DNN using gradient-based learning methods and backpropagation can cause the vanishing gradient problem. Backpropagation uses the chain rule to compute the gradient of the activation functions and many common activation functions have gradients in the range $(-1, 1)$. For many hidden layers multiplying many small gradients make the gradients in the front layers very small and slow down the learning.

Pretraining as introduced in [23] uses unsupervised methods for a meaningful initialization of the weights. Each layer is trained separately to learn how to represent it's input. The training does not require labeled data and can be done using DAE:

1. Train first autoencoder DAE_1 using the original training data X
2. Input X into DAE_1 and use output as X'
3. Train second autoencoder DAE_2 using X'
4. Use DAE_2 to transform X' to X''
5. Continue to create $\text{DAE}_3, \text{DAE}_4, \cdots$
6. Use weight matrices and biases from autoencoders to stack a DNN

The stacked deep autoencoder knows already how to extract complex features from the data. It is then fined tuned using backpropagation and supervised training data. While the pretraining learns representations to regenerate the original input (generative) the fine tuning selects deep features to discriminate classes. Analogous to DAEs in the algorithm above RBMs can be used.

Recent studies indicate that layer-wise pretraining is not necessary if normalization methods are used properly and there is enough data. But the technique shown can still speed up learning process.

NewBob Schedule

The learning rate α is critical for training process. If it is too small learning will take very long and is likely to stop in a bad local minimum of the cost function. In contrast high learning rates can lead to "jumps" over a good local minimum or fail to converge.

To solve this many approaches have been invented to adjust the learning rate dynamically. TODO: name and cite 2-3

A very simple, yet effective schedule is the NewBob schedule. It has two phases:

Phase 1: Start the training with a constant learning rate α_0. Continue Training until the validation error change from current to previous epoch falls below a threshold τ_1, then switch to Phase 2.

Phase 2: Continue training with exponential decreasing learning rate $\alpha_i = \beta^i * \alpha_0$, where $\beta \leq 1$ is the factor for exponential decrease (usually set to 0.5). Terminate training as soon as the change of the validation error is below a second threshold τ_2.

Using a high learning rate for Phase 1 allows the network to learn fast and skip local minimums. Phase 2 guaranties a conversion and fine tunes the learned weights.

2.3. Acoustic Models with Neural Networks

The GMM-HMM combination presented in section 2.1.1 was the standard model for speech recognition for over 20 years. Only recent advances in neural networks and the ever increasing computational power and new huge data sets showed NNs to be the better choice. There are various ways to use them in speech recognition and this section will briefly explain two: Bottle-Neck Features and Hybrid DNN-HMM systems

New approaches using recurrent neural networks (RNNs) and connectionist temporal classification (CTC) were not used in this work [24, 25].

2.3.1. Bottle-Neck Features

As seen in section 2.2.2 NN can be used to find compressed representations. This can help phoneme classifiers with acoustic context.

Acoustic context

In speech recognition frames are classified into phonemes. The standard frame width is 32 ms while phonemes can be as long as 200 ms. Classifiers can therefore be enhanced by providing feature vectors of neighboring frames. This is called temporal context and usually given in time steps $t-x, \ldots, t-1, t, t+1, \ldots, t+y$, where the current frame is t, the previous frame is $t-1$ and so on. The context goes from $[-x, y]$. While providing acoustical is likely to increase the frame classification accuracy the context increases the dimensionality of the feature vector. Our preprocessing, described in 5.1.1, produces 54-dimensional feature vectors. Using a acoustic context of $[-6, 6]$ we end up with 702 features.

Using a high dimensional input makes Gaussian mixture model (GMM) unpractical. To reduce dimensionality after adding acoustic context we can train an autoencoder to create a more compact representation:

1. Pretrain an DNN with any number of x hidden layers using stacked DAEs.
2. Add layers on top of the neural network: One bottle-neck layer with a small number of neurons and one or two layers mapping from the small bottle-neck to the output. Past experiments have shown that 42 is a good bottle-neck size for speech recognition.
3. Train the whole network in a supervised fashion.
4. Discard layers after the bottle-neck layer.

The final neural network can then be used to transform the features from the preprocessing into more powerful features. A GMM or a DNN, as we will show in the next section, can then be used to do the actual phoneme classification [26].

2.3.2. Hybrid DNN-HMM Systems

Using the softmax function from 2.2.4 we can train a NN to output a probability distribution for a multi-class classification task. In speech recognition hidden Markov models (HMMs) are used to deal with great variety in speech. To use them we need to run Viterbi Alignment and the Forward-Backward Algorithm. Both require to calculate $P(o_t|s)$, where o_t is the observation at time t (our feature vector) and s is the HMM state. Each HMM state corresponds to one class in our DNN, but the DNN result is $P(s|o_t)$. Thanks to Bayes' theorem we can calculate $P(o_t|s)$ from it:

$$P(o_t|s) = \frac{P(s|o_t) * P(o_t)}{P(s)} \tag{2.14}$$

$P(s)$ is the likelihood of seeing state s and can be directly estimated from our training data. $P(o_t)$ is the probability of seeing the specific features vector. It is unknown but will be equal for all states. Thus we neglect it and use $\frac{P(s|o_t)}{P(s)}$ as a scaled likelihood for being in state s at time t. With that we can use a DNN to replace GMMs in GMM-HMM acoustic models. The input can be traditional preprocessing features or bottle-neck features. The later help to increase the acoustic context and improve recognition accuracy [27]. Hybrid DNN-HMM models have shown state of the art results and tend to outperform GMM-HMM systems [11].

2.4. Room Impulse Response

The Acoustic impulse response (AIR) characterizes the acoustics of a given environment. In case the enclosure is a room it is more appropriate to refer to it as room impulse response (RIR) to imply some of the limits in the acoustic context. This section will introduce the characteristics of RIRs.

The acoustics of the room are influenced by it's size, the objects inside it and the reflectivity properties of those and the walls. The number of possible rooms is infinite and only simple cases can be simulated efficiently (e. g. empty cuboids). A RIR additionally depends on the position of the source of the signal and the receiver [28]. As shown in figure 2.12 the impulse response can be divided into three parts: the impulse of the direct path, the strong early reflections and the diffuse late reflections

Front-end based approaches to deal with dereverberation (see 3) often exploit that we can use this knowledge to divide an impulse response $h(t)$ into two parts:

$$h_{\text{early}}(t) = \begin{cases} h(t) & \text{if } t \leq t_e \\ 0 & \text{otherwise} \end{cases} \quad h_{\text{late}}(t) = \begin{cases} h(t+t_e) & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

t_e is the boundary between the early reflections and late reverberation, which is typically ranges from 30 to 60 ms after the arrival of the direct sound. Using 2.15 we can rewrite the microphone signal $y(t)$ the following:

$$y(t) = \sum_{l=-\infty}^{t} x(t)h(t-l) + v(t) \tag{2.16}$$

$$= \sum_{l=t-t_e}^{t} s(l)h_e(t-l) + \sum_{l=-\infty}^{t-t_e} s(l)h_l(t-l) + v(t) \tag{2.17}$$

where $v(t)$ denotes to the additive ambient noise component, the first sum is the early speech component $z_e(t)$ and the second sum is late reverberant component $z_l(t)$. The joint suppression of $z_l(t)$ and $v(t)$ can be done using spectral enhancement. It increases the speech fidelity and intelligibility.

Measurement

RIRs can be measured by emitting an impulse and recording the received signal. Even so simple approaches as a gun shot work to create an impulse with a good signal-to-noise ratio (SNR), exponential sine sweeps are suited better. They provide high energy for the low frequencies, resulting in higher accuracy [29, 30]. An examples of such a impulse response is plotted in figure 2.13.

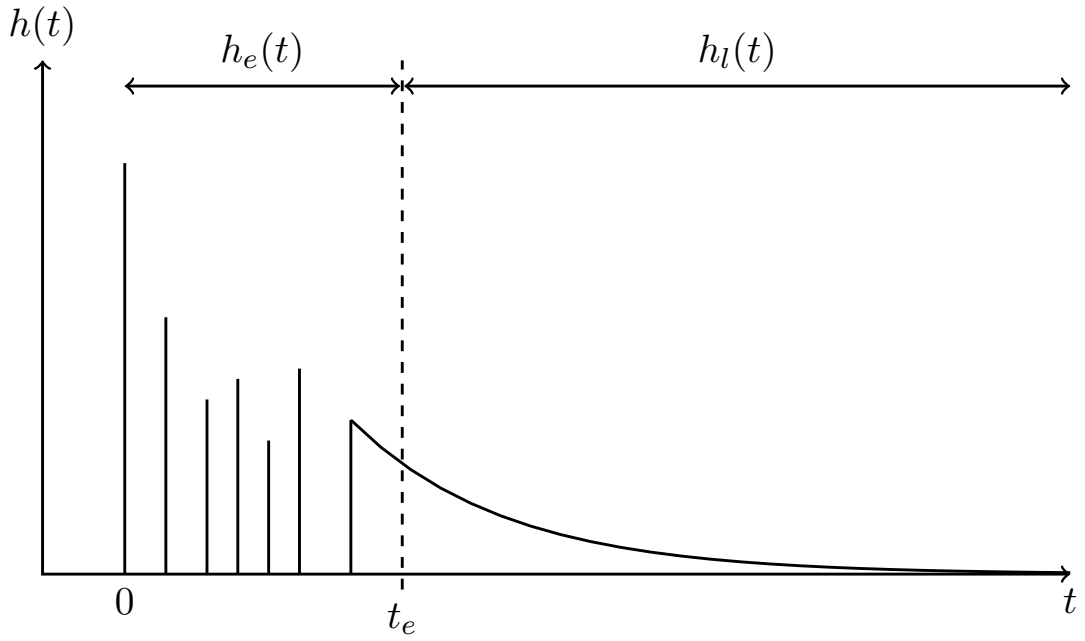

Figure 2.12.: Schematic view of a RIR. The impulse response $h(t)$ is split into two parts $h_e(t)$ $h_l(t)$ by the boundary t_e.

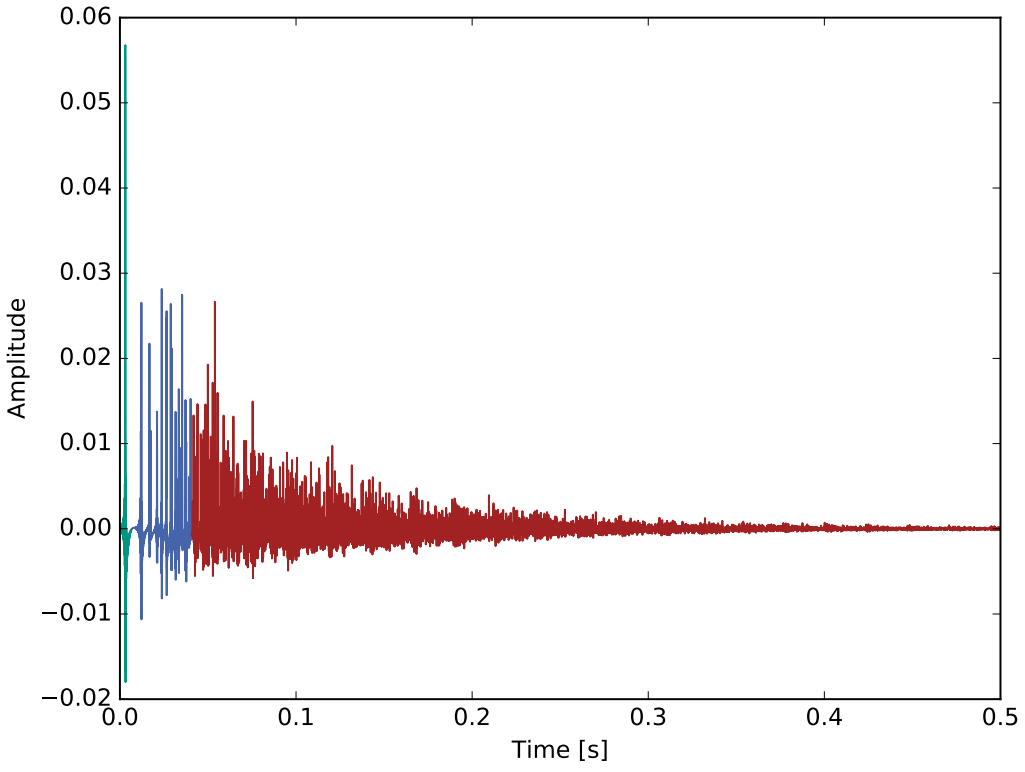

Figure 2.13.: Example of a measured room impulse response (RIR) with the direct sound-path (green), early reflections (blue) and late reflections (red) and $T_{60} = 0.7$. The impulse of direct-path sound is not always clearly visible.

Reverberation Time

An impulse response can be quantified by the reverberation time T_{60}. It is defined as the time it takes for a signal to decay by 60 dB relative to the level of direct sound [31, chapter 1.5]. For typical office and home environments, the reverberation time lies between 0.2 and 1 seconds. The reverberation time is governed by the geometry and the reflectivity of the reflecting surfaces and in approximation independent of the receiver position.

3. Related Work

Reverberation is a long known problem for speech recognition and a lot of research has gone into building robust systems. The authors of [32] classify the different approaches in a common framework.

- **Front-End based approaches** aim at remove the features passed to the acoustic model by inserting additional steps into the preprocessing. Depending on the position there are three types:
  - **Time domain**: *Linear filtering* exploits both the amplitudes and phases of the signal, which is advantageous in terms of accuracy because reverberation is a superposition of numerous time-shifted and attenuated versions of a clean signal. Linear filtering allows exploiting the acoustical differences between multiple microphone positions. It has been successfully been applied to actual meeting data [33].
  - **Spectrum**: The objective of *spectral enhancement* is to restore the clean power spectrum coefficients. Because it deals with the late reverberation, which is largely insensitive to source and receiver position, it is robust to speaker movement.
  - **Log Spectrum**: The *Feature Enhancement* methods try to model the effect of reverberation on log Mel-frequency filterbank features. An example method is given in [34].
- **Back-End based approaches** aim at adjusting the parameters of the acoustic model. They exploit the statistical properties of reverberation in the final features. Examples are HMM adaptation [35] and acoustic context-dependent likelihood evaluation [36].

Front-end based approaches do not requires changes in the back-end. They become part of the preprocessing. Their computational complexity does not increase with the acoustic model size as it is the case for back-end based approaches. They can also be combined with other advanced recognition techniques. On the downside various assumption about the acoustic environment are done that can introduce estimation errors. Overall back-end based approaches seem to perform better [32].

More methods and detailed explanations can be found in [31]. We think that these explicit methods increase the system complexity. Therefore we focused on newer approaches using NN. The next section shows work already done on it.

3.1. Dereverberating Autoencoder

Ishii et al. proposed a DAE to reconstruct the clean speech spectrum from reverberated speech [2]. The autoencoder is first layer-wise pre-trained using artificially reverberated speech and fine tuned by providing features of the clean audio signal as reference audio. The authors found that an acoustic context of as big as $[-11, 11]$ produced the highest recognition accuracy. The features calculated by the autoencoder were fed directly into the HMM.

Feng et al. followed a similar approach, but used a GMM-HMM acoustic model [3]. Their autoencoder takes an input of 15 ($t-7$ to $t+7$) frames of reverberated features and aims to output the clean features of the center frame t. They used the CHiME-WSJ0 corpus, which is only medium-vocabulary size (5k), but showed improvements up to 25 % absolute word error rate (WER).

Finally, [37] presented a combination of a dereverberating autoencoder, using bidirectional LSTMs, and spectral subtraction. The combination of both, data- and model-based approaches, outperforms each on it own.

3.2. ASpIRE Challenge

In 2015 the Intelligence Advanced Research Projects Activity (IARPA) held the Automatic Speech recognition in Reverberant Environments (ASpIRE) Challenge [38, 39], an evaluation campaign focusing on speech recognition in reverberant speech recognition. Limited to the Fisher corpus [40] for the training set participants had to construct automatic speech recognition systems robust to far field recordings. The Fisher corpus consists of approximately 2,000 hours of transcribed telephone speech. Algorithmic transformation of it as additional training data were allowed. In contrast the provided development set and the evaluation set consisted of recordings from several different rooms with different distances between speaker and microphone to create a variety of acoustic environments.

The development set was taken from the *Mixer 6 Corpus* (LDC2013S03), while the evaluation set was taken from *Mixer 8 Pilot Corpus* and had more variability in terms of 7 instead of 2 rooms and 2-3 speaker locations per room instead of 1.

ASpIRE Challenge had two tracks, one for single microphone and one for multiple microphones (6 channels). In terms of WER the only submitted multiple microphone system performed best. The system is described in [41]. The single microphone track had three winners [42], who described their systems in [4, 5, 41]. All three teams used data augmentation to increase the disparity in the training data. There are different methods for it:

- Adding random background noises as described in [4].
- Applying room impulse responses. Hsiao et al. used generated room impulse responses while the other teams relied on real impulse responses.
- [5] also tested speed and volume perturbation, but neither showed a gain.

Beside from multi conditioning the training data systems benefit from increased temporal context, feature enhancement with DAE, and a TDNN. Systems with better speech activity detection performed better [43].

4. Implementation

Training a state of the art speech recognition is a complex task. Many substeps are required to train the models for language and acoustics and fine tune the parameters (e. g. master beam, language model weight, warp factor). The experiments for this work were only possible through the provided frameworks Janus Recognition Toolkit (JRTk) and *detl*.

The following pages will give an overview of the tools and the functionality that was added as part of this thesis. Programming was done in C and Python.

4.1. Janus Recognition Toolkit (JRTk)

The Janus Recognition Toolkit (JRTk), or just *Janus* for short, is a general-purpose framework for speech recognition [44, 45]. It can be used as a static library and through a Tcl/Tk environment. The Tool Command Language (Tcl) allows a high level abstraction on objects and algorithms while the highly optimized C code provides fast execution with relative small memory footprint. The flexibility is a great advantage, especially in research, over other toolboxes for automatic speech recognition (ASR) (i. e. *kaldi*[1], *HTK*[2]).

JRTk includes methods for audio preprocessing, various types of language models, acoustic modeling with GMM-HMM, standard ASR algorithms and the IBIS decoder [46]. The IBIS decoder is a one-pass decoder that makes use of the concept of linguistic context polymorphism and is therefore able to incorporate linguistic knowledge at an early stage. Using hybrid DNN-HMM systems is possible, but the implementation prior to this work was limited to simple feed-forward networks.

During this work development of the toolkit was pushed forward. The additions target three aspects:

- The existing Makefiles were replaced with a new *CMake*[3] build system. CMake offers great cross-platform support and allows users to program with the compiler environment of their choice. CMake can search and find libraries itself while Makefiles require libraries to be in by the users [47]. *SCons*[4] and *Autoconf*[5] were also considered, but found to be less suited for the project. Extensive documentation of the new build system was added to the internal Wiki of the JRTk repository.

[1] `http://kaldi-asr.org/`
[2] `http://htk.eng.cam.ac.uk/`
[3] `https://cmake.org/`
[4] `http://scons.org/`
[5] `http://www.gnu.org/software/autoconf/autoconf.html`

- A Python interface for the objects and methods in the JRTk was developed. The interface is implemented as a Python module and can be used as any other module, from the Python shell or written scripts.
- Python scripts that use the new Python module to train and test hybrid DNN-HMM systems were created. The scripts incorporate *detl* (see 4.2) for training and evaluating neural networks. This improves the execution speed and allows the use of complex NN architectures in the acoustic model. Using *detl* also lead to 0.1 % absolute improvement in the WER in some cases. It was not investigated, but slightly different implementation of the network evaluation are plausible.

In the following the old Tcl and the new Python interface are explained.

4.1.1. Tcl Interface

The JRTk provides an interface to Tcl by wrapping the standard Tcl interpreter. The wrapper deals with JRTk specific objects and methods while passing pure Tcl to the standard Tcl interpreter, running in the background. The wrapper is deeply integrated into the C library. It manages objects, parses arguments and cleans up memory. The deep integration provides a great user experience, as if the classes and functionality were part of Tcl, but also makes using JRTk without Tcl very hard. This is worsen as some pure C methods rely on the registration of variables through the wrapper and some logic exits inside interface methods. A great amount of functionality is implemented in Tcl directly.

4.1.2. Python Interface

Python has become very popular in scientific computing and offers many tools for it - NumPy, SciPy, Pandas, matplotlib and Pylearn to name just a few. This includes also many frameworks for machines learning and training neural networks. But no toolkit targeting speech recognition is available. The Python syntax is easy to learn. On the other side the Tcl syntax, used by the existing JRTk interface, is less self-explaining to beginners and does not provide a smooth integration of neural networks libraries.

The Python module for JRTk was inspired by a proof of concept by Florian Metze. He used *ctypes*[6] to run a simple decoder. After a comparison of ctypes, Cython and SWIG it was decided that Cython offers the best control over memory and performance (see next section for details about it).

Table 4.1 shows the structure of the Python module. The JRTk C code is divided into several folders (base, models, features, etc.). The Python module follows this structure to split the nearly 200[7] classes into several submodules. A main prerequisite for the Python module was to leave the Tcl interface untouched. This will allow users to transition when they are ready. Once the Python interface is complete and stable the Tcl interface will be removed to decrease further engineering effort.

A detailed documentation on the Python module, targeting users and developers, was created in the internal JRTk Wiki[8]. All experiments, presented in this work, were run through the newly created Python interface. The investigation on TDNNs would not have been possible with the old Tcl interface.

[6] ctypes is a foreign function library that provides C compatible data types. It is part of the Python standard.
[7] The Tcl interface discloses 193 types. It has to be determined if all of them will be required in the future.
[8] The JRTk Wiki is part of the bitbucket repository. `https://bitbucket.org/jrtk/janus/wiki/Home`

Filename Pattern	Purpose
setup.py.in	Install information for Python package manager
___init___.py	Module meta data, module initialization and loading of submodules. The file is executed when the module is imported.
c_*.pxd	C definitions that do not belong in a submodule
common.pxi	Cython include file with some basic imports, should be included at the beginning of every Cython source file (*.pyx). Sets up NumPy.
core.pyx	Tcl interpreter class and logging configuration
itf.pyx, itf.pxd	Helpers methods to register and unregister objects to the Tcl interpreter
MODULE.pxd	Define classes of the submodule MODULE
MODULE.pyx	Common imports for all classes in MODULE
MODULE/c_*.pxd	C definitions needed by the MODULE (one file for each C header file)
MODULE/*.pxi	Define Python classes and methods (one file for each Python class)

Table 4.1.: Structure of the Python module. Keeping compatibility with the Tcl interface introduced some overhead and can be removed in the future.

4.1.3. Cython

The Cython programming language is a superset of the Python programing language. It has full support for Python and the same syntax, but also lets developers access C/C++ methods, structs and classes. Programs are compiled to pure C code using the Cython compiler. A C-compatible compiler can then be used to compile the C code into a dynamic library. The library will have the necessary interface to be imported as module in Python.

While Python is dynamically typed, C requires static types. The Cython compiler will try to deduce the types of variables or use generic Python objects. The code may contain type annotations to help the compiler. This allows writing high level Python code, but speeding up performance critical sections by explicitly defining type and fall back to C functions.

Because of the bridging between C and Python Cython is often used to wrap C library into Python modules. The latest version of the Cython compiler is 0.24 (released on 04/04/2016).

4.2. detl

detl is a Python library for deep learning and available as Python module and command-line toolbox. It uses the *theano* library [48] to compile executions graphs for CPUs and GPUs. Theano takes care of many low-level optimizations for fast GPU training, while detl implements common models and algorithms. The later includes different neural layers (e. g. fully connected, convolutional, autoencoders), over a dozen activation functions and advanced methods like momentum, newbob scheduling and weight regularization.

During the work on this thesis detl was ported from Python 2 to Python 3. The Python 2 branch is no longer developed and Python 3 offers many new features and performance

database	number of rooms	number of RIRs
ACE [50]	7	14
AIR [51]	16	214
MARDY [52]	1	9
OMNI [53]	3	468
RWCP [54]	3	118
total	30	823

Table 4.2.: Sources for professional recorded RIRs used for training and testing. In MARDY all recordings were done in the same room, but the acoustic characteristic (e. g. reflectivity of the walls) was varied. If a database provided multi-channel recordings only the first channel was used. Multiple microphones can definitely help the reverberation problem but were not part of this work.

improvements. Unfortunately it is not backward compatible and requires many, though often tiny, changes. Supporting both versions would be possible, but tedious.

- The pfile reader now supports asymmetric context. Thus when reading in feature files detl can now built feature vectors that contain features for the frames $t_{-x}, \ldots, t_{-1}, t_0, t_1, \ldots, t_y$ where x and y no longer have to be the same. Experiments have shown that asymmetric context can help speech recognition tasks [49].
- The convolutional layer now supports sub-sampling as explained in section 2.2.3.
- Scripts for constructing a TDNN were added.
- A backup hook was added that periodically saves the current model. This was necessary as some of the GPUs used for experiments only allowed training up to 48 hours. If didn't finish within the time the job was canceled and the data lost. Trainings of neural networks with bigger amounts of training data or complex structure were trained up to 144 hours.

4.3. Collection of Room Impulse Responses

To built a speech recognition system that can deal with many different environments the acoustic characteristics of various rooms have to be learned. As argued in section 2.4 impulse response contain these information. To get a high variety in training and test data impulse responses from various rooms are required. Collecting data is always expensive, but fortunately there are already several databases of room impulse responses available. Table 4.2 shows those used in this work.

Audio format and sampling rate did not always match the training data. To correct this and organize the files Python scripts were written[9]. The scripts also create lists that split the 823 RIRs into a training set (658 RIRs), a dev set (82 RIRs), a test set (83 RIRs) and several other subsets for individual rooms.

4.4. Generating Room Impulse Response

In addition to collecting real room impulse responses the 'Room Impulse Response Generator' tool from E. Habets[10] was used to create synthetic room impulse responses. The tool uses

[9] `https://github.com/Marvin182/rir-database`
[10] `https://www.audiolabs-erlangen.de/fau/professor/habets/software/rir-generator`

the frequently used image method, proposed by Allen and Berkeley in [55]. Given the room size, reflection coefficients of the walls and position of sender and receiver, the algorithm generates an impulse response. For a swift integration into the preprocessing a thin Python wrapper was added[11].

The virtual rooms don't have windows, doors and do not contain obstacles. Further the generated impulse response does not have noise. Thus the synthetic RIRs are less suited than real world RIRs.

[11] https://github.com/Marvin182/rir-generator

5. Evaluation

To evaluate the capabilities of DNNs for acoustic models for reverberated environments several experiments were performed. This chapter will first describe the baseline speech recognition system and the data used for the experiments. After that each experiment is explained and followed by a short discussion. Experiments performed include a comparison between measured and generated RIRs, training for a single room, the use of bigger temporal context and finally the use of a TDNN for the acoustic model.

5.1. Baseline System

The baseline system is a hybrid DNN-HMM with context dependent triphones and a n-gram language model. The next section describe the components and their training. The acoustic model training is kick-started by using labels generated from a development GMM-HMM system.

5.1.1. Preprocessing

The preprocessing takes a 16 kHz audio as input. The audio is divided into frames using a frame shift of 10 ms and a window size of 32 ms. From each frame 40 log Mel frequency features and 14 tonal features are calculated. As demonstrated in [56] the tonal features give small gains even for non tonal languages as English. Mean subtraction, weighted by a speech detection measure, is applied on the combined features. At last features are stacked to a temporal context, 13 frames (+/- 6) if not mentioned otherwise.

5.1.2. Acoustic Model

The acoustic model is a hybrid DNN-HMM. 8000 context dependent tri-phonemes are used. Hence the output layer of the DNN is always a softmax layer with 8000 units. The input are the features from the preprocessing with the temporal context. If not mentioned otherwise a DNN with 5 hidden layers is used. Each hidden layer has 1200 hidden units and uses sigmoid activation functions. The network is trained as follows:

1. Initialize connections weights and biases to uniform random values between -0.2 and 0.2
2. Preform pretraining as described in 2.2.4, using tied weights, a learning rate of 0.01 and mask corruption with probability 0.2. Each layer is pretrained for 300k minibatches.

Text corpus	Number of words in million
TED	3
News + News-commentary + -crawl	4,478
Euronews	0.780
Commoncrawl	0.185
GIGA	2323
Europarl + UN + multi-UN	829
Google Books	(1 billion n-grams)

Table 5.1.: Sources of the language model. After data cleaning the total number of words was 7.8 billion, not counting the Google Books.

3. Fine tune the network using stochastic gradient descent (SGD) with minibatches of 256 and the newbob learning rate schedule. The initial learning rate is 1, but will decay with a factor of 0.5 after the validation error did decrease less than 0.005 the first time. The training will stop after the validation error decreased less than 0.001.

The training is performed on a single GPU and runs in less than 2 days.

5.1.3. Frame Labels

While it is possible to train new speech recognition systems from scratch using DNNs [57] it is faster to start from labels generated from an existing system. We use a strong GMM-HMM system to write labels for the training data. All experiments use the same labels.

5.1.4. Language Model and Vocabulary

The language model is identical to the English language model in [58]. It was build from various sources (see table 5.1). For each (sub-)corpora a separate n-gram language model with modified Kneser-Ney smoothing was build. The final model is a linear interpolation of those models. Held-out data from the TED corpus was used to tune the interpolation weights.

For the vocabulary unigram language models from all text sources were built using Witten-Bell smoothing. Unigram probabilities were then chosen to maximize likelihood of held-out TED data set. The top 150k words become the vocabluary. All experiments used the same language model and vocabulary.

5.2. Training Data

Beside the room impulse responses described in 4.3 transcribed TED talks and noises are used to train the models.

5.2.1. TED Talks

The main audio corpus is extracted from TED-LIUM corpus release 2 [59]. It contains English TED talks. TED is a nonprofit organization that invites thinkers and doers to give fascinating talks about their ideas. The talks range from 10 to 15 minutes. While some speaker are well prepared it is still spontaneous speech. The recordings are high quality but remain very challenging due to the large variability of topics and the presence of non-native speakers.

After an automatic segmentation and removing a disallowed talk, which is part of the test set, 168 h of labeled audio remained. The 723 talks are split into 107117 utterances (average utterance length: 5.6 s). Utterances shorter than 300 ms were removed from the training data. Due to errors in the labeling process 1166 utterances could not be labeled and were not used to train the acoustic model. This 1 % loss of training data was not further investigated.

5.2.2. Noises

Additional to the TED talks some noise samples were added to train the phonemes modeling noises. Training the acoustic model to learn models for common noises is very important and reduces the WER. The noises, in total 10 h, were:

- 240 minutes of microphone and signal disturbances
- 160 minutes of 5 s long samples from various radio songs.
- 70 minutes of pause and silence
- 60 minutes of rustle
- 40 minutes of various non-speech humans sounds (i. e. sniffing, smacking)
- 30 minutes of applause

5.3. Test Sets

As test set we use the evaluation set for the English ASR track of the International Workshop on Spoken Language Translation (IWSLT) 2013. For the IWSLT 2015 it was published as development set, including transcripts and automatic segmentation. The set contains 28 TED talks, with a total of 4.2 hours of speech. The segmentation gave 1388 utterances with an average length of 10.9 s. Again the TED talks were recorded in high quality. We refer to this test set as *tst2013* and use it to measure the performance of our systems on clean, close-talk speech.

To test performance in a reverberant environment we created a reverberated version of tst2013, called *tst2013_reverb*. We used a selection of 28 RIRs from our RIR test set (see 4.3) to convolve each talk with one RIR. These 28 RIRs were picked to maximize the number of simulated rooms in tst2013_reverb.

In a similar fashion we created another reverberated test set for a single room, *tst2013_classroom*. The 28 RIRs that roughly form a grid were taken from the set of RIRs available for the *classroom* in the OMNI database [53].

5.4. Experiments

This next sections contain the result of our experiments. We describe each experiment, give the results and finish with a discussion.

5.4.1. Multi condition training

To train our system on different reverberant conditions we use the following steps to obtain a feature vector from an utterance:

1. Sample random RIR $h(t)$ from training RIRs
2. Resample $h(t)$ to match sampling rate of utterance

3. Remove silence at the beginning of $h(t)$
4. Convolve utterance audio with $h(t)$, including samples before utterance window that could cause reverberation.
5. Run preprocessing to retrieve log Mel frequency and tonal features.

The removal of silence in step 3 is important. Silence at the beginning of a RIR can occur if the microphone is several meters away. The speed of sound is 342.2 $\frac{\text{m}}{\text{s}}$, so that it will take the sound 10 ms to travel 3.42 m. The delay does apply to the direct sound path and all reflection and the resulting signal will be the same as without silence, but shifted in time. While in real time application this will introduce a small latency, it does not effect the speech recognition itself. However, in order to use the frame labels generated for the clean audio using the GMM-HMM label system, a time shift would worsen the quality of the labels.

We choose to convolve each utterance with a random chosen RIR instead of using one for a whole talk (speaker) because it creates higher variability in the training data. The results in table 5.2 confirm this.

Reverberating each utterance individually gives better result. Therefore all further experiments use the steps presented above. Next we trained systems using measured and generated RIRs. Our baseline is a system trained on clean speech as described in 5.1. It achieves 19.1 % WER on the tst2013 test set. This is similar to other state of the art systems [49]. On the reverberated tst2013_reverb in performance degrades to 59.7 %, which reveals great room for improvement and highlights again the need for reverberation robust speech recognition.

Table 5.3 shows different systems trained with multi condition data. The best system, Reverb Real, achieves a WER of 41.9 %. This is an relative improvement of 30 % over baseline. Peddinti et al. reported a relative improvement of 35 % when switching from clean to reverberant and noisy training data [5].

The systems trained with generated RIRs, Reverb Gen A and Reverb Gen B, show significant improvements over the baseline, but cannot challenge the system trained with measured RIRs. The test set tst2013_reverb was created using measured RIRs. These can contain noise and the rooms can be very complex and big. The image method used to simulate the RIRs can only create RIRs of small and simple rooms.

All systems were also tested against the tst2013 (clean speech) and it was found that the system trained with reverberated audio do not match the baseline. We tried mixing clean and reverberated training data, but with decreasing WER on clean speech the WER on reverberated speech started increasing.

5.4.2. Single condition training

In this scenario it was investigated if a system can be trained explicitly for one room. We choose the *classroom* from the OMNI database [53]. The authors measured 130

	tst2013	tst_reverb
per speaker	26.9 %	42.5 %
per utterance	26.2 %	41.9 %

Table 5.2.: Convolving each utterance with a different impulse response instead of using one for all utterances of a speaker gives a gain in system performance. Our training set has 723 different speakers, intuitively the difference should be bigger for small training sets.

System name	Training data	tst2013	tst_reverb
Baseline	clean	19.1 %	59.7 %
Reverb Gen A	reverb	22.2 %	48.1 %
Reverb Gen B	reverb	24.2 %	46.5 %
Reverb Real	reverb	26.2 %	41.9 %

Table 5.3.: Evaluation of different system trained and tested on clean and reverberant data. *Reverb Gen A* is trained with artificially generated impulse responses for rooms not bigger than $5\,\mathrm{m}^3 \times 5\,\mathrm{m}^3 \times 3\,\mathrm{m}^3$. For *Reverb Gen B* rooms up to $8\,\mathrm{m}^3 \times 9.5\,\mathrm{m}^3 \times 4\,\mathrm{m}^3$ were allowed and *Reverb Real* used real room impulse responses.

impulse responses in a grid. The room measures roughly $7.5\,\mathrm{m} \times 9\,\mathrm{m} \times 3.5\,\mathrm{m}$ ($236\,\mathrm{m}^3$) with reflective surfaces of a linoleum floor, a large whiteboard, painted plaster walls and ceiling. The conditions are bad for far-field speech recognition. The WER of 94.6 % by the baseline system confirms this.

We used up to 100 RIRs for training and the remaining 30 RIRs to create *tst2013_classroom* from tst2013 in the same fashion as tst2013_reverb. Additional we tested with a single RIR for the whole test set to get word error rates for specific receiver positions.

We trained several systems using different number of impulse responses from the classroom. The results are summarized by table 5.4. The error rates range form 59.1 % for all 100 RIRs to 78.3 % for 1 RIR in the middle of the room. While 1 RIR is clearly not enough, 5 RIRs seem to provide enough diversity to learn to room acoustics and get close to the system trained with 100 RIRs. Measuring 5 RIRs for a room is feasible and could be used for system explicitly build for a single room.

An addition we also tested the systems trained with RIRs from various (simulated) rooms. While the system trained with generated RIRs is not able to deal with the distortions the system trained on the measured impulse response does as good as 60.2 %. This is due that some RIRs from the classroom are part of its training set. Using only RIRs from other rooms for training, as done for system *Reverb Real** yields a WER of 69.5 %. The neural network did learn features for reverberant speech from other rooms.

Further we evaluated how the performance correlates with the distance between speaker and receiver. Figure 5.1 shows the results for the system trained with 100 RIRs from the classrom. For positions closer to the speaker the system could handle reverberated audio a lot better. The worst results can be seen on the outsides of the first row. This is similar to the findings in [43]. The authors suggest that ASR system performance is better correlated with a measure that depends not only on the distance but also on the orientations of both speaker and receiver. Similar graphics for the baseline system and the system trained with a single RIR from the middle of the room can be found in the appendix.

5.4.3. Temporal Context

To create the tst2013_reverb the tst2013 test set was convolved with RIRs from various rooms. These rooms include big lecture halls and thus have reverberation times up to 1 second. The temporal context of our baseline system is $[-6, 6]$, which are 13 frames. We use a window size of 32 ms and 10 ms shifts. Thus the input to the DNN covers 152 ms. This is enough to cover the direct sound path and some early reflection of most phonemes but the network will miss most of the late reflections. For clean speech it has been observed that simply increasing the input context of the DNN can reduce the frame recognition

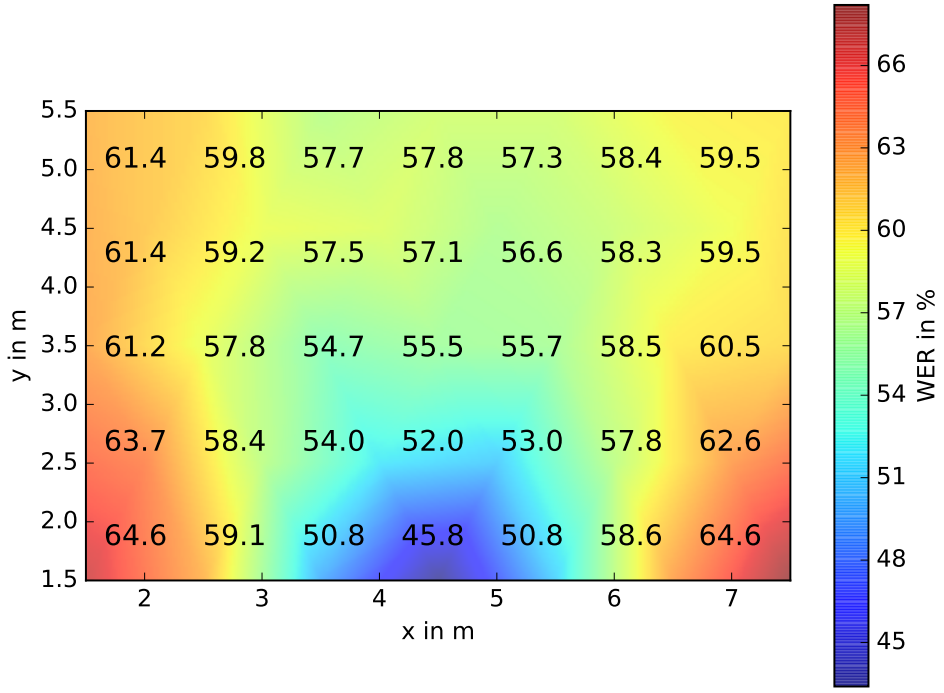

Figure 5.1.: Word error rate depending on the receiver position in the classroom. The acoustic model was trained with 100 RIRs from the room and tested on the remaining 30 RIRs. The background color is a linear interpolation between the measurement points. It is not said that the WER degrades linearly, but the visualization highlights that the WER does depend on the distance and the microphone angle. The speaker is positioned at $x = 4.5$ and $y = 0.5$.

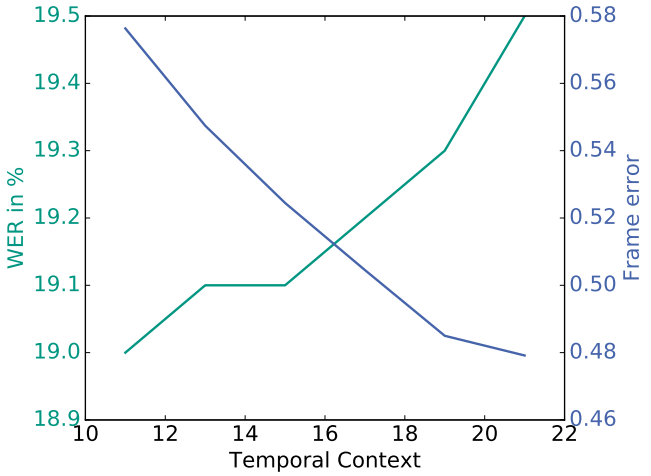

Figure 5.2.: WER (blue) on tst2013 and frame classification error (green) for DNN acoustic models with different temporal context. The context is given as number of frames and is always symmetric (e. g. context 13 indicates that the network it given the features from frame $t - 6$ to $t + 6$). The system was trained with clean speech.

error but at the same time increases the WER. We verified that this is the case for our system as well and that the optimal context window is indeed $[-6, 6]$ (see figure 5.2.

By running the same experiments with reverberated training data and testing against reverberated training data we found that the optimal context for a reverberant system is $[-8, 8]$ (see figure 5.3). We conclude that a bigger temporal context is necessary for reverberation robust speech recognition, but the phenomenon of increasing WER despite decreasing frame error remains.

5.4.4. Bottle-Neck features

As explained in section 2.3.1 Bottle-neck features (BNF) can improve the system performance by providing more powerful features. If used in speech recognition a DNN with a bottle-neck is trained in the first step and the bottle-neck output is then used as input for the classifier. The classifier can be another DNN. Our bottle-neck network has 5 hidden layers of size 1200 and a bottle-neck with 42 units.

The two steps allow 4 different combinations of training with clean and reverberant speech, as displayed in table 5.5. When trained with clean data in both steps the system does not learn to handle reverberation. If reverberant audio is used in one of the steps it does better than the baseline, but does not reach the performance of a single DNN trained on reverberated data. But in case of both steps use multi-condition training the system achieves better result on both test sets, compared to our previous system *Reverb Real* (see section 5.4.1). Even so the BNF network and the classifier network only have an input context of $[-6, 6]$ each the total input context is $[-12, 12]$ (Reverb Real uses $[-8, 8]$). Additional the two networks can be seen as one very deep network that is able to discover very complex features.

5.4.5. Time Delay Neural Network

As another NN architecture we chose TDNN. As its computation is simlar to a convolution is should be able to undo a convolution, if possible at all. Our TDNN consists of 3 time-delay layers, each with 200 filters and 3 fully connected layers with 500 units each on

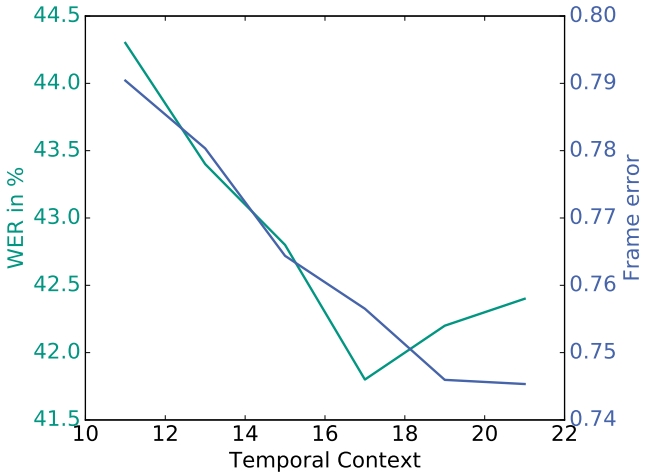

Figure 5.3.: WER (blue) on tst2013_rvb and frame classification error (green) for DNN acoustic models with different temporal context. The models were trained on reverberated audio from measured RIRs.

top of it. Due the weight sharing in the lower layers the number of parameters is smaller than in the DNN (8M for the TDNN and 16M for the DNN). However, training time is about equal.

Table 5.6 compares the performance of the three network structures. Unfortunately results for the TDNN were not satisfying. We did not see the improvements reported in [49]. Further the use of subsampling gave only a small boost in training speed, but also 0.3 % increase in WER. Experiments with the context size and activations functions did not yield better results.

RIRs for training	tst_classroom
0 (Baseline)	94.6 %
1	78.3 %
5	63.5 %
10	63.0 %
50	62.5 %
100	59.1 %
Reverb Gen	80.7 %
Reverb Real	60.2 %
Reverb Real*	69.5 %

Table 5.4.: WER of systems trained with different number of RIRs from the classroom in the OMNI database and tested against other RIRs from the same room. The *Reverb Gen* system is trained with simulated RIRs for rooms close to the classroom dimensions. The *Reverb Real* systems was trained with all 658 training RIRs which also include RIRs from the classroom. For *Reverb Real** these were explicitly removed.

BNF training	DNN training	tst2013	tst2013_reverb
clean	clean	19.3 %	60.4 %
clean	reverb	21.8 %	47.1 %
reverb	clean	22.2 %	50.0 %
reverb	reverb	24.8 %	41.1 %

Table 5.5.: Results for BNF-DNN-HMM systems: First the BNF network was trained. Afterwards the training data was transformed into the bottle-neck features and used to train the classifier DNN.

Network Type	Number for parameters	tst2013_reverb
DNN	16M	41.8 %
BNF-DNN	32M	41.1 %
TDNN	8M	48.4 %

Table 5.6.: Performance of the 3 network architectures. All three models were trained on reverberant speech for roughly 1.5 days.

6. Conclusion

In this final chapter the achievements of the thesis are first summarized and analyzed. After that some ideas for future work are expressed.

6.1. Summary

The aim of this work was to investigate the use of neural networks for lecture speech recognition in reverberant environments. This can be accomplished by training under multiple (reverberant) conditions. As collecting new training would be expensive and a tedious work we followed the approach of using room impulse responses (RIRs) to add the acoustics of a room to close-talk speech recordings. A database of professionally measured room impulse responses was collected from five different sources. In addition code for generating room impulse for small rooms was adopted. The RIRs were then used to reverberate lecture talks.

In the evaluation phase we trained hybrid DNN-HMM acoustic models with the reverberated training data. By testing against a baseline on both clean and reverberated speech we found that the use of measured RIRs gives great improvements on reverberant speech (59.7 % down to 41.8 % WER) but performance on clean speech degrades (19.1 % up to 26.2 %).

Improvements on close-talk capturing were not the goal of the approach but the performance lost should be kept in mind. The decreased WER for reverberant speech is similar to those found by [5], reporting a drop from 47.6 % to 31.7 % WER on far-field recordings in meetings rooms. The amount of measured RIRs is limited and more data could help to close the gap between reverberant and clean speech recognition rates. Generating RIRs is another alternative. For now generated RIRs did produce slightly higher errors (46.5 % on reverberant speech). Possible reasons for this were given in 5.4.1.

Next we investigated the case of a single condition, were RIRs of a room are given and reverberant speech from the same room has to be recognized. The room was a very reverberant, big classroom and very challenging for far-field speech recognition. The experiments showed that using more than 5 RIRs gave only small gains. Measuring 5 impulse responses seems feasible and improved WER for microphones within a close distance (1-2 meters) by 28.0 % (from 70.9 % to 43.3 %). This could help to build speech speech recognition systems for lecture rooms that do not require the speaker to be equipped with a microphone.

To further improve the performance we tested variations in the acoustic model. This showed that an increased acoustic context [8, 8] is advantageous in reverberant environments. The use of bottle-neck features lowered the error rate from 41.8 % to 41.1 %. This improvement is significant, but smaller than in [60].

And finally the use of a time-delay neural network was examined. The network did show better training and validation error on frame level, but failed to produce better word sequences. This is a widely known phenomenon. Adjusting parameters of the speech recognition system (e. g. language model weight, master beam) did not solve the problem.

All the experiments described above were done using a newly created Python interface of the Janus Recognition Toolkit (JRTk). Python features superior performance and usability over the obsolete Tcl interface. The switch to Python will allow new users to become familiar with JRTk quickly and allow faster evaluation of new approaches. This is due to the simplicity and the powerful Python ecosystem as described in 4.1.

This work presented a simple approach to train reverberation robust speech recognizer that works well for multi- and single-conditions. The outcomes were submitted to the ITG conference on speech communication and will be published via IEEExplore.

6.2. Future Work

The approaches presented in this thesis targeted the reverberation problem. In many far-field scenarios the speech recognition is also confronted with additive noise. The framework developed during this thesis can be extended to create noisy conditions. This was done successfully for telephone speech in [4]. To investigate reverberation separately presented experiments were performed on artificially reverberated data. Further work should use test sets by evaluation campaigns on far-field speech recognition (e. g. ASpIRE and REVERB).

The time-delay neural network outperformed simple DNNs on frame level recognition, but not from a WER perspective. Finding the causes for this could yield better performance on both, clean and reverberant, speech.

Lastly the scope of this thesis did not allow investigation on neural networks other than feed-forward networks. As new findings indicate recurrent neural networks achieve state of the art results ASR [61]. Theie use in reverberant speech recognition has still to be examined.

Bibliography

[1] D. Giuliani, M. Matassoni, and M. Omologo, "Training of HMM with filtered speech material for hands-free recognition," *International Conference on Acoustics, Speech, and Signal Processing - Proceedings*, 1999.

[2] T. Ishii, H. Komiyama, T. Shinozaki, Y. Horiuchi, and S. Kuroiwa, "Reverberant speech recognition based on denoising autoencoder.," in *INTERSPEECH*, pp. 3512–3516, 2013.

[3] X. Feng, Y. Zhang, and J. Glass, "Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 1759–1763, 2014.

[4] R. Hsiao, J. Ma, W. Hartmann, M. Karafiat, F. Grezl, L. Burget, I. Szoke, J. H. Cernocky, S. Watanabe, Z. Chen, S. H. Mallidi, H. Hermansky, S. Tsakalidis, and R. Schwartz, "Robust Speech Recognition In Unknown Reverberant And Noisy Conditions," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, p. 533, dec 2015.

[5] V. Peddinti, G. Chen, V. Manohar, T. Ko, D. Povey, and S. Khudanpur, "JHU ASpIRE System : Robust Lvcsr with TDNNs, iVector Adaptation and RNN-LMs," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, p. 539, dec 2015.

[6] S. Young, G. Evermann, M. Gales, and T. Hain, "The HTK Book (v3. 4)," *Cambridge University*, 2006.

[7] X. Huang, A. Acero, and H. Hon, "Spoken language processing: A guide to theory, algorithm, and system development," 2001.

[8] B. Ziółko and M. Ziółko, "Time durations of phonemes in Polish language for speech and speaker recognition," *Language and Technology Conference*, 2009.

[9] A. P. Dempster, N. Laird, and D.B. Rubin, *Maximum Likelihood from Incomplete Data via the EM Algorithm*, vol. 39. 1977.

[10] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, 1989.

[11] G. Hinton, L. Deng, D. Yu, and G. Dahl, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012.

[12] R. Collobert, J. Weston, L. Bottou, and M. Karlen, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493—-2537, 2011.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[14] J. Deng, W. Dong, R. Socher, L. Li, and K. Li, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248—-255, 2009.

[15] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. JOHN WlLEY & SONS, INC., 2000.

[16] I. G. Y. Bengio and A. Courville, "Deep Learning." 2016.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

[18] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks.," *Science (New York, N.Y.)*, vol. 313, pp. 504–7, jul 2006.

[19] A. Waibel, T. Hanazawa, and G. Hinton, "Phoneme recognition using time-delay neural networks," *IEEE transactions on*, 1989.

[20] H. X. Chen Yun-Nung Hakkani-Tur Dilek, C. Yun-Nung, and Y.-n. Chen, "Detecting Actionable Items In Meetings By Convolutional Deep Structured Semantic Models," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, p. 375, dec 2015.

[21] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2146–2153, 2009.

[22] M. Ritter, "Neural Networks - Formulae Collection," 2015.

[23] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, pp. 1527–54, jul 2006.

[24] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks," *Proceedings of the 23rd international conference on Machine Learning*, pp. 369–376, 2006.

[25] Y. Miao, M. Gowayyed, F. Metze, M. Y. Gowayyed Mohammad Metze Florian, M. F. Gowayyed Mohammad, Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pp. 167—-174, dec 2015.

[26] F. Grézl, M. Karafiát, S. Kontár, and J. Černocký, "Probabilistic and bottle-neck features for LVCSR of meetings," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 4, 2007.

[27] J. Gehring, Y. Y. Miao, F. Metze, and A. Waibel, "Extracting deep bottleneck features using stacked auto-encoders," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3377–3381, IEEE, 2013.

[28] H. Kuttruff, *Room acoustics*. Spon Press/Taylor & Francis, 2009.

[29] M. Holters, T. Corbach, and U. Zölzer, "Impulse Response Measurement Techniques and their Applicability in the Real World," in *Proc. 12th Int. Conference on Digital Audio Effects*, vol. 9, 2009.

[30] A. Farina, "Simultaneous measurement of impulse response and distortion with a swept-sine technique," *Proc. AES 108th conv, Paris, France*, no. I, pp. 1–15, 2000.

[31] P. A. Naylor and N. D. Gaubitch, *Speech dereverberation*. Springer Science & Business Media, 2010.

[32] T. Yoshioka, A. Sehr, M. Delcroix, K. Kinoshita, R. Maas, T. Nakatani, and W. Kellermann, "Making machines understand us in reverberant rooms: robustness against reverberation for automatic speech recognition," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 114–126, 2012.

[33] T. Hori, S. Araki, T. Yoshioka, and M. Fujimoto, "Low-latency real-time meeting recognition and understanding using distant microphones and omni-directional camera," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 499—-513, 2012.

[34] A. Krueger and R. Haeb-Umbach, "A model-based approach to joint compensation of noise and reverberation for speech recognition," *Robust speech recognition of uncertain or missing data*, pp. 257–290, 2011.

[35] T. Takiguchi, M. Nishimura, and Y. Ariki, "Acoustic model adaptation using first-order linear prediction for reverberant speech," *IEICE transactions on information and systems*, vol. 93, no. 3, pp. 908–914, 2006.

[36] A. Sehr, R. Maas, and W. Kellermann, "Reverberation model-based decoding in the logmelspec domain for robust distant-talking speech recognition," *IEEE transactions on audio, speech, and language processing*, vol. 18, no. 7, pp. 1676–1691, 2010.

[37] F. Weninger, S. Watanabe, Y. Tachioka, and B. Schuller, "Deep Recurrent De-Noising Auto-Encoder and Blind De-Reverberation for Reverberated Speech Recognition," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, no. 2, pp. 4656–4660, 2014.

[38] IARPA, "Automatic Speech recognition In Reverberant Environments (ASpIRE) Challenge," 2015.

[39] Mary Harper, "The Automatic Speech Recogition In Reverberant Environments (Aspire) Challenge," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, p. 547, dec 2015.

[40] C. Cieri, D. Miller, and K. Walker, "The Fisher Corpus: a Resource for the Next Generations of Speech-to-Text.," *LREC*, vol. 4, pp. 69–71, 2004.

[41] J. Dennis and T. H. Dat, "Single And Multi-Channel Approaches For Distant Speech Recognition Under Noisy Reverberant Conditions: I2R'S System Description For The Aspire Challenge," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, p. 518, dec 2015.

[42] IARPA, "IARPA Announces Winners of its ASpIRE Challenge," 2015.

[43] J. Melot, N. Malyska, J. Ray, and S. Wade, "Analysis Of Factors Affecting System Performance In The Aspire Challenge," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, p. 512, dec 2015.

[44] M. Finke, P. Geutner, H. Hild, T. Kemp, K. Ries, and M. Westphal, "The Karlsruhe-Verbmobil speech recognition engine," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, 1997. ICASSP-97*, vol. 1, pp. 83–86, IEEE, 1997.

[45] M. Woszczyna, N. Aoki-Waibel, F. D. Buo, N. Coccaro, K. Horiguchi, T. Kemp, A. Lavie, A. McNair, T. Polzin, and I. Rogina, "JANUS 93: towards spontaneous speech translation," *1994 IEEE International Conference on Acoustics, Speech, and Signal Processing, 1994. ICASSP-94*, vol. 1, pp. I–345, 1994.

[46] H. Soltau, F. Metze, C. Fugen, C. Fügen, and A. Waibel, "A one-pass decoder based on polymorphic linguistic context assignment," in *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001. ASRU'01*, pp. 214–217, IEEE, 2001.

[47] K. Martin and B. Hoffman, *Mastering CMake - A cross-platform build system.* [Clifton Park, NY.]: Kitware Inc., 2015.

[48] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A CPU and GPU math compiler in Python," in *Proc. 9th Python in Science Conf*, pp. 1–7, 2010.

[49] V. Peddinti, D. Povey, and S. Khudanpur, "A Time Delay Neural Network Architecture for Efficient Modeling of Long Temporal Contexts," *Interspeech*, pp. 3214–3218, 2015.

[50] J. Eaton, N. D. Gaubitch, A. H. Moore, and P. A. Naylor, "The ACE challenge - Corpus description and performance evaluation," in *2015 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 1–5, IEEE, 2015.

[51] M. Jeub, M. Schäfer, and P. Vary, "A binaural room impulse response database for the evaluation of dereverberation algorithms," in *16th International Conference on Digital Signal Processing*, pp. 1–5, IEEE, 2009.

[52] J. Y. C. Wen, N. D. Gaubitch, E. a. P. Habets, T. Myatt, and P. a. Naylor, "Evaluation of Speech Dereverberation Algorithms using the MARDY Database," *Proc. Intl. Workshop Acoust. Echo Noise Control (IWAENC)*, pp. 12–15, 2006.

[53] R. Stewart and M. B. Sandler, "Database of omnidirectional and B-format room impulse responses.," in *ICASSP*, pp. 165–168, 2010.

[54] S. Nakamura, K. Hiyane, F. Asano, T. Nishiura, and T. Yamada, "Acoustical Sound Database in Real Environments for Sound Scene Understanding and Hands-Free Speech Recognition.," in *LREC*, pp. 2–5, 2000.

[55] J. B. Allen and D. A. Berkley, "Image method for efficiently simulating small-room acoustics," *The Journal of the Acoustical Society of America*, vol. 65, no. 4, pp. 943–950, 1979.

[56] F. Metze, Z. A. W. Sheikh, A. Waibel, J. Gehring, K. Kilgour, Q. B. Nguyen, and V. H. Nguyen, "Models of tone for tonal and non-tonal languages," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pp. 261–266, IEEE, 2013.

[57] L. Zhu, K. Kilgour, and S. Stüker, "Gaussian Free Cluster Tree Construction Using Deep Neural Network," *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[58] K. Kilgour, M. Heck, M. Müller, and M. Sperber, "The 2014 KIT IWSLT Speech-to-Text Systems for English, German and Italian," *International Workshop on Spoken Language Translation (IWSLT)*, 2014.

[59] A. Rousseau, P. Deléglise, and Y. Estève, "Enhancing the TED-LIUM Corpus with Selected Data for Language Modeling and More TED Talks.," in *LREC*, pp. 3935–3939, 2014.

[60] J. Gehring, W. Lee, K. Kilgour, I. Lane, Y. Miao, and A. Waibel, "Modular Combination of Deep Neural Networks for Acoustic Modeling," *Proc. Interspeech*, no. August, pp. 94–98, 2013.

[61] Y. Miao, M. Gowayyed, X. Na, T. Ko, F. Metze, and A. Waibel, "An Empirical Exploration of CTC Acoustic Models," *Icassp 2016*, pp. 2623–2627, 2016.

Acronyms

AIR acoustic impulse response. 17

AM acoustic model. 5

ASpIRE Automatic Speech recognition in Reverberant Environments. 21

ASR automatic speech recognition. 1, 4, 5, 7, 22, 29, 37

BNF Bottle-neck features. 3, 33

CNN convolutional neural network. 10

CTC connectionist temporal classification. 16

DAE denoising autoencoder. 8, 10, 15, 16, 21

DNN deep neural network. 3, 8–10, 12, 15–17, 27, 28, 31

EM expectation-maximization. 6, 7

GMM Gaussian mixture model. 6, 7, 16, 17, 22

GPU graphics processing unit. 8, 13, 25, 28

HMM hidden Markov model. 3, 5, 6, 16, 21, 22

IARPA Intelligence Advanced Research Projects Activity. 21

IWSLT International Workshop on Spoken Language Translation. 29

JRTk Janus Recognition Toolkit. 22, 23, 37

LM language model. 5

LSTM long short-term memory. 9, 21

LVCSR large-vocabulary continuous speech recognition. iv

MSE mean-square error. 10

NN neural network. 1, 3, 8, 11, 12, 14, 16, 20

PCA principal component analysis. 10

RBM restricted Boltzmann machine. 10, 15

ReLU rectified linear unit. 13

RIR room impulse response. 3, 17, 18, 25–27, 29–31, 34

RNN recurrent neural network. 16

SGD stochastic gradient descent. 14, 28

SNR signal-to-noise ratio. 17

STT speech to text. 4

Tcl Tool Command Language. 22

TDNN time delayed neural network. 3, 8, 10, 21, 23, 25, 27

WER word error rate. iv, 8, 21, 23, 29–31, 33

Appendix

A. WER per speaker and RIR

B. Performance depending on receiver position

Speaker ID	Source Database	Room Name	single condition clean	single condition reverb	multi condition clean	multi condition reverb
talkid1541	ACE	building_lobby_2	8.3 %	24.4 %	11.0 %	15.9 %
talkid1673	ACE	office_1	29.2 %	63.0 %	40.9 %	50.9 %
talkid1649	AIR	aula_carolina	21.8 %	99.1 %	27.6 %	62.8 %
talkid1617	AIR	aula_carolina	22.8 %	89.9 %	29.1 %	50.3 %
talkid1658	AIR	booth	18.4 %	23.9 %	25.4 %	26.8 %
talkid1654	AIR	lecture	6.2 %	70.4 %	9.1 %	23.8 %
talkid1694	AIR	lecture	44.1 %	88.6 %	56.0 %	67.9 %
talkid1651	AIR	meeting	13.5 %	28.9 %	17.8 %	21.8 %
talkid1637	AIR	meeting	9.1 %	19.3 %	15.0 %	17.8 %
talkid1520	AIR	office	21.1 %	36.1 %	32.1 %	34.5 %
talkid1659	AIR	stairway	10.1 %	15.8 %	15.0 %	15.9 %
talkid1640	AIR	stairway	16.1 %	88.2 %	22.3 %	43.8 %
talkid1685	AIR	bathroom	37.5 %	44.5 %	42.7 %	43.9 %
talkid1532	AIR	lecture2	16.8 %	53.4 %	24.2 %	38.2 %
talkid1647	AIR	meeting	17.0 %	17.3 %	22.4 %	22.3 %
talkid1518	AIR	stairway3	17.2 %	83.1 %	22.3 %	49.2 %
talkid1646	MARDY	2_c	28.6 %	49.9 %	34.3 %	40.8 %
talkid1666	OMNI	greathall	9.5 %	90.0 %	13.5 %	35.3 %
talkid1548	OMNI	greathall	16.6 %	95.1 %	22.0 %	50.6 %
talkid1699	OMNI	octagon	46.2 %	96.1 %	50.4 %	80.6 %
talkid1610	OMNI	octagon	12.5 %	98.0 %	14.6 %	49.6 %
talkid1534	OMNI	classroom	23.5 %	96.2 %	31.8 %	71.8 %
talkid1553	OMNI	classroom	10.6 %	95.4 %	12.9 %	54.9 %
talkid1665	RWCP	ane	28.3 %	63.0 %	34.4 %	49.4 %
talkid1600	RWCP	e1a	23.6 %	63.6 %	38.4 %	52.1 %
talkid1592	RWCP	e2a	6.4 %	69.7 %	13.4 %	37.2 %
talkid1539	RWCP	jr1	14.1 %	97.2 %	20.1 %	86.9 %
talkid1634	RWCP	ofc	7.6 %	97.4 %	13.4 %	86.5 %

Table A.1.: Comparison of the baseline DNN-HMM system trained on clean data (single condition) and trained on reverberated data (multi condition). The clean columns show word error rates for the IWSLT 2013 evaluation set on a per speaker basis. For the reverb columns every talk was convolved with a room impulse response from the given room and tested against the reverberated audio. Impulse response were not used twice. If two talks were reverberated with the same room the receiver position inside the room differs.

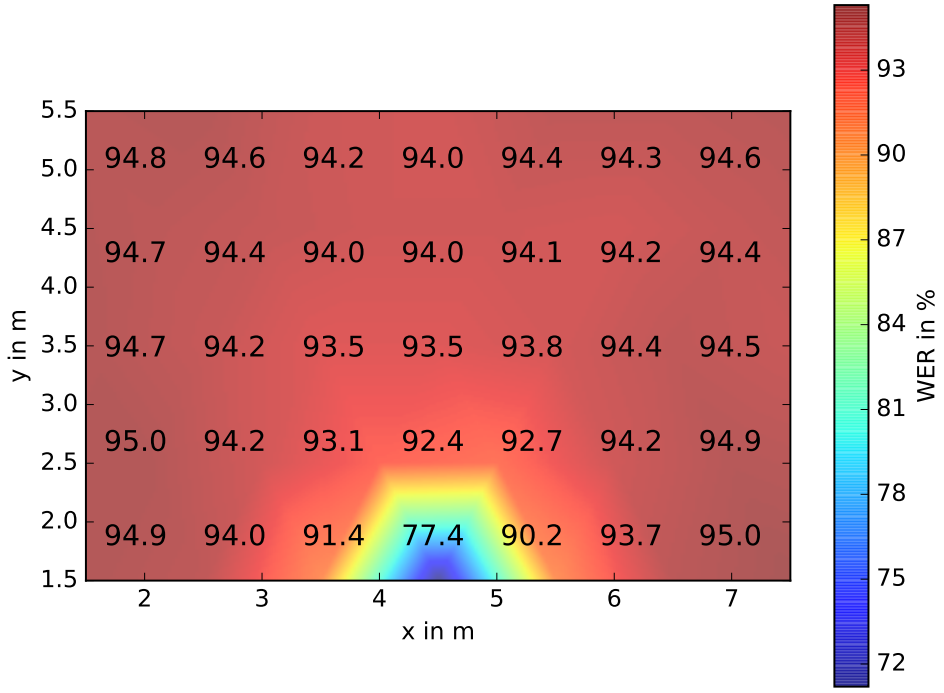

Figure B.1.: Word error rate depending on the receiver position in the classroom for the baseline system. The system cannot deal with reverberation except when directly in front of the speaker. Even then it is falls short to the system trained with one or more RIRs from the classroom.

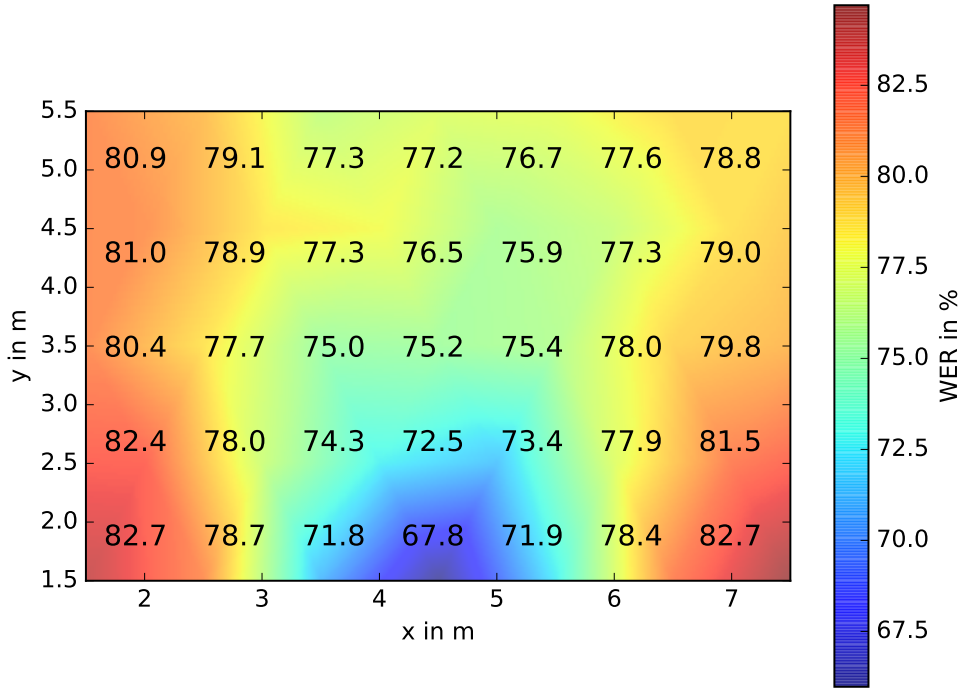

Figure B.2.: Word error rate depending on the receiver position in the classroom for a system trained with a single RIR from the middle of the room.