CLEMENS VAYDA, BSC

WOLFGANG HRAUDA, BSC

WAKE-UP WORD DETECTION USING LSTM NEURAL NETWORKS

Master's thesis

Graz University of Technology

Institute for Signal Processing and Speech Communication (SPSC)
Head: Univ. Prof. Dipl.-Ing. Dr.techn. Gernot Kubin

Supervisor:
Dipl.-Ing. Dr.techn. Martin Hagmüller

Co-Supervisors:
Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf
Dipl.-Ing. BSc Matthias Zöhrer

Graz, May 2016

TITLE OF THE THESIS:
Wake-up Word Detection using LSTM Neural Networks

SUBMITTED BY:
Clemens Vayda, BSc
Matriculation number: 0931235

Wolfgang Hrauda, BSc
Matriculation number: 0973083

STUDY:
Masterstudium Elektrotechnik-Toningenieur
Identification number: F 033 213

SUPERVISOR:
Dipl.-Ing. Dr.techn. Martin Hagmüller

CO-SUPERVISORS:
Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf
Dipl.-Ing. BSc Matthias Zöhrer

ABSTRACT

Wake-up word (WuW) detection is used to put an intelligent device in a state of alert so that it expects further spoken commands. It allows for hands-free operation of devices such as smart phones, multimedia systems in cars or home automation systems.
Recently, Google researchers were able to outperform standard HMM-based systems on WuW detection tasks with a template-based method using LSTM networks. The network was trained on an enormous amount of speech data (2500h) and was then used to extract fixed-length representations from speech features.
In the present thesis, we re-implement their approach and evaluate its potential with extremely limited training resources (1-5 h). We investigate how to best exploit the available resources and deal with practical problems such as the appropriate preparation of training data. We show that WuW detection can be performed despite the limited resources, with equal error rates down to 8% and less for certain speakers. The results provide evidence that for a more robust performance, a larger training database (> 50h) is necessary.

ZUSAMMENFASSUNG

Wake-up-word (WuW)-Erkennung dient dazu, ein intelligentes Gerät in einen Alarmmodus zu versetzen, in welchem es weitere Sprachbefehle entgegennimmt. So wird die berührungslose Bedienung von Geräten wie Smartphones, Multimediasystemen in Autos oder Heimautomatisierungssystemen möglich.
Kürzlich übertrafen Entwickler von Google konventionelle HMM-basierte Systeme in der WuW-Erkennung mit einem auf Referenzmustervergleich basierten System, das LSTM-Netzwerke verwendet. Zum Trainieren des Netzwerks benutzen sie eine enorme Menge an Trainingsdaten (2500h) und verwenden das so trainierte Netz um Repräsentationen gleicher Länge aus den Sprachmerkmalen zu gewinnen.
In der vorliegenden Arbeit beschäftigen wir uns mit der Reimplementierung dieser Idee und evaluieren das Potential dieses Ansatzes für extrem limitierte Trainingsressourcen (1-5 Stunden). Wir untersuchen, wie man die zur Verfügung stehenden Ressourcen am besten nutzen kann und setzen uns mit praktischen Problemen wie der entsprechenden Aufbereitung der Trainingsdaten auseinander. Dieser Ansatz ist in der Lage, trotz der beschränkten Ressourcen Gleichfehlerraten von bis zu 8% zu erreichen, beziehungsweise diesen Wert für ausgewählte Sprecher sogar zu unterschreiten. Die Ergebnisse untermauern, dass eine größere Trainingsdatenbank (> 50 Stunden) notwendig ist, um eine stabilere Erkennung zu gewährleisten.

ACKNOWLEDGMENTS

First of all, we want to thank Martin Hagmüller for his constant support and valuable advice throughout the thesis. Whenever we were in trouble, he took the time to listen to us. After every meeting, things became more clear and we could continue our work with renewed focus and energy.

Furthermore, we are grateful to Matthias Zöhrer for the sophisticated technical help with Python and for providing a start-up framework. Thank you for your time and for all the fruitful discussions.

We would like to thank Franz Pernkopf as well for providing his profound knowledge at the right times.

Our most heartfelt thanks go to Magdalena and Natalia, our precious wives who both provided us with best support, care and food.

And finally, we really appreciate the numerous tips and the good atmosphere of the DSP-Lab community.

STATUTORY DECLARATION

We declare that we have authored this thesis independently, that we have not used other than the declared sources / resources, and that we have explicitly marked all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present Master's thesis.

date	signature
date	signature

EIDESSTATTLICHE ERKLÄRUNG[1]

Wir erklären an Eides statt, dass wird die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kentlich gemacht haben. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, am Datum	Unterschrift
Graz, am Datum	Unterschrift

1 Deutsche Fassung: Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

CONTENTS

ACRONYMS

CC	Cross-Correlation
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
DNN	Deep Neural Network
DTW	Dynamic Time Warping
ED	Euclidean Distance
EER	Equal Error Rate
FA	False Alarm
FFT	Fast Fourier Transform
FR	False Reject
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
KWS	KeyWord Spotting
LR	Learning Rate
LSTM	Long Short-Term Memory
MFCC	Mel-Frequency Cepstral Coefficients
NN	Neural Network
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VAD	Voice Activity Detection
WER	Word Error Rate
WuW	Wake-up Word

Part I

THEORY

1 INTRODUCTION

1.1 MOTIVATION

Hands-free operation of machines based on speech recognition allows for fast, natural and convenient interaction. Despite all advances in understanding spoken commands, it is difficult for machines to decide whether an utterance is actually intended as a command or if it is just conversational speech.
One solution is to have the user push a button before uttering a command. However, this greatly reduces the aforementioned flexibility and convenience of hands-free systems. An approach that retains these advantages is to use a pre-defined word or short phrase to *wake up* the machine and signal that the following speech will be a command.
The problem of recognizing this so-called *Wake-up Word (WuW)* is a sub-field of automatic speech recognition. The task and its major challenges are precisely put into words by Këpuska and Klein:

> WUW SR [speech recognition] is defined as detection of a single word or phrase when spoken in the alerting context of requesting attention, while rejecting all other words, phrases, sounds, noises and other acoustic events and the same word or phrase spoken in non-alerting context with virtually 100% accuracy.
>
> [KK09]

WuW recognition thus comprises of the following main components:
Detecting speech, spotting the WuW within speech and, finally, deciding whether the WuW was addressed to the system or occurred in conversational speech (*addressee detection*).

1.2 OBJECTIVE

In the present thesis, we will solely focus on the actual detection of the Wake-up Word (WuW), as Voice Activity Detection (VAD) and addressee detection are research fields of their own.

The first target of the thesis is to conduct a literature review of existing WuW detection approaches. In section 3.1 *Literature review*, we show that Hidden Markov Model (HMM) and Dynamic Time Warping (DTW) based approaches have been used and developed for speech recognition related tasks since the 1980s. Therefore, the expectable further performance gain is rather limited. On the other hand, neural networks have made huge progress in the past 20 years, proving their enormous potential when used as acoustic models in speech recognition [HDY+12]. One of

the most spectacular results with neural networks in the field of WuW detection is an approach presented by Chen et al. [CPS15], achieving an Equal Error Rate (EER) of 0.5%.
Due to its promising potential for the future, we chose to re-implement their template-based approach and evaluate its potential with extremely limited training resources ($\approx$ 2h of speech data). The target is to create a system which can easily be adapted for real-time applications in the future and yields satisfying error rates provided that enough training material is available.
To this end, a framework to train an LSTM network is built. Various experiments are conducted to better understand the training process. Furthermore, practical problems on which no information is given in the corresponding papers, such as the appropriate preparation of training data, are considered and experiments are conducted. The question of how to best exploit the available resources is investigated by training models on different selections of the available speech data. Theses trained models are evaluated on a custom recorded corpus regarding their EER.

1.3 REMARKS FOR THE READER

In the present thesis, the following citation convention is used: A citation before the end of a sentence refers to the source of this sentence; a citation at the end of a paragraph refers to the source of that paragraph. Citations on the right-hand side, below a paragraph, refer to the source of that entire section.

We, the autors of the thesis, worked together very closely. Therefore, we both contributed to all parts of the thesis in form of thoughts, tips and corrections. Nevertheless, the responsibility for actually writing certain chapters was divided between us and this division is briefly presented in the following.

Wolfgang Hrauda (WH) wrote the *Introduction* and then introduced *Recurrent Neural Networks* and *Wake-up word recognition*. The section about template-based systems in the *Literature review* was written by Clemens Vayda (CV).
The theory for using neural networks was covered by CV. He wrote the chapters *Data preparation* and *Neural network training* with the exception of the section about *Backpropagation through time*. The *Template matching* approach was described by WH.

In the practical part, WH wrote about the *Preliminary Experiments*. CV described *Training the LSTM network on WSJo* and presented the *Results for different output target sizes*, while WH presented the *Results for reduced average word length*.

Conclusions and Outlook and the *Acknowledgments* were written together.

2 RECURRENT NEURAL NETWORKS

In speech recognition, the words or phonemes contained in a sequence such as a spoken sentence should be recognized. One word or phoneme may span tens or hundreds of time steps in the sequence. Therefore, it is vital to capture and combine information from that time range.
With a simple feed-forward Deep Neural Network (DNN), the only way to achieve this is to stack information from multiple time frames and feed it into the network at once. The potential of this method is limited, though, as the network architecture is not designed for this kind of task.

2.1 RNN

Recurrent Neural Networks are explicitly designed to capture temporal context: The information learned from the input sequence is carried in the recurrent layer's hidden activations h. As a sequence is processed step-by-step, the hidden activations are remembered by feeding a weighted version of them back into the layer at each time step. Therefore, information from the past can be retained throughout time.

The signal flow through a generic one layer Recurrent Neural Network (RNN) as depicted in figure 1 (*top*) is given by

$$h_t = f\left(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h\right) \tag{2.1}$$

where f is a non-linear activation function.
At each time step, the new hidden state h_t is computed from two components: First, the input vector x_t, weighted by a weight matrix W_{xh}; and second, the previous states h_{t-1} of the layer, weighted by a recurrent update weight matrix W_{hh}. The second term contributes information remembered from the past, while the first term contributes information from the current frame. Together, they (theoretically) allow exploiting temporal information of the entire sequence seen so far.

The time evolution of the processing can be visualized by *unfolding* the network in time as shown in figure 1 (*bottom*). This way, the recurrent (feedback) notation is written out and becomes a representation of the network without any cyclic connections.
In recurrent networks, the error at each time step depends on the network's states of all previous time steps. Therefore, it is necessary to backpropagate the error through time. The unfolded graph allows for a clearer visualization of this procedure. Backpropagation through time is covered in more detail in section 5.3.

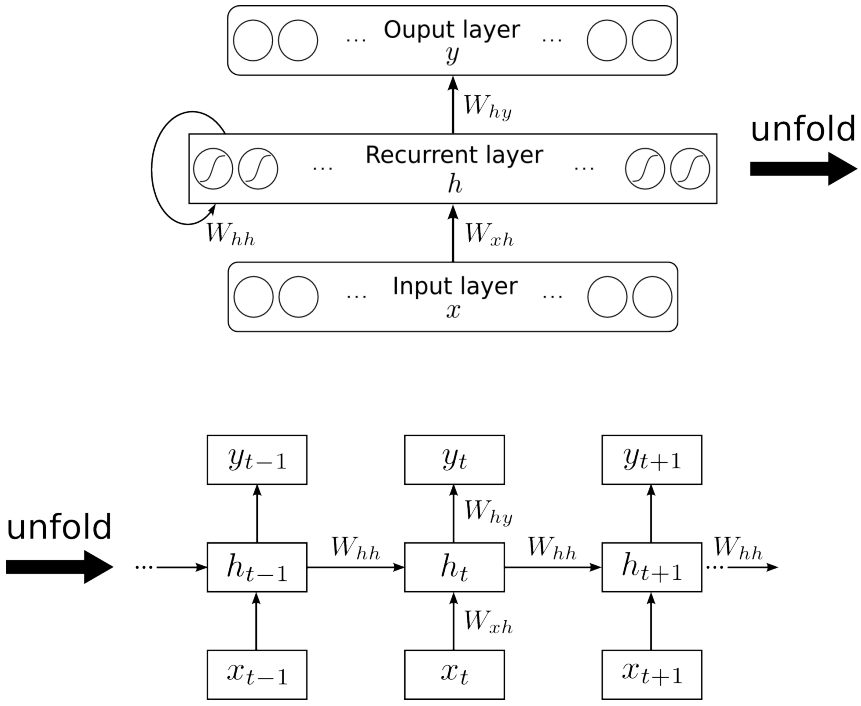


Figure 1: *Top*: Generic RNN network with a single recurrent hidden layer. Each block represents an entire layer with an arbitrary number of neurons. Note the feedback loop in the recurrent layer that allows to capture and combine information from past time steps.
Bottom: The same network unfolded; time steps evolve from left to right. At each time step, one vector from the input sequence $\mathbf{x}$ is fed into the network and an output vector $\mathbf{y}$ is obtained. The hidden states $\mathbf{h}$ are passed to the hidden layer for the next time step and to the output layer. Note that the weight matrices W do not change during one sequence. Bias vectors are omitted for the sake of clarity.

However, basic recurrent networks are problematic when an error is backpropagated through time during the training stage: The error tends to either explode or vanish. Learning from sequences is thus slow, exploits a limited temporal context or is actually impossible. In [HSH+97], Hochreiter and Schmidhuber proposed a novel RNN architecture to overcome this problem, called Long Short-Term Memory (LSTM).

[Gra08, p. 18-21]

2.2 LSTM

The core idea of the Long Short-Term Memory (LSTM) architecture is to set the recurrent update weight to 1. This assures that the error can flow through the network without modification during backpropagation through time, which eliminates the vanishing gradient problem.
In addition, each cell has an input gate i_t controlling when the internal memory can be updated by the cell input x_t. This prevents irrelevant information from being remembered. Output gates are used to control when the memory content is passed to the next layer and the other LSTM blocks in the same layer (via the output state h_t), which avoids an unwanted perturbation of other cells' contents. The gate mechanism enables LSTM units to store information over a longer period of time and handle long-range dependencies, which is a weakness of the standard RNN architecture. [HSH+97]
With continual sequences consisting of several subsequences, it might be necessary to reset the cell memory state at appropriate times. Therefore, Gers et al. introduced forget gates f_t that enable the "LSTM cell to learn to reset itself" . [GC00]
The following set of equations describes the basic signal flow in one LSTM cell, while figure 2 visualizes it:

$$i_t = \sigma\left(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + W_{ci} \cdot c_{t-1} + b_i\right) \tag{2.2}$$
$$f_t = \sigma\left(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + W_{cf} \cdot c_{t-1} + b_f\right) \tag{2.3}$$
$$o_t = \sigma\left(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + W_{co} \cdot c_t + b_o\right) \tag{2.4}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh\left(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c\right) \tag{2.5}$$
$$h_t = o_t \cdot \tanh(c_t). \tag{2.6}$$

[GMH13]

INPUTS The following variables are used as inputs to compute a new candidate value for the memory cell (second summand of equation 2.5) and to compute the activations of the gates (equations 2.2 to 2.4):
The input vector x_t typically consists of the activations from the previous layer. The output states of the previous time-step h_{t-1} are also used as an input. However, if the output gate is closed, h is close to zero and the gates have no access to information about the current state of the cell. Therefore, Gers et al. [GS00] later

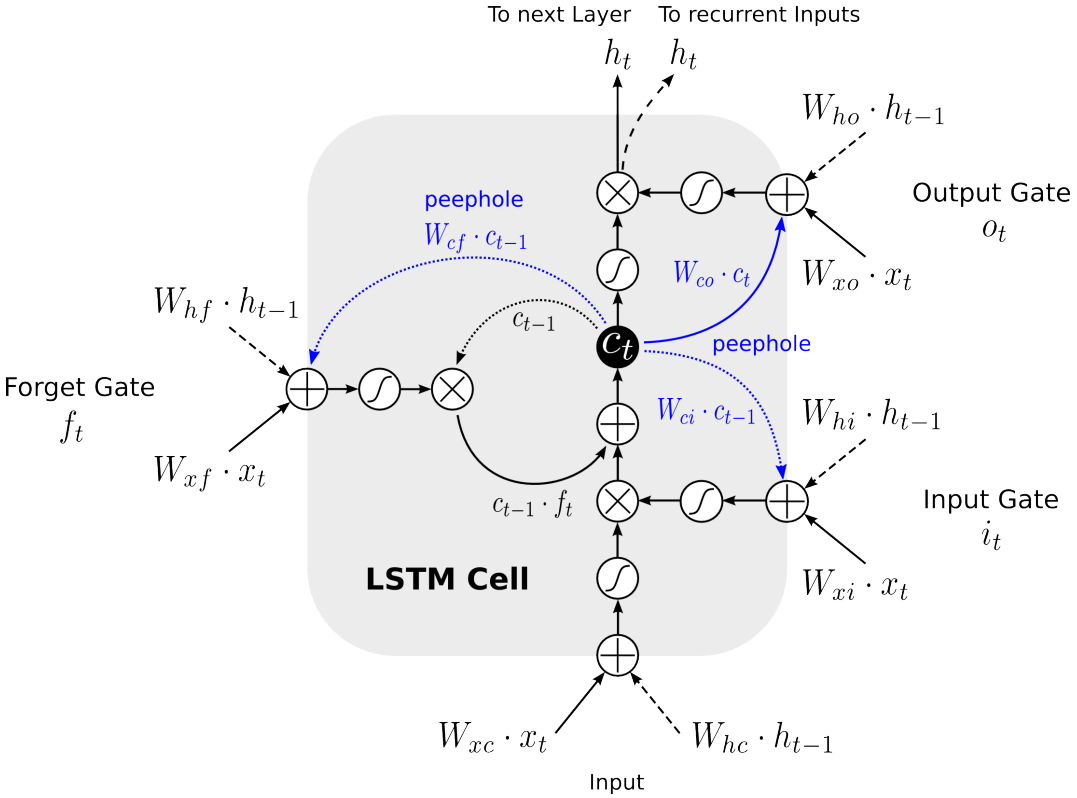


Figure 2: Structure of a single LSTM memory cell.

proposed to add "peephole" connections to allow gate states to be influenced by the actual memory cell value c. For the output gate, the current cell value c_t is used, whereas for all other gates, the previous value c_{t-1} is used.
These input variables are weighted with weight matrices W, biased and then squashed with a non-linear function: A sigmoid function in the case of the gates and a tanh function for the cell state candidate value.
Storing, passing and forgetting information (i. e. the gate states) thus depends on the current input, previous cell state and previous output state.

CELL UPDATE The new cell value c_t (see equation 2.5) consists of two summands: Its previous value c_{t-1} and a new candidate value computed from the inputs as described above.
The first is the recurrent update with weight 1, allowing to retain the memory content. Hereby, the forget gate f_t controls how much of c_{t-1} is retained or forgotten and also allows resetting the memory.
For the new candidate value, the input gate i_t controls how much of it is allowed to enter the memory cell.
In this way, the memory cell can either store learned information or acquire new information, depending on the input variables that influence the gate states.

OUTPUT STATE For computation of the output h_t (see equation 2.6), the cell value c_t is squashed by the non-linear tanh function and then multiplied by the output gate o_t to control whether the cell passes its stored information on to the next layer, depending on the current state of the input variables.

3

WAKE-UP WORD DETECTION

The task of WuW detection as a sub-field of speech recognition was already introduced in 1.1 *Motivation*. Its main goal is to *wake up* machines and put them into a state where they expect further commands to be spoken by the user.
Such systems require different components like VAD and optionally addressee detection. However, as explained in 1.2 *Objective*, we will focus on the core task of actually spotting the WuW in speech.
Regarding the technical approaches, this task is very similar to the more common KeyWord Spotting (KWS)[1] problem. Therefore, it is important to note that the terms WuW and keyword are mostly interchangeable for the remainder of this thesis.

In this chapter, we first give a brief introduction on assessing the performance of WuW recognizers. Then, an overview of different approaches to WuW spotting is given. Finally, the paper we chose as the conceptual basis of this thesis is presented in more detail.

The performance of WuW recognizers is often measured by two rates. In conjunction, they characterize the behavior of the system:

FALSE ALARM RATE A *false positive* is defined as the incorrect detection of a keyword, i. e. the recognizer detects a keyword, even tough none has occurred. The number of false positives divided by the total number of examples gives the False Alarm (FA) rate.
For WuW applications, a false alarm means that the system is activated even though the user did not intend to do so. This can be very disturbing or lead to an execution of wrong commands. Given that a WuW recognizer is usually active in the background for a long period of time, the FA rate should be kept as low as possible.

FALSE REJECTION RATE A *false negative* is defined by missing the detection of a keyword, i. e. a keyword occurs, but the recognizer does not detect it. The number of false negatives divided by the total number of examples gives the False Reject (FR) rate.
For WuW applications, a false rejection means that the user wants to activate the system, but the system does not wake up. In such a case, users will tend to repeat the keyword. Therefore, the FR rate is usually not as critical as the FA rate, but should still be low enough to ensure a satisfying experience for the user.

1 Keyword spotting: Scanning huge amounts of data for the occurrence of a certain word or phrase that is of special interest for some reason (e. g. news programs might be scanned for a politician's name).

The optimal trade-off between FA and FR rate depends on the type of application and can usually be tuned for a given system. For a performance measure that is independent of this trade-off, the system can be evaluated with settings that result in identical FA and FR rates, which is then called equal error rate.

Instead of FA and FR rates, recall and precision can also be given as performance assessment figures. A high recall corresponds to a low FR rate, whereas a high precision corresponds to a low FA rate.

3.1 LITERATURE REVIEW

The following gives an overview of the three most common approaches to the actual detection of a given keyword. Some of the most relevant examples of each approach are discussed and contrasted.

3.1.1 *HMM-based systems*

The most common approach to keyword spotting and WuW detection is based on HMM, as they have been successfully applied to speech recognition since the 80s [Lev83].

Each HMM corresponds to a certain unit of speech, such as a triphone or a single phone and consists of several states (typically 3-10). One state models a short stationary part of the speech. Each state uses Gaussian mixtures to represent their typical features in a so-called acoustic model. Both the means and variances of the Gaussians and the transition probabilities between the states of the HMM are learned during the training process. This usually requires a large amount of transcribed speech data, which is not always easily available.

Most methods use a keyword model to describe the desired word, while a garbage model describes anything else. During system run-time, the acoustic model generates posterior probabilities for each possible speech unit given the input feature vector. A Viterbi or similar algorithm then finds the most likely sequence of HMM states (each state corresponding to one speech unit). The operating point of the system (i. e. the trade-off false alarm *vs.* false rejection ratio) can be tuned by the prior probabilities of the keyword *vs.* garbage model.

HMM-based WuW implementations were presented for personal communication devices in [BGA00] and with enhanced voice activity detecion for a meeting room scenario in [CKSK11]. The former report a FA rate of 0.55% and a FR rate of 15% while the latter report an equal error rate of 3%. In [LCYK09], it was shown that an HMM-based system similar to [BGA00], achieves an equal error rate of about 10% and outperforms a DTW-based system by approx. 4%, even when the DTW system is augmented with codebook-based keyword and garbage models. However, the computational cost of the HMM system was about four times higher.

In [BFHC03], frame-wise posterior probabilities are averaged to obtain a word-level confidence measure that an utterance contains the keyword. The results of different averaging methods are combined using a Support Vector Machine (SVM), which

transforms data into a high-dimensional space for easier separation, i. e. classification. The idea was to exploit complementarities of the different averaging methods, but the performance gain was very modest. However, a similar idea is used in one of the most sophisticated HMM-based WuW detection systems presented in [KK09], which utilizes multiple feature streams and a triple-scoring method followed by a nine-dimensional SVM classification. They report an astonishing equal error rate of 0.1%. However, the essential point for performance gain is the proprietary triple-scoring method which is not publicly available.

In [KGB09], a discriminative approach for keyword spotting was shown to outperform a HMM baseline system, similar to the one presented in [BGA00], on two speech corpora including the TIMIT corpus [GLF+93]. Instead of determining the most likely sub-unit sequence, it aims at directly maximizing the area under the Receiver Operating Characteristic (ROC) curve, which is an important performance measure for keyword spotters.

3.1.2 *Template-based systems*

Another approach to keyword spotting are template based methods. One or more spoken utterances of the keyword represented in feature domain are used as a template, which is then compared against any test audio segment to make detection decisions.
A main difficulty for template based methods is to get a fixed length representation of the time varying speech input. The most common way to perform this template comparison in speech recognition is called DTW. This method has been used for speech recognition since the 80s as proposed in [MR81]. It aligns the sequence of the parametrized input keyword in time to the spoken test utterance and outputs a similarity score. Drawbacks are a high computational complexity and often an insufficient modeling of the word duration.
The simplest way is to use Mel-Frequency Cepstral Coefficients (MFCC)s directly as input to a DTW-algorithm to output a minimum distance score as it was shown in [BRS11] but with limited success. In [ZHP14], a quite high precision of almost 100% was obtained at the cost of a lower recall rate (between 30% and 80% depending on the background noise) with a DTW-based system for an emergency use-case on mobile devices. In this paper, they showed as well that the Euclidean Distance (ED) or the Cross-Correlation (CC) are a suitable approach for WuW detection and that a combination of those measures improves the performance.
Instead of directly using MFCCs as input to a DTW-algorithm, a 50 component Gaussian Mixture Model (GMM) is trained in [ZG09]. This is done on all the training data without any transcription information. The trained GMM then outputs a posteriorgram vector for each speech frame. These vectors are then compared by applying a segmental DTW between the keyword samples and the test utterances. They report that the performance increases if more than one keyword is provided and that keywords with more syllables tend to have better performances. However, none of their reported results is below 15% equal error rate.

3.1.3 *Neural network based systems*

Due to the advances of neural networks from the 1990s onwards [HDY+12], the possibility to replace a GMM based acoustic model by neural networks became attractive. In general, DNN systems can serve as acoustic models due to their ability to map same/similar words or phonemes into similar parts of the network's hidden activations space.

HMM based systems with a DNN as an acoustic model are often referred to as *hybrid* systems. E.g., the baseline system presented in [CPH14] uses a DNN to compute the state densities for a HMM keyword-filler model with reasonable success.
Other variants of similar combinations have been investigated as well: Phone posterior estimates from an LSTM plus MFCC features serve as a "mutli-stream" input to a HMM in [WMSS11]. Two years later, Wöllmer et al. [WSR13] used a bidirectional LSTM network[2] with Connectionist Temporal Classification (CTC)[3], along with dynamic bayesian networks (a generalization of HMM) to outperform a pure HMM system.

One of the first successful non-hybrid systems was proposed by Fernandez et al. in [FGS07]. They also used a bidirectional LSTM network with CTC to outperform an HMM/GMM system on a keyword spotting task.

Chen et al. [CPH14] took it one step further by directly using the output neurons of a DNN to detect a keyword. Therefore, there is an output neuron for each word in a keyword phrase (*okay google* in figure 3) plus one neuron for all non-keywords (*filler*). Figure 3 gives an overview of the main components that are typical for this kind of approach: Log-filterbank energy features are extracted and fed into the DNN. After training, the network is capable of generating frame posteriors for each of the keywords and all the other words (*filler*). This way, simplistic posterior handling is sufficient to compute a confidence score that the keyword phrase was detected. The system outperformed the above mentioned hybrid DNN/HMM baseline system.

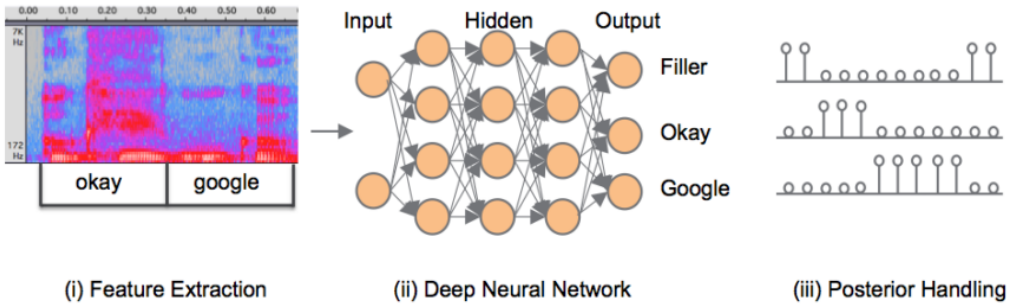


Figure 3: Components of the Google Deep KWS system [CPH14].

2 Bidirectional recurrent networks process sequences forward and backward in order to exploit future and past context.

3 In [GFGS06], Graves et al. introduced CTC, a technique that allows training neural networks even when transcriptions of the speech data are only available without time alignments.

Sainath and Parada [SP15] even improved this performance by using a Convolutional Neural Network (CNN) architecture. Compared to DNNs, CNNs are better at modeling the local correlations in time and frequency found in speech signals. Furthermore, they are better suited to deal with variances between different speaking styles.
While the networks in the latter two approaches need to be trained on specific keywords, the proposal in [CPS15] only requires few templates of the keyword that can be recorded after training. To this end, an acoustic model is generated by training an LSTM network to be able to recognize 15K different whole words. The core idea is to use the hidden states of this acoustic model to represent the *keyword* and any *live speech data* as two fixed-length vectors. The keyword is then detected by observing the similarity between these vectors. This system outperforms even the CNN approach in [SP15] and will be presented in more detail in the following section.

The last three shown approaches only require simple forward-pass computations of the network at runtime; they do not require any computationally intensive operations such as Viterbi decoding or time alignment via DTW. Compared to HMM/DTW methods, the computational complexity of neural network based systems lies in the training process rather than at runtime.

3.2 QUERY-BY-EXAMPLE KEYWORD SPOTTING USING AN LSTM NETWORK

As far as we know, the LSTM based system introduced by Chen et al. in [CPS15] is the best approach in WuW detection with neural networks. It achieves an equal error rate of 0.5% in clean conditions. As mentioned in chapter 1.2 *Objective*, the idea of this thesis is to implement the basic concepts of their work. Hence we will now briefly explain their key concepts:

BUILDING THE ACOUSTIC MODEL For training, the network consists of two hidden LSTM layers with 128 cells each and a softmax output layer with one neuron for each word to be learned, known as one-hot encoding. The network is trained to be able to recognize 15000 different whole words. It is worth noting that the modeling of entire words as opposed to phonemes is crucial to the success of this approach. The authors report performance degradation of more than an order of magnitude if phonemes are modeled.
When input speech features are fed into a trained acoustic model, the states of the second hidden layer are representative of the spoken sounds.

TEMPLATE MATCHING The ability of the hidden states to capture the characteristics of spoken words allows using the acoustic model as a *feature extractor* to create a keyword model.
As shown in figure 4 (*enrollment* phase), the log-filterbank energy features of a keyword are fed into the feature extractor. The hidden states of all time frames are concatenated to form a vector that is representative of the keyword. This vector is called a *template*.

As LSTM cells can store significant information in their memory, their hidden states also represent the past. Therefore, these vectors can be resized to a fixed-length, i. e. truncated (or zero-padded), without losing relevant information. For increased robustness, the features extracted from several recordings of the same keyword can be resized to matching lengths and then averaged to form a *keyword model*.
At *runtime*, a hidden state vector of the same length is generated from live speech. The similarity between this vector and the template allows detecting whether the keyword was uttered or not.

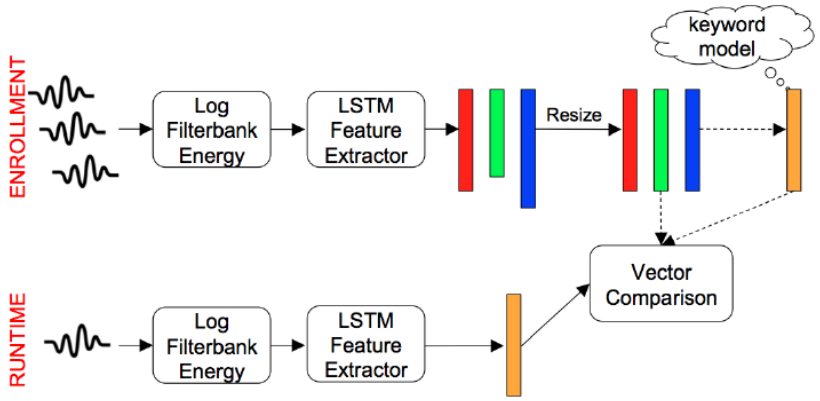


Figure 4: Template matching in the Google LSTM KWS system: Creating a keyword model during *enrollment*; comparing live speech to the keyword model at *runtime* [CPS15].

RESULTS At an operating point of 0.5% False Alarm (FA) rate, the system achieves 0.5% False Reject (FR) rate in clean conditions and 2% FR rate in the presence of babble noise. When the original enrollment is corrupted by background noise from a café, performance is degraded significantly, though the proposed system is still far more robust than the hybrid neural network and DTW baseline systems.

[CPS15]

4

DATA-PREPARATION

In this chapter we shortly describe the used programing framework. Then we will give an overview of how data has to be prepared in order to serve as input for the Neural Network (NN)-model. In the end we describe in detail how features are generated from the speech signal.

4.1 FRAMEWORK

We used the programming language Python for the whole implementation. The code of the neural network model is based on the library Theano first introduced in [BBB+10]. Theano brings some important advantages for training and building neural networks, amongst other things automatic differentiation as well as efficient and transparent use of the GPU, which is necessary for fast execution [web16].
We used a Theano-based framework called Blocks and Fuel, which was developed at the University of Montreal to build neural network models. Blocks simplifies the prototyping and the training process of neural networks. It provides many components, called bricks, for building and training neural networks.
Fuel provides data processing routines and a standard data interface for Blocks. The standard data format utilized is HDF5 (Hierarchical Data Format 5). It was designed to cope with large amounts of data and allows meta data and annotations to be added to the datasets. In particular, all the data for training, evaluation and testing is stored as a single HDF5 file. The annotations are used in a fixed way to label the data sources, e.g. *features*, *labels* and *mask*.
Fuel also provides several schemes to iterate over the data during the training processes.

[Uni14a, Uni14b, O'D15]

4.2 THE DATA-PREPARATION PROCESS

The data-preparation process consists of several preprocessing steps which have to be done to prepare the speech data in order to train the NN-model.
In Figure 5, the data-preparation process is illustrated: **Features** are calculated from the speech signals, **labels** are generated from the transcriptions and the word-alignments ,and a **mask** is generated for computational purposes. Features, labels and the mask are called data-sources. In addition, the data-sources are split into training-, validation- and a test-data.

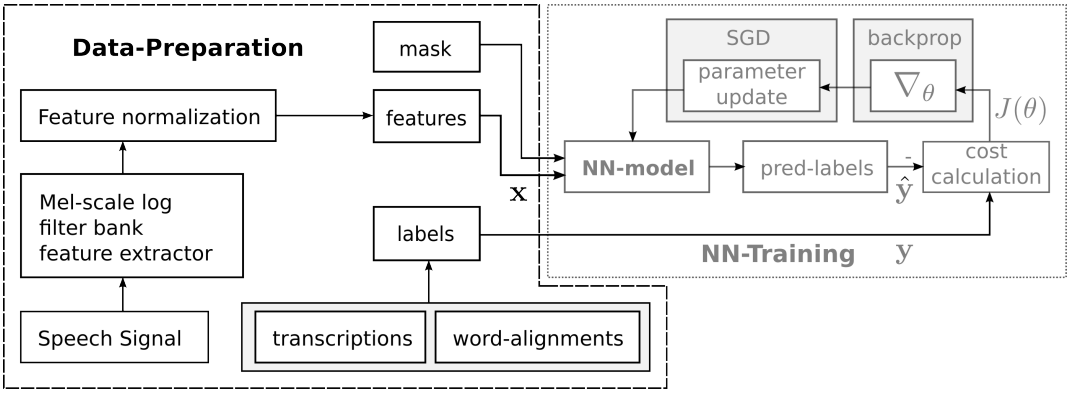


Figure 5: The data-preparation process. Features and labels are calculated out of the raw data and a mask is generated. Then the data can be used to train a Neural Network.

FEATURES 40-dimensional Mel-scale log-filter bank features are calculated from the speech signal for each time-frame and an utterance level mean-variance normalization is performed (see section 4.3).

LABELS As we do supervised learning, labels have to be provided for each time step in order to be able to calculate an error between the predicted labels $\hat{y}$ and the true labels y. Therefore a dictionary is generated that contains the vocabulary of the speech corpus and every word of the dictionary is mapped to a binary representation. Knowing this binary-representation and the given word-alignments for each word enables us to assign a label (or target) to each speech-frame of an utterance.

MASK At every time-step of the training process, the LSTM-network is provided with a feature vector as input and the corresponding label as target. Batch-processing requires all processed sequences to be equal in length. So all the utterances as well as the labels are zero-padded to the length of the longest utterance. The mask is generated to tell the LSTM network for every utterance in which time-step the zero padding starts. According to the mask, the LSTM network will stop updating its parameters (it stops learning) from the time-step on where the last frame of real data was located.

As a last step, the features, the labels and the mask for the training-, evaluation- and test-set are stored in a single HDF5-file. This has to be done according to the specifications of Fuel to guarantee a flawless interaction of Fuel and Blocks. During training, we use the *Shuffled Scheme* provided by Fuel to iterate over all the examples in the dataset in shuffled batches.

4.3 SPEECH FEATURES FOR NEURAL NETWORKS

The main goal of feature extraction is to get a representation that captures the required characteristics of the input data.
For conventional GMM based systems, Mel-Frequency Cepstral Coefficients (MFCC)s are widely used. In NN however, Mel-scale log-filter bank features are commonly used as they lead to performance improvements compared to the standard MFCC features [LYHG12].
The generation of this feature type is described in this section.

4.3.1 *Mel-scale log-filter bank features*

The calculation process of Mel-scale log filter bank features (or simply log-filter bank energies) is motivated by how humans perceive speech signals: Pitch perception happens in the cochlea on the basilar membrane. Every area of the membrane responds to different frequencies resulting in a frequency to place mapping. This mapping is non-linear in so far as lower pitches have a better spatial resolution on the basilar membrane than higher ones. [KK14]
To capture this non-linear pitch perception of the human auditory system the Mel-scale was developed. It is given by

$$M(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right). \tag{4.7}$$

[FG01]

Experiments showed that loudness is also perceived in a non-linear manner. It can be approximated on the logarithmic scale. Figure 6 shows the corresponding processing steps.

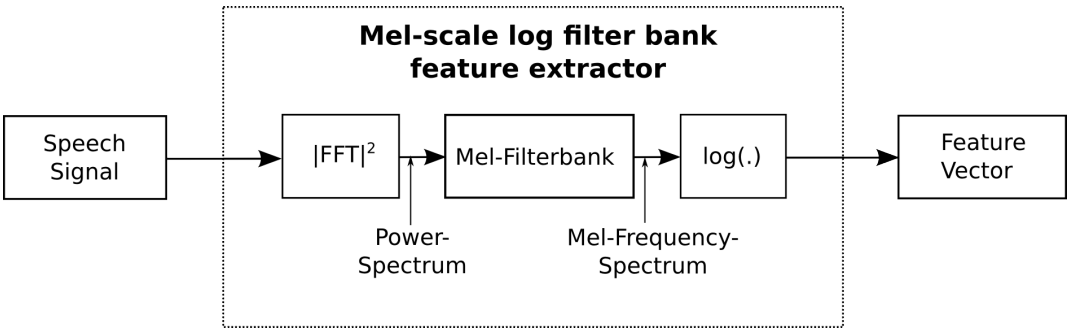


Figure 6: Processing steps to obtain log-filter bank energies. Every block mimics a certain characteristic of the human ear.

FREQUENCY TO PLACE MAPPING First, a Fast Fourier Transform (FFT) transforms the speech signal into the frequency domain and the power spectrum is obtained.

NON LINEAR-PITCH PERCEPTION Second, the warped power spectrum is filtered by N triangular band-pass filters (figure 7). Each of the filters has a different bandwidth on the frequency axis but an equal bandwidth on the Mel-scale. The width of the filters is calculated using equation 4.7. In the end, one value per filter is obtained corresponding to the energy in that frequency region .

LOUDNESS PERCEPTION The third computation step simply takes the logarithm of each of these energy values.

For speech recognition tasks, 40 Mel-filters is a common number of filters to use [SKMR13].

[KL14]

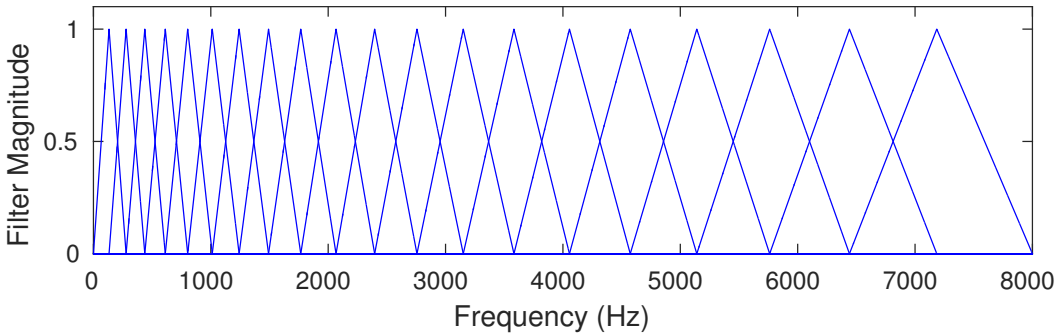


Figure 7: filter bank with triangular bandpass filters to compute the filter bank energies.

4.3.2 *Normalization of features*

> "Feature normalization is critical in neural network training to achieve good convergence in training. As discussed in [Fal13], when features are not centered around zero, network updates will be biased towards a particular direction and this will slow down learning. The paper even discusses an extreme case when all inputs into a layer are positive. This causes all weights to increase or decrease together for a given input, and thus the weight vector can only change direction by zigzagging which is extremely slow."
>
> [SKMR13]

The extreme case mentioned above is the exact problem of our application. We calculate the log-filter bank energies from the magnitude of the power spectrum, so that without normalizing all inputs are in the positive range.

In [Fal13] they propose to shift the input in such a way that the average of the inputs over the training set is close to zero. This can be done by global mean-variance normalization: Let m_i be the logarithmic output of the i-th Mel-filter, μ_i the mean and σ_i the variance over the training set corresponding to this Mel-filter value. Then the normalized Mel-filter value n_i is defined as

$$n_i = \frac{m_i - \mu_i}{\sigma_i}. \tag{4.8}$$

Normalizing all the input data in that way ensures fast convergence.

5 NN-TRAINING

This chapter explains all the necessary parts to understand how a training process of a NN is structured.
This is illustrated in figure 8: At first, a NN-model is designed (section 5.1). Then the data sources are fed batch-wise into the NN. For each time-step, the NN produces a predicted label $\hat{\mathbf{y}}$. At the end of each utterance, the predicted labels $\hat{\mathbf{y}}$ are compared to the true labels $\mathbf{y}$ and the cost is calculated. Then the error is back-propagated through time (section 5.3). Using a training algorithm like Stochastic Gradient Descent (SGD), the network parameters are updated in the direction of the negative gradient (section 5.2). Then the next batch of utterances is fed into the NN and the process is repeated until all the data was seen once. This is called epoch. To further optimize the network parameters, tens or hundreds of epochs are done.

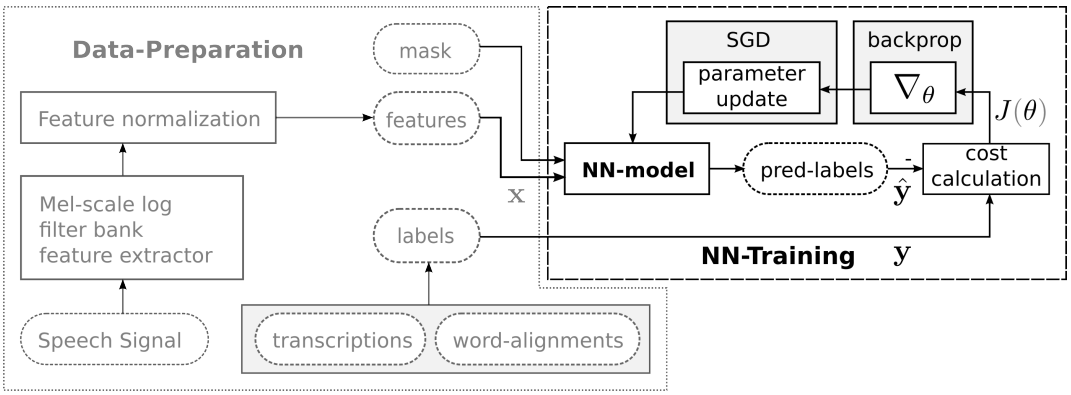


Figure 8: NN-Training: Data is fed into the NN-model and a cost is calculated. Then the error is back-propagated through time. The network parameters are updated in the direction of the negative gradient via SGD.

5.1 LSTM-MODEL

Using Blocks, we built a LSTM-Model with two hidden layers, each with 128 LSTM-cells as shown in figure 9. The linear input layer maps the 40-dimensional input features $\mathbf{x}$ to $\tilde{\mathbf{x}}$, which serves as an input to the 128 hidden cells of the first LSTM layer. The linear output layer maps the output $\mathbf{h^2}$ of the 128 hidden cells of the second layer to the size of the output target vector $\hat{\mathbf{y}}$. Then a softmax is applied to the output layer to represent each output value as a probability.
As we use one-hot encoding, each output neuron corresponds with one word in the vocabulary. Training the model with a huge vocabulary (e.g. 2 k words) results in a large output layer. This leads to difficulties in the training process regarding memory consumption and duration.

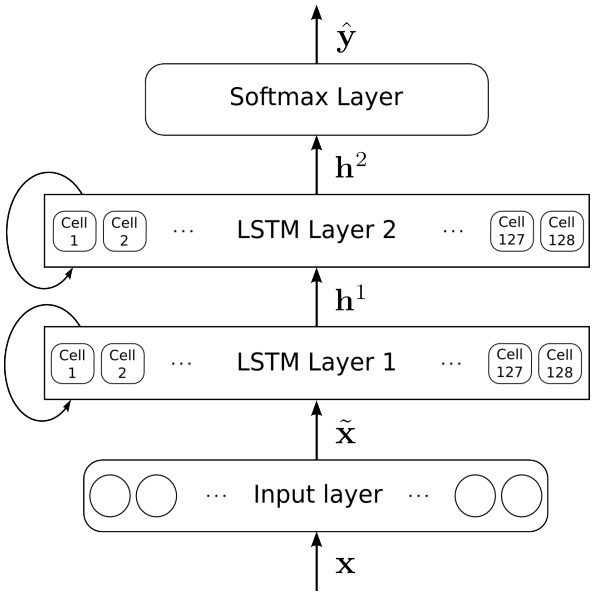


Figure 9: LSTM-Model with 2 hidden layers, 128 LSTM-cells, and a Softmax Layer as output layer. The bent arrows on the left side of the hidden layers illustrate the temporal recurrence.

We train the network in shuffled mini-batches on the GPU using *Adam*, a special version of SGD (see section 5.2) and categorical cross-entropy, the matching cost function of the softmax output layer. Both algorithms are provided by Blocks.
The categorical cross-entropy is defined as

$$J(\theta) = H(\mathbf{y}, \hat{\mathbf{y}}(\theta)) = -\sum_i y_i \cdot \log(\hat{y}_i(\theta)) \tag{5.9}$$

where i covers the range of the number of output targets.

5.2 STOCHASTIC GRADIENT DESCENT

The general goal of learning is to minimize a kind of *loss* function $J(\theta)$ with parameter θ. In practical applications, one has to minimize the cost over an entire training set with many observations x and a probability distribution over the observations $dp(x)$ resulting in a loss function $J(x, \theta)$.
Learning means finding the optimal parameters θ that minimize the expectation value of the loss function

$$C(\theta) = \mathbb{E}(J(x, \theta)) = \int J(x, \theta) dp(x). \tag{5.10}$$

As the training set is finite, the distribution $dP(x)$ is discrete,

$$C(\theta) = \frac{1}{N} \sum_{i=1}^{N} J(x_i, \theta) \tag{5.11}$$

The simplest approach to minimize the loss function is to update the parameters by going a small step in the negative direction of the gradient known as *gradient descent*,

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta C(\theta_t) \tag{5.12}$$

$$= \theta_t - \alpha \cdot \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta J(x_i, \theta) \tag{5.13}$$

where α is the learning rate. This method however has weaknesses in terms of practical application in neural networks. One of them is that each step requires the entire training set to be processed in order to evaluate $\nabla_\theta C(\theta)$.

SGD overcomes this weaknesses by updating to the parameters based on one datapoint or on the average of more datapoints (called batches) at a time

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta J(x, \theta_t). \tag{5.14}$$

The update is then repeated by going through all datapoints or batches of datapoints in a dataset. This small change in the algorithm leads to several advantages:

- The algorithm converges much faster when the examples are redundant. In the extreme case of duplicating every data point, the total gradient algorithm doubles its computational effort, whereas the batch gradient algorithm is unaffected.
- It is possible to escape local minima because of the random behavior of the SGD-algorithm
- The loss function can converge even if it is not differentiable everywhere.

[Bis06, Bot91]

5.2.1 *Stochastic Gradient Descent with Adam*

The name Adam is an abbreviation for adaptive moment estimation. It was first introduced by Kingma in [KB14] and is one of the best SGD-based-algorithms at the moment.
In particular it is an algorithm for gradient-based optimization of stochastic objective functions. Adam is a progression and fusion of AdaGrad and RMSProp. AdaGrad has the advantage to work well with sparse gradients and RMSProp can deal with non-stationary objectives. Adam combines both of these properties resulting in a computationally efficient algorithm with fast convergence. It is designed to work well with large datasets and in high-dimensional parameter spaces. Also rescaling of the gradient does not influence the magnitudes of parameter updates.

Let $J(\theta)$ be a noisy objective function that is differentiable w.r.t its parameters θ, and let α be the step-size, then the algorithm works as follows:

- Get gradients w.r.t. stochastic objective at timestep t: $G_t = \nabla_\theta J_t(\theta_{t-1})$
- Update biased first and second moment estimate of the gradient (m_t, v_t)
- Compute bias-corrected first and second moment estimate $(\hat{m}_t, \hat{v}_t)$
- Update parameters: $\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t}$

The goal of the algorithm is to minimize the expectation value of the noisy objective function $\mathbb{E}[f(\theta)]$ w.r.t to its parameters. The stochasticity in our case comes from the evaluation of random subsamples of data-points which we called batches. The parameters m_t, v_t are moving averages of the gradient with exponential decay rates. Thus, they are estimates of the mean and the variance of the gradient and then used to update the parameters.
As they are initialized with zeros, they get bias-corrected, which is especially necessary in the situation of sparse gradients.
An important part of Adam's algorithm is its update rule. The effective step taken in parameter space is $\Delta_t = \alpha_t \cdot \hat{m}_t / \sqrt{\hat{v}_t}$, i. e. α is multiplied by the ratio of the bias-corrected first and second moment estimates. Except for the situation of a severe case of sparsity, this factor is always smaller than one. Therefore, the effective step in parameter space is approximately bounded by the step-size hyperparameter i.e., $|\Delta| \lessapprox \alpha$.
The advantage of this method is that the effective step size in parameter space Δ_t gets smaller when it comes closer to an optimum where the uncertainty of direction is bigger (because $\hat{m}_t / \sqrt{\hat{v}_t}$ gets smaller). This is a desired property and a form of automatic annealing (viz."to gradually lower").

[KB14]

5.3 BACKPROPAGATION THROUGH TIME

The previous section 5.2 showed that Stochastic Gradient Descent requires the calculation of the gradient of the loss function $J(x, \theta)$ w.r.t. its parameters θ. As explained before, this gradient is used to update the parameters so as to follow the steepest descent of the loss function.
Simple *error backpropagation* can be used to obtain this gradient in simple neural networks [Bis06, p. 241-245]. However, for RNNs, this concept has to be extended to *backpropagation through time* due to the recurrence in time [ZW95, Gra08].

In an RNN, the loss or error $J(x, \theta)$ comprises of the sum over $J_t(x, \theta)$ at all time steps. Therefore, the gradient $\nabla_\theta J$ over an entire sequence is obtained by summing over the gradients $\nabla_\theta J_t$ at each time step. $J_t(x, \theta)$ depends on the input observations of all previous time steps and the parameters θ. Note that the parameters θ are the same for all time steps.

When computing the gradient at time step t with respect to a certain parameter e. g. W_{xh}, the influence of this parameter on the current error needs to be taken into account *for all previous time steps*. In figure 10, the influence of W_{xh} at time step $t-1$ on the current error J_t is depicted as a red path. This influence is expressed as

$$\frac{\delta J_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta W_{xh}}$$

by "following the path" of the error through the network and applying the chain rule in Leibniz notation.

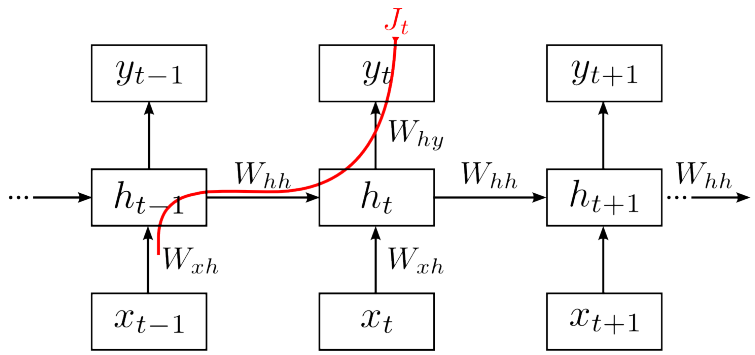


Figure 10: Error flow through a generic RNN: The influence of W_{xh} at time step $\tau = t-1$ on the current gradient w.r.t. W_{xh} is shown as a red path. The gradient is obtained by backpropagating the error through time according to the chain rule.

In order to obtain the entire gradient at the current time step w.r.t. W_{xh}, the paths to all previous time steps have to be followed and derived according to the chain rule. The gradient is then obtained by summing over these paths:

$$\frac{\delta J_t}{\delta W_{xh}} = \sum_{\tau=1}^{t} \frac{\delta J_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_t}{\delta h_\tau}\frac{\delta h_\tau}{\delta W_{xh}} = \tag{5.15}$$

$$= \underbrace{\frac{\delta J_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_t}{\delta W_{xh}}}_{\tau=t} + \underbrace{\frac{\delta J_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta W_{xh}}}_{\tau=t-1} +$$

$$+ \underbrace{\frac{\delta J_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_t}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta h_{t-2}}\frac{\delta h_{t-2}}{\delta W_{xh}}}_{\tau=t-2} + \dots$$

(5.16)

The derivation of the gradient in a sequence of length S is obtained by summing over the gradients at each time step. For the gradient w.r.t W_{xh}, we obtain:

$$\begin{aligned}\frac{\delta J}{\delta W_{xh}} &= \sum_{t=1}^{S} \frac{\delta J_t}{\delta W_{xh}} \\ &= \sum_{t=1}^{S} \sum_{\tau=1}^{t} \frac{\delta J_t}{\delta y_t} \frac{\delta y_t}{\delta h_t} \frac{\delta h_t}{\delta h_\tau} \frac{\delta h_\tau}{\delta W_{xh}}.\end{aligned} \tag{5.17}$$

[dF15]

For updating all the parameters in a network, this gradient has to be calculated w.r.t. to each parameter.

In the case of LSTM networks, the same principle applies, but the actual derivation of the signal flow in one LSTM cell is more complex than in a simple RNN.

5.4 MONITORING OF THE TRAINING PROCESS

So far, we have described how the input data is fed into the NN and how parameters are updated in a process called training. This section explains how monitoring scores are obtained to supervise the training process and prevent overfitting.

5.4.1 *Calculation of monitoring scores*

In the training process parameters are updated after every batch. After this has been completed for all the data, we want to know the progress made during the epoch. This is achieved by using the network in feed-forward mode only. Again, all the training data is fed through the network and an error score is computed for every batch. All these error scores are averaged to obtain a final monitoring score. In the same way the monitoring scores are calculated for the validation set. For the next epoch the batches are reshuffled and the process of training and monitoring is repeated.
Normally, the cost serves as a monitoring score but it is also possible to create user defined monitoring scores such as the Word Error Rate (WER).

We record the monitoring scores for the training and validation set in a log file after every epoch. In addition, performance is measured on a test set after the final epoch.

5.4.2 *Definition of the word error rate*

As we train on whole word output targets, it is useful to measure performance in terms of a Word Error Rate (WER). We use a frame-based WER that is obtained as follows: For each prediction $\hat{\mathbf{y}}$, the index of the prediction with the highest

probability $\hat{y}_{max}$ is compared to the "true" word index given by the data labels. A false guess (FG) means that the index was not predicted correctly, i. e. the word was not recognized correctly. A true guess (TG) corresponds to a matching index. The WER for one utterance is obtained by summing up all the frame-wise FG and dividing this sum by the total number of guesses:

$$\text{WER} = \frac{\sum \text{FG}}{\sum \text{FG} + \sum \text{TG}} \tag{5.18}$$

5.4.3 *Overfitting*

Our LSTM-Model consists of approximately 150 k parameters which results in an almost infinite number of possible parameter combinations. In other words, our model is incredibly flexible. So if we train it long enough, the LSTM-Model has the ability to learn the training data by heart. This is known as overfitting.
Overfitting is not desired as it leads to performance losses when predicting new, unseen data, so the generalization performance decreases. Typically, the training set performance still increases while validation set performance decreases as it can be seen in figure 11. One solution of this problem is called Early Stopping, that is to stop training when the network performs best on the validation set.
In practical applications, training is done for many epochs and the network parameters are saved if validation set performance increases. In the end, the parameters of the best epoch are used.

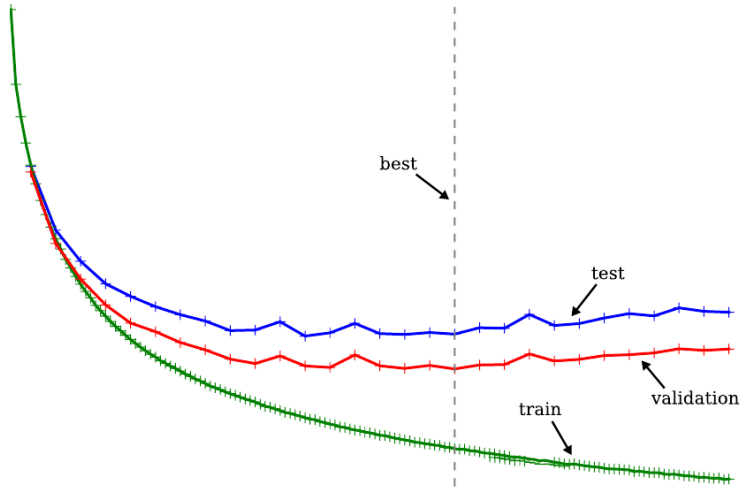


Figure 11: Overfitting: Training set performance still increases while validation set performance decreases. Early stopping solves the problem by saving the parameters of the epoch where the network performed best on the validation set. [Grao8]

6 TEMPLATE MATCHING

The detection of a Wake-up Word (WuW) is the main goal of this thesis as pointed out in 1.2 *Objectives*. In this context, the approach of template matching is to specify a WuW by a reference recording which is called template. This template recording of the WuW is then compared to speech utterances in which the WuW should be found. The underlying principle remains the same, no matter if these speech utterances originate from a database or are live input speech from a user.
The advantage of the template matching approach is that the user can arbitrarily choose the WuW; still, the model does not need to be retrained for a different WuW (in contrast to [CPH14, SP15]).

The main challenge is to recognize the WuW again, even if spoken in a different manner. This requires extracting a fixed-length representation from the template recording that captures all defining characteristics of the WuW. For this purpose, Chen et al. [CPS15] use the hidden states of an LSTM layer, because they contain all significant information about a word from the current and past frames. As explained in section 3.2, Chen et al. create a template by feeding the reference recording through the network and stacking the last LSTM layer's hidden states of all time steps.

In our work, we adopt the same approach: We use the hidden states of a successfully trained acoustic model to extract characteristic information from speech data. For this purpose, we load the model parameters from the epochs that performed best on the validation set during training.
As we use the Fuel/Blocks based framework, data for evaluation of the template matching performance have to be prepared by storing WuW templates and evaluation utterances in separate subsets of an HDF5 file. For the evaluation utterances, time-aligned labels that indicate the occurrence of the WuW have to be created in order to provide a ground truth.

In our implementation, we use the hidden states of the 2^{nd} LSTM layer from the last k frames as our fixed-length representation of the speech data. According to [CPS15], we choose k to be the length of the WuW template in frames so as to include as much information as possible.

The hidden state vector h^2 at one time step consists of the output activations from the 128 LSTM cells. Stacking k hidden state vectors yields a matrix with dimensions $<k, 128>$. If the current time step is t and we stack the states of the last k frames, we obtain

$$\begin{bmatrix} h_t^2 \\ \vdots \\ \vdots \\ h_{t-k+1}^2 \end{bmatrix}.$$

In practice, we access the hidden activations of the second layer at each time step and append them to a zero-initialized buffer matrix. At the same time, we drop the activations of k time steps ago from the matrix.

Before runtime, a reference matrix A representing the template is created by feeding the template recording through the network. For increased robustness, it is possible to obtain the reference A as an average of e. g. three recordings of the WuW as proposed in [CPS15]. For this purpose, we truncate the matrix representations of the recordings to the length of the shortest one and then take the average. During runtime, the speech data to be evaluated is also fed through the network frame-by-frame. At each time instance, a representative hidden state matrix B is obtained as explained. Both matrices are flattened into vectors a and b, which contain representative information about the template and the current speech respectively. As both vectors have the same length, a simple vector distance score can be used to measure the similarity of the vectors. In practice, the cosine distance is used and the vector similarity

$$c_{sim} = 1 - \frac{a \cdot b}{||a|| \cdot ||b||} \tag{6.19}$$

is an indicator whether the WuW has been detected or not at the current frame.

Part II

TRAINING AND RESULTS

7 PRELIMINARY EXPERIMENTS

After building a framework that allows the NN model to be trained and used for template matching, the next step was to evaluate the performance of differently trained models in order to explore how to properly select and prepare training data. This chapter summarizes the most important findings of this procedure.

At that stage, the WuW detection performance was mainly evaluated on a couple of informal test recordings: Two speakers were recorded and for each of them, two instances of the chosen WuW *hello genie* were captured as templates. For the recordings, the built-in microphone of a smart-phone was used with a distance of about 1 m between speaker and microphone.

For both speakers, we captured three longer recordings with four WuW occurrences each and two short utterances, where a single WuW was part of the utterance. As shown in table 1, the longer recordings either contained silence, short non-WuW phrases or entire sentences between the WuW occurrences. The short recordings consisted of a sentence followed by the WuW and vice versa. This design allows to quickly acquire a dependable impression which aspects of the WuW detection work for a trained model.

Table 1: Design of the informal test recordings for evaluation during the pre-experimental phase. In the long recordings, there is either silence, a short non-WuW phrase or a sentence between the WuW occurrences.

	0:15 m	0:30 m	0:45 m	1:00 m	1:15 m	...	2:00 m
Long file 1	silence	WuW	silence	WuW	silence	...	WuW
Long file 2	phrase	WuW	phrase	WuW	phrase	...	WuW
Long file 3	sentence	WuW	sentence	WuW	sentence	...	WuW
Short file 1	sentence + WuW						
Short file 2	WuW + sentence						

7.1 TRAINING ON FULL TIMIT DATABASE - WITH GRAY ENCODING

The first set of experiments was conducted with models trained on the TIMIT speech database [GLF+93], which consists of 4.5 hours of speech data. For a start, we trained the LSTM network on the whole TIMIT database. The sentences were cut into separate words using the provided word-level aligned transcriptions.
For the output target labels, the identification number of each word in the dictionary was binary encoded using gray codes. The output neurons of the network were thus trained to predict the correct combination of zeros and ones for each word. In this way, we wanted to prevent huge output layer sizes leading to exces-

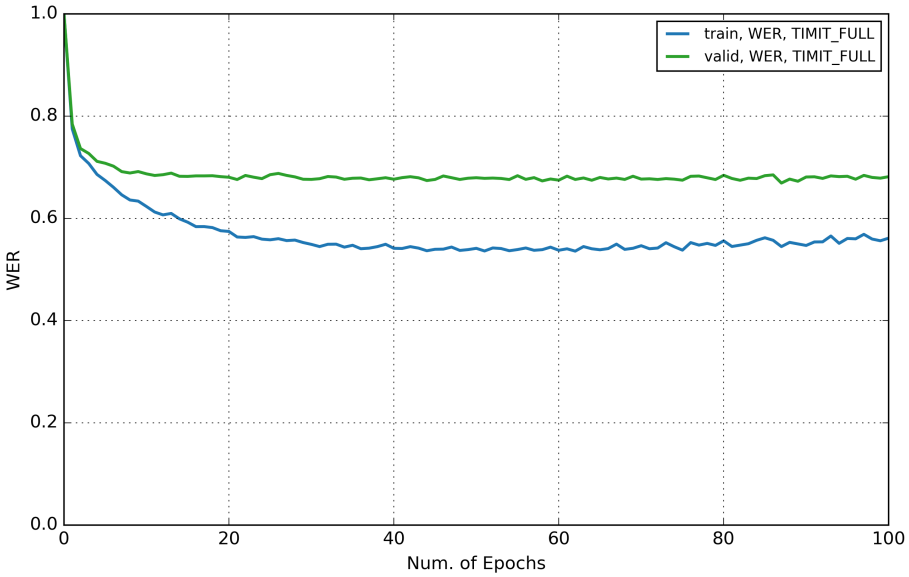
sive memory consumption.
The evolution of the WER during training on the full TIMIT database is depicted in figure 12a and shows that although the model converges, the WER stays above 50% even for the training set. The best performance on the validation set is achieved in epoch 87 with a WER of 66.9%, so only a third of previously unseen examples of trained words are recognized correctly.
In this chapter, a word is counted as correctly recognized if the network predicts the correct word label at the last frame. The WER is then obtained as the number of incorrectly predicted words divided by the total number of words (unlike the frame-based WER used elsewhere in the thesis).

We found the resulting WuW detection performance of that model to be very poor in spite of the fact that the entire TIMIT database was used for training. Figure 12b shows an example of the performance on a recording consisting of four WuW occurrences with short non-WuW phrases in between. The cosine similarity score exhibits no difference between WuWs and non-WuWs; it would not even allow to distinguish between the presence of speech and silence by means of a simple threshold.

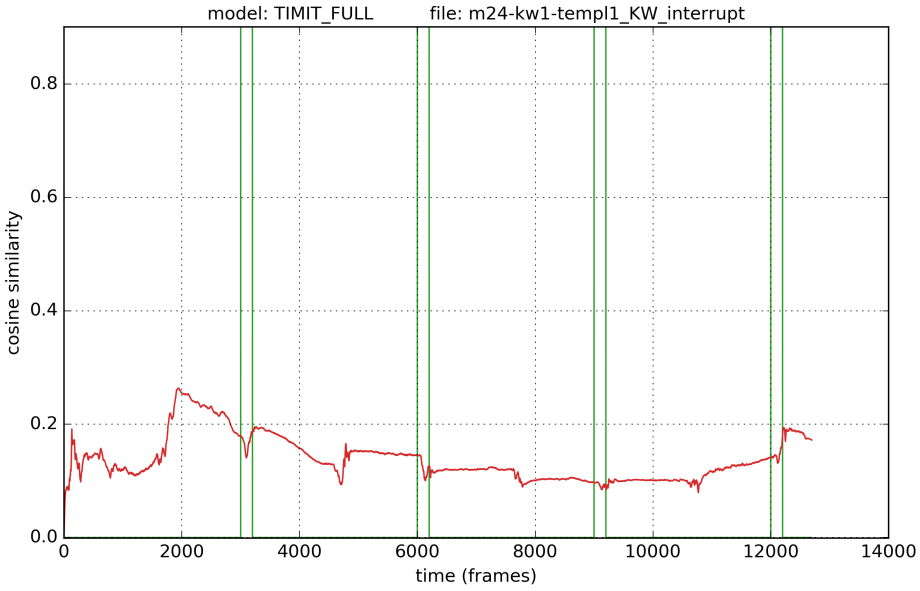
We reckoned that two main factors contribute to the poor performance:

Firstly, too few occurrences per word in the training data do not allow the network to learn to recognize that word, which is also reflected by the high WER. Thus, the network did not sufficiently learn to capture the characteristics of a variety of words in the activations of the last hidden layer. However, our template matching approach fully relies on this ability of the network.

Secondly, we rethought the binary encoding of the word labels and discovered that it might severely degrade the ability of our model to discriminate between different words: As mentioned, our general goal is to learn abstract representations of words in the second hidden layer. These hidden activations span a vector space in which similar sounding words should be close together, whereas dissimilar words should be clearly separated. As the transformation from the activations of the last hidden layer to the output layer is a simple linear transformation, there is a strong interdependency between the last hidden layer and the output layer.
We now consider the following two examples with binary encoding of the output layer using gray codes: The labels of two completely different sounding words might differ by just a single bit, but their abstract representations are expected to reside in different parts of the vector space. On the other hand, the labels of two very similar sounding words might be completely different although their abstract representations are expected to be very similar. Therefore, gray encoding in the output layer prevents learning clearly separable representations of words as it corrupts these representations in the last hidden layer.



(a) Evolution of the Word Error Rate during training on the full TIMIT database. The lowest WER on the validation set is 66.9%, indicating that the provided data was not sufficient to properly learn all words.



(b) Testing the model reveals that WuW detection does not work at all with this setup. The figure shows the cosine similarity between speech in the evaluation recording and the template. The 4 occurrences of the WuW are indicated by green labels. In between the WuWs, short non-WuW phrases were uttered.

Figure 12: Performance of the LSTM model when training it on the full TIMIT database.

7.2 TRAINING ON TIMIT SA SUBSET - WITH REDUNDANT ENCODING

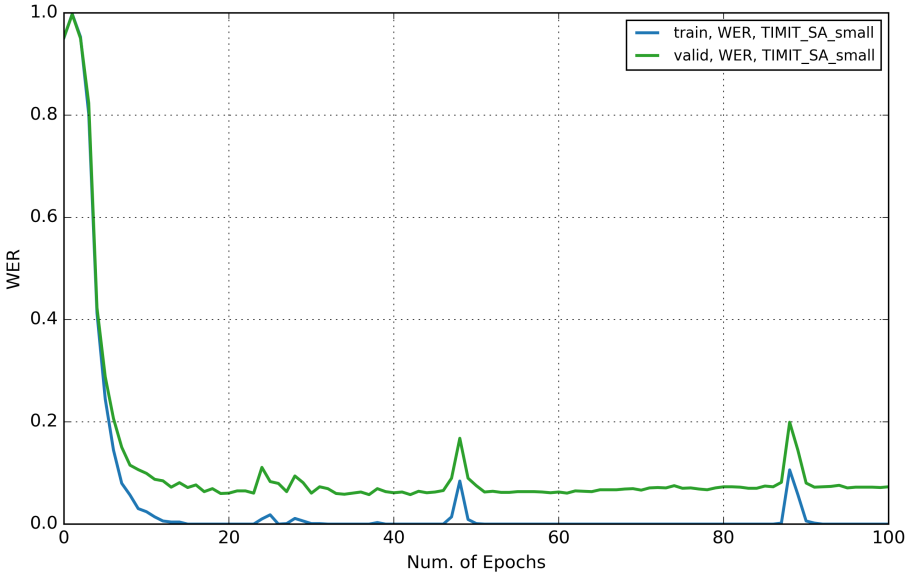
The next experiment was conducted on a subset of the TIMIT corpus. We wanted to explore the effect of training a model on few different words with many occurrences, therefore we selected the SA sentences which are available for all speakers in the database. These two sentences contain 21 different words:

(sa1) She had your dark suite in greasy wash water all year.
(sa2) Don't ask me to carry an oily rag like that.

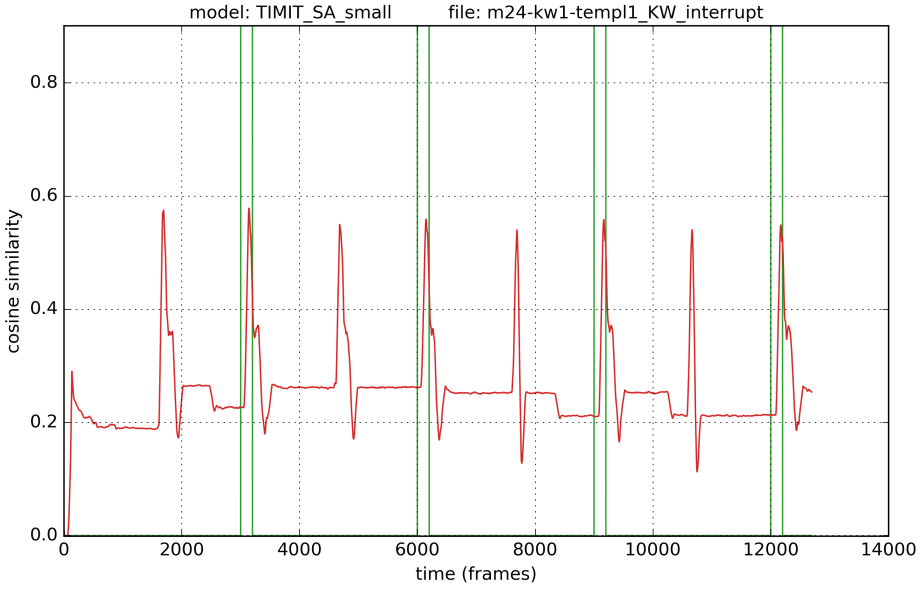
After we had already finished the experiment, we discovered that we had coincidentally encoded the word labels with random 10 bit sequences, although 5 bits would have been sufficient for 21 words. I. e. redundant information was added to the labels, similar to the concept of error correcting codes. The resulting labels are a compromise between the previously used binary encoding with gray encoding and sparse one-hot encoding. In this way, the separation between labels of different words is much better, also allowing for the representation in the last hidden layer to be more clearly separable for different words.

For training, we chose to use approx. 50 of the SA sentences, resulting in 50 different occurrences of each word in the training set. We refer to this as the SA *small* set. For comparison, we also trained on all available SA sentences, resulting in almost 400 occurrences per word.
The evolution of the training with the small set is depicted in figure 13a with a best validation set performance of 5.8% WER in epoch 37. The error rate on the training set even descends to 0%. The provided data of 50 occurrences per word appear to be sufficient to learn the 21 words almost perfectly. For the entire set of SA sentences, the best error rate on the validation set is reduced by a factor of 2 to 2.8% WER due to the higher number of occurrences per word.

However, despite the flawless validation set performance, the WuW detection evaluation revealed that learning to recognize 21 different words is not enough to allow the model to separate previously unseen WuWs from non-WuWs. As shown in figure 13b, the similarity score spikes in the presence of speech, but the peaks of WuWs and non-WuWs are not separable. In fact, the model behaved similar to a voice activity detector. Interestingly, the performance difference between the small SA set and the full SA set was negligible in the context of these considerations.
We concluded that the words in the SA sentences were too short and the diversity of their phonetic content not significant enough to ensure that the model learns to extract meaningful abstract representations from words.
For the sake of completeness, other subsets of TIMIT, such as only SI or only SX sentences and a combination of the two, were also investigated but did not bring new insights.



(a) Evolution of the WER during training with the small SA subset. The lowest WER on the validation set is 5.8%, indicating that the network learned to recognize the 21 different words almost perfectly.



(b) The model trained on the small SA subset also does not perform WuW detection but behaves like a voice activity detector. The evaluation file is the same as in figure 12b.

Figure 13: Performance of the LSTM model when training it on approx. 50 instances of SA sentences from the TIMIT database.

7.3 WORD-WISE VS. SENTENCE-WISE TRAINING

As it had become clear that many occurrences per word are required for sufficient training of a model, we switched to the Wall Street Journal database [PB92] as a source of training material. It consists of selected sentences from the Wall Street Journal, a newspaper mainly dealing with business matters. The material is provided in predefined training, validation and test sets. We used the pilot database commonly referred to as WSJ0 with approx. 15 hours of training data duration.

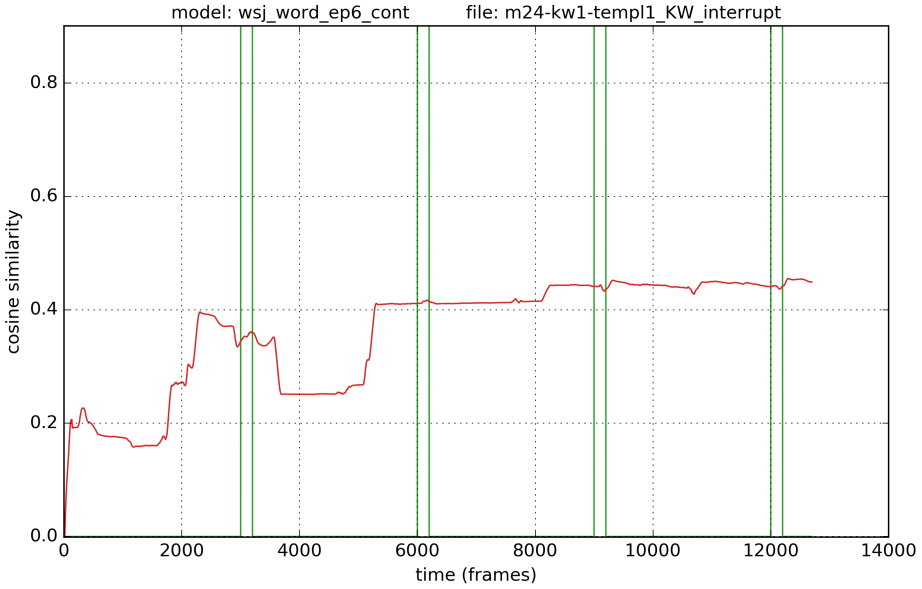
Note that for the very first tests with WSJ0, we still used binary encoded word labels using gray codes, because at that point we had not yet discovered that the SA experiments described in 7.2 used labels with redundant information added.
Our first investigation comprised two variants of preparing the training data: One variant is to treat entire sentences as one sequence, while the other is to cut sentences into separate words and treat single words as one sequence.
The evaluation of WuW detection as depicted in figure 14a clearly shows that word-wise training does not work with WSJ0. The obscure score evolution indicates that the states do not seem to capture any significant information at all. In addition, the hidden states of the last hidden layer seem to remain in a similar space most of the time, resulting in little changes of the cosine similarity score. A natural explanation would be that as one sequence evolves, the states are never really reset when the forget gates are not properly learned. With word-wise training, this could easily happen as the network is never exposed to a succession of words during the training process.
On the other hand, the sentence-wise trained model exhibits some promising behavior as shown in figure 14b. The evolution of the similarity score follows the recorded events with spikes whenever speech is present. However, WuWs and non-WuWs still cannot be separated, probably due to the binary encoding of labels and the presence of many seldom and insignificant words in the training data despite the larger database.

The comparison shows that it seems to be vital for the network to learn how to deal with a succession of multiple different words during training as opposed to only learning to recognize a single word. Therefore, we chose to proceed with sentence-wise training for the remainder of our work.

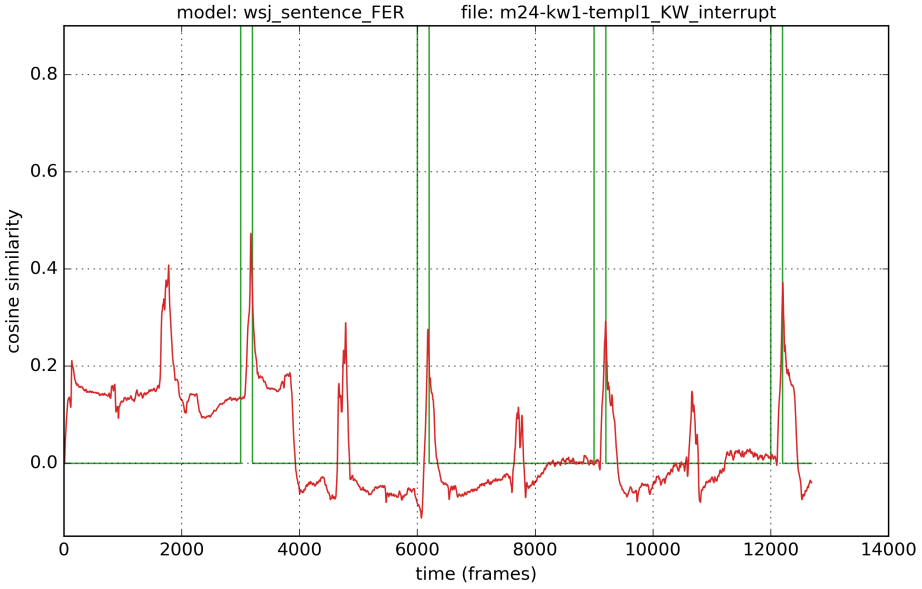
The bottom line of the preliminary experiments were two main findings:

The results of the TIMIT SA experiments indicated that a sufficient number of occurrences should be available for *all* words in the training set and that wise selection of the words might be important. For this reason, it was clear that we had to stick to the WSJ0 database at least. Note that according to results from our later experiments even more than 50 occurrences per word are required, probably because when training on more different output targets, more occurrences per word are required to learn to separate all of them.

As none of the pre-experiments brought promising results, we decided to use one-hot encoding, which is common practice.



(a) Testing a model trained on single words from the entire WSJo database. The evolution of the similarity score indicates that the model has not learned to deal with the succession of multiple spoken events.



(b) Testing a model trained on whole sentences from the entire WSJo database. In this case, the temporal evolution of the similarity score follows the occurring spoken events, even though there is still no separation between WuWs and non-WuWs.

Figure 14: Comparison of word-wise and sentence-wise trained models. In both cases, the evaluation file was the same as in figures 12b and 13b.

8

TRAINING THE LSTM NETWORK ON WSJ0

A core part of this master thesis was to find the right way to train the NN-model. In chapter 7 we describe how we gained knowledge regarding the right network and data structure.
In this chapter, we describe how we carefully selected the training data from the WSJ0 corpus to ensure that only meaningful information was provided to the network (8.1). Then we train several systems with different hyperparameter settings to explore the influence of each parameter. We start by explaining our reference system to which we compare all other trained models. In all the experiments the goal is to achieve a low WER.

8.1 SELECTION OF THE DATA

In the pre-experiments it was evident that it is not beneficial to provide as many data as possible to the network. On the contrary, providing all available speech data seems to degrade performance.
As we saw in section 7.2, it is beneficial to use only a few words with many occurrences. So we decided to take the most frequently occurring N words from the WSJ0 corpus. Spoken punctuation marks such as "double quote" or "comma" as well as noise and silence-tags were excluded from these words because of their disproportionate number of instances.
We additionally restricted the most common N words to words that have at least 5 characters in order to ensure that we train on meaningful and significant data.
As described in 7.3 *Word-wise vs. sentence-wise training*, sentence-wise training ensures to learn the forget gates in a proper manner. So we cut out the desired words from the given sentences and compose with them new sentences of random length between six and ten words.

8.2 REFERENCE SYSTEM

We wanted to design a system that downscales the problem and is able to learn the provided data properly. In [SKMR13], a CNN-based system trained on a 50-hour English Broadcast News database and 512 output targets is described. It achieves a WER of 22.3%.
If we assume that they also trained on the most frequently occurring 512 words, we can estimate that the corpus is reduced to about $\frac{1}{4}$ of the original data. Assuming an average word length of about 0.35 s, we can calculate that they used about 250 occurrences per word on average for training.
Taking this into account, we decided to set the output-layer size to 64 output targets, which results in 180 occurrences per word on average and in 1.21 hours of

training data. According to section 8.1, each of the 64 words has at least 5 characters.
Normalization is performed as described in section 4.3.2, except that we do not use global but utterance based normalization. That means that μ and σ are estimated for every utterance separately. This is closer to real-time applications where μ and σ have to be estimated from a window of past frames of the current input signal.
The learning rate is set to a factor of 0.002 as proposed for the Adam-algorithm by Blocks and log-filter bank features are used. These settings are summarized in table 2.

Table 2: Settings of Reference System wsj_64_reference

Parameter	**Setting**
Output-Targets	64
Avg. num. of instances/word	181
Hours of data for training	1.21
Min. Chars	5
Words per training sentence	6-10
Normalization	utterance based
Feature type	log-filter bank
Learning Rate	0.002

In figure 15, the training- and validation set performance of the reference system (subsequently marked in figures as wsj_64) is shown. The training curve shows convergence and a minimum WER of 4.26% at epoch 158. This shows that the provided data can be learned almost perfectly. Unlike the previous chapter, the WER is now frame-based (see section 5.4.2). A WER of about 5% means that on average a learned word is detected after 1/20 of its frames.
The validation set performs best in epoch 55 and yields a WER of 23.97%. As mentioned above in [SKMR13], a similar WER of 22.3% is reported. This suggests that the training data selection was done in a meaningful way.
For the training process of the reference system, the necessity of early stopping can be observed in practice. Due to overfitting, the training set performance improves up to epoch 158 but validation set performance already degrades after epoch 55. Therefore it is crucial to save the network parameters of the best epoch.

Apart from the fact that the training curve converges, huge spikes are visible in figure 15. This will be discussed in more detail in section 8.5.

To maintain clarity, only the validation set performance is shown in the figures for all subsequent systems.

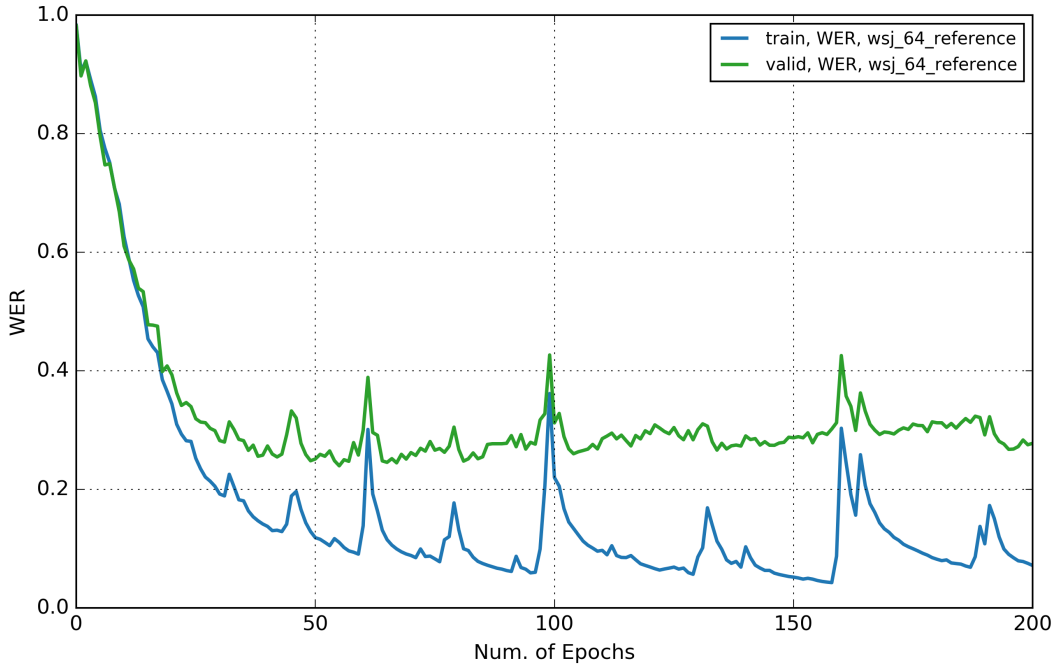


Figure 15: Performance of training-set (blue) and validation-set (green) of the reference system. Overfitting of the training data occurs after epoch 55.

8.3 STOCHASTICITY OF THE TRAINING PROCESS

Batch shuffling randomly reorders the training examples in the batches after every epoch. This causes the SGD-algorithm to descend on different paths on the error surface in every epoch. Therefore every training process reaches different (local) minima resulting in a different overall performance.
These variations in valid performance of the training process are shown in figure 16. The first two repetitions of the reference-system (WH1 and WH2) share the same data file as the reference system. For each of the last two repetitions (WH3 and WH4), a new data file was written. This introduces an additional element of chance to the training process because the training sentences are generated with random length between 6-10 words. This means that every time a data file is written, sentences contain words in different orders and have a different length. In table 3, this additional element of chance is observable as well. The WER of the first three systems differs only slightly (0.4% in WER absolute), whereas the latter two systems differ by more than 2% in WER absolute if compared with the average performance of the first three systems.
Clearly the random composition of the data file of the last system (WH4) is favorable and leads to the best obtained WER of 22.14%.

Altogether these experiments show that there is a large variation of more than 2% in WER absolute depending on the composition of the sentences in the data file and on the order of the batches used to perform SGD. This has to be taken into account when comparing the training processes of systems with changes in hyperparameter settings.

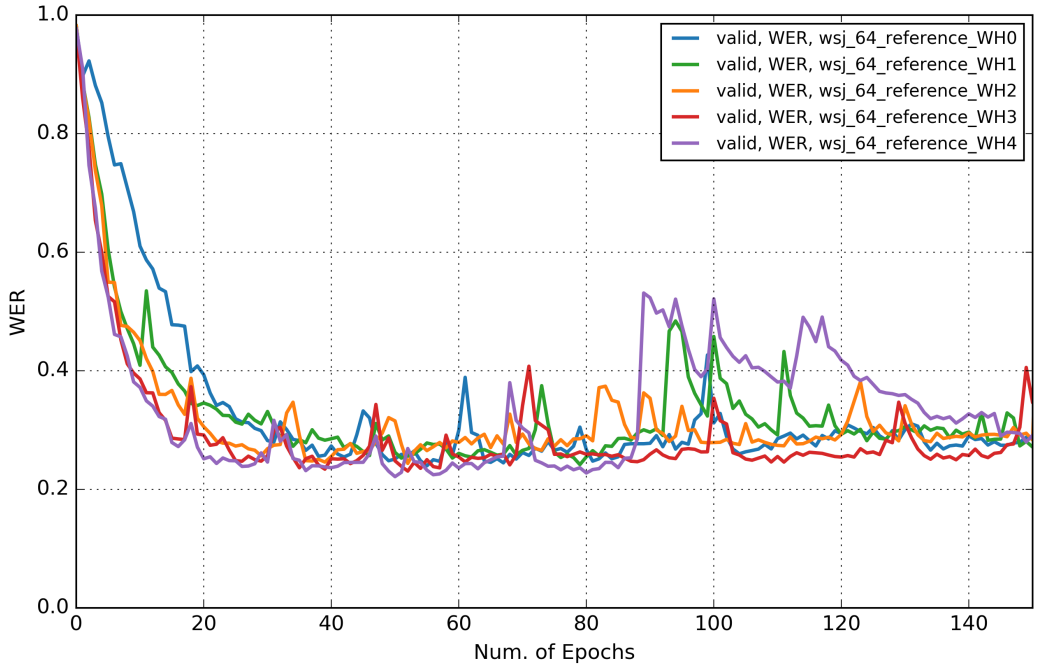


Figure 16: Training performance for different repetitions: Due to batch shuffling the SGD-algorithm descends on different paths on the error surface reaching different minima which leads to a slightly different best WER for each repetition.

Table 3: Large variation of more than 2% in WER absolute for repetitions of the training process due to the changing composition of sentences for different data files and the random order of batches for the SGD-algorithm. The first two repetitions show less variation because they share the same data file as the reference system. For both of the last repetitions, a new data file was written. This leads to bigger deviations in WER.

System	best WER in %	@ epoch
wsj_64_reference_WH0	23.97	55
wsj_64_reference_WH1	24.17	79
wsj_64_reference_WH2	24.37	52
wsj_64_reference_WH3	23.08	52
wsj_64_reference_WH4	**22.14**	50
Average	23.54	57

8.4 INFLUENCE OF GLOBAL NORMALIZATION

In 4.3.2 *Normalization of features* we discussed in theory that it is important to apply feature normalization when training NNs. For a CNN based system, [SKMR13] show that global normalization is favorable over utterance based normalization. So far, utterance based normalization was applied in our thesis and we wanted to show the performance impact of global normalization for our LSTM-based system. Figure 17 shows a severe performance degradation for global normalization. Not only is convergence much slower but the minimum WER on the valid set is also 46% in epoch 170, which is almost twice as high as the WER of the reference system. It is out of scope for this thesis to investigate the reasons for this severe performance degradation when applying global normalization. Nevertheless, we choose utterance based normalization for all further applications, which has the additional advantage of being applicable in real-time scenarios.

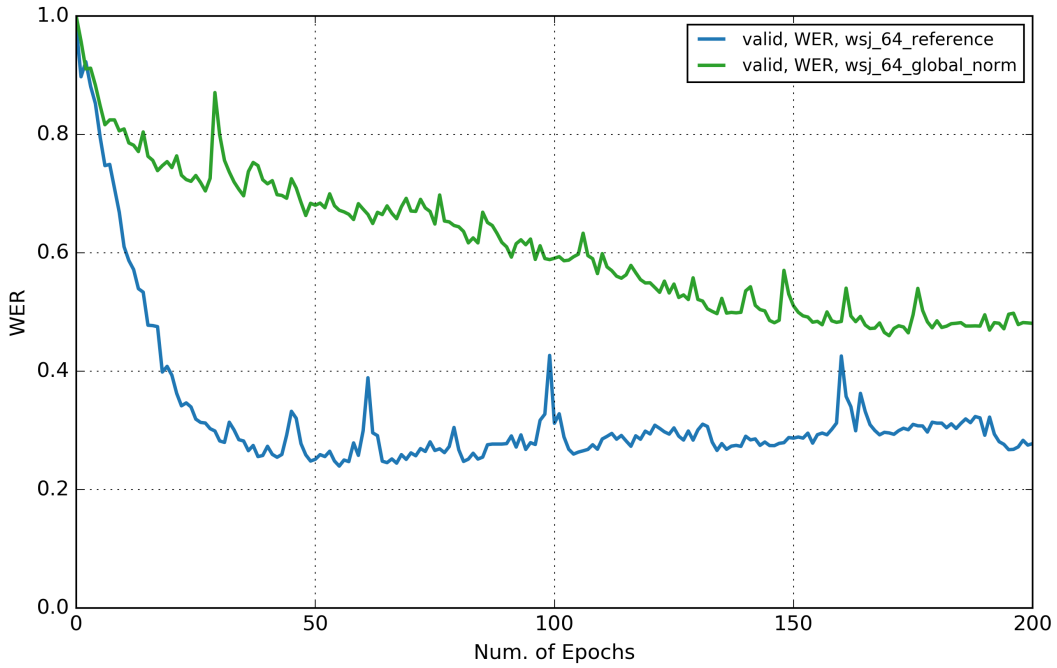


Figure 17: Global normalization causes severe performance degradation and slows down convergence. The best WER drops from 23.97% in epoch 55 to 46% in epoch 170.

8.5 INFLUENCE OF THE LEARNING RATE PARAMETER

One hyperparameter, which influences convergence significantly, is the Learning Rate (LR). As explained in 5.2.1 *Stochastic Gradient Descent with Adam*, the effective step taken in parameter space is always bounded by the step-size hyperparameter α or LR, as it is generally called. This means that the algorithm is able to reduce the LR when it comes closer to an optimum. Nevertheless, the predefined LR heavily influences convergence behavior. We conducted experiments on the reference system by varying the LR in the scope of two magnitudes.
In figure 18, the convergence behavior of four selected systems is shown. Table 4 gives an overview of the chosen LRs and associated WERs for each system in figure. It can be observed that the lower the learning rate is, the slower is the convergence. For the system with the smallest LR (blue curve) convergence is already too slow to reach a good minimum in reasonable time. It needs 177 epochs to achieve its best WER of 26.96%, which is 2% worse than the system with the default LR of 0.002. The red curve with a LR of 0.02 shows that if the LR is too large the algorithm does not converge at all because the steps taken in parameter space are so big that they "overshoot the aim".
For our dataset, a learning rate of 0.001 (green curve) shows an absolute improvement of 0.9% in WER compared to the reference system WH2. However, this behavior is not significant when taking into account the observations from 8.3 *Stochasticity of the training process*.

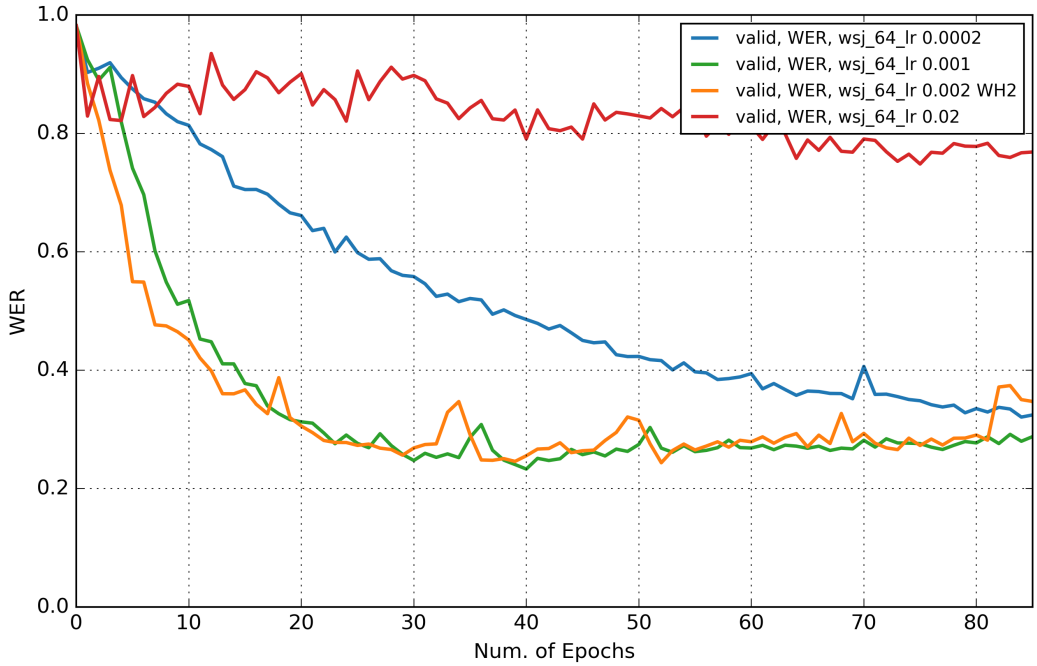


Figure 18: Varying the LR: The smaller the learning rate, the slower the convergence (orange, green and blue curve). If the LR is too large, the system does not converge at all (red curve).

Table 4: Chosen LRs and associated minimum WERs for each system in figure 18.

LR	WER in %	epoch
0.0002	26.96	177
0.001	23.30	40
0.002	24.37	52
0.02	74.83	75

In 8.2 *Reference System*, spikes were observed especially in the training curve. These spikes are closely related to the LR parameter. To observe this in more detail, the training curves of the converging systems shown in figure 18 are plotted in figure 19.
It is observable that the spikes get bigger the lower the WER of the training curve gets and that the spikes are bigger for a larger LR. The blue curve shows the smallest spikes. The orange curve shows much larger spikes than the green curve, even though the LR is only a little bit smaller. Overall it is visible that the spikes slow down convergence significantly and also prevent from fully reaching a minimum.

ADAM evaluates the gradient at subsamples of data called batches. It uses moving averages of the first and second order moments of the gradient to anneal the LR. Towards an optimum, there is more uncertainty whether the direction of the estimated gradient corresponds to the true direction of the gradient over the entire set. The algorithm automatically decreases the learning rate when it comes closer to an optimum (annealing) but obviously for LSTM networks this does not always work in practice.
So when the estimated direction of the gradient does not correspond to the true direction of the gradient, the parameters are updated in the wrong direction. When the LR parameter is smaller, this happens as well but the steps taken in the wrong direction are smaller and therefore the spikes in the cost and consequently in the WER are also smaller, leading to smoother (but slower) convergence.

An other explanation for this phenomenon was found in [Gra13]. Graves reports that

> "one difficulty when training LSTM with the full gradient is that the derivatives sometimes become excessively large, leading to numerical problems."

This seems to happen especially late on in training, after the process has started overfitting on the training data. This would be an explanation for the huge spike at epoch 150 of the orange curve in figure 19. According to Graves, one solution to this problem is to clip the derivatives of the cost if they exceed a certain threshold.

We assume that maybe a combination of the two mentioned phenomena takes place. To prove this, a detailed investigation of the derivatives during the training process would be necessary, which is beyond the scope of this thesis.

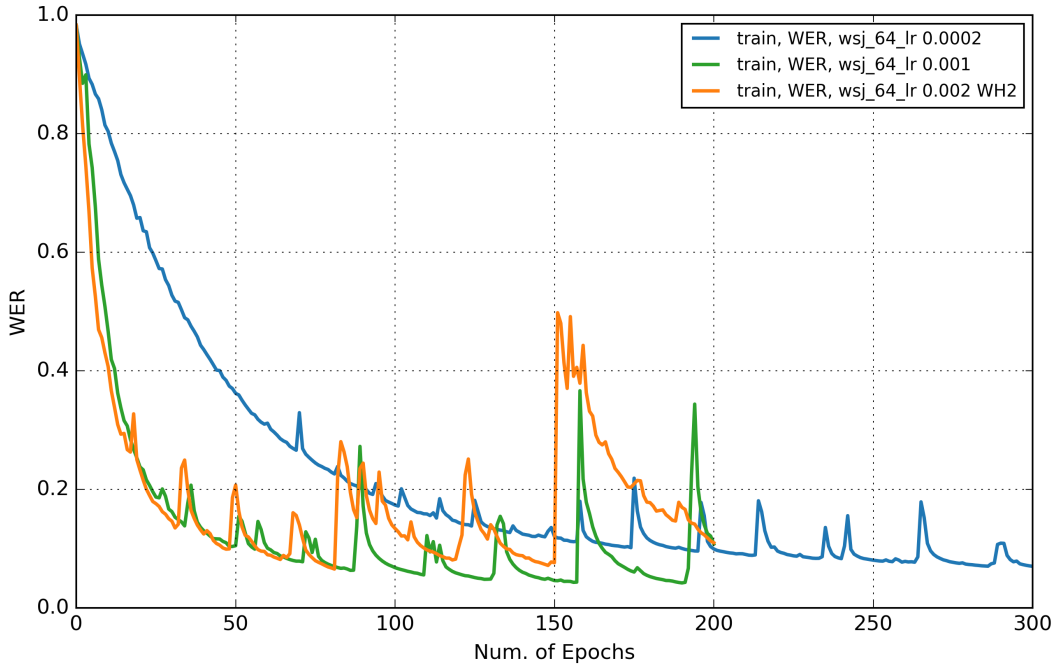


Figure 19: Spikes in the training curves of the converging systems shown in figure 18. Automatic annealing does not always work in practice. A smaller LR reduces the phenomenon but also increases convergence times significantly.

8.6 INFLUENCE OF MFCC FEATURES

For reasons described in 4.3.1 *Mel-scale log-filter bank features*, log-filter bank features are the better choice for NNs. For a context-dependent-DNN-HMM system presented by [LYHG12], a relative improvement of 5.6% in WER could be achieved using 40 dimensional log-filter bank features instead of MFCC features.
During our experiment, we could not observe a significant difference. The systems trained on MFCC features achieve a WER of 24.37% which is similar to all the all repetitions of the reference system for this data file (reference, WH1 and WH2, see table 3). Also the convergence behavior is very similar, as can be seen in figure 20. We believe that the effect of the two different types of features is not observable for these systems. On the one hand because there is too little training data at all and on the other hand because the stochasticity of the runs is too high. Averaging over multiple runs would certainly help to point out a difference but anyway it is sufficient for us to know that there is no significant difference.

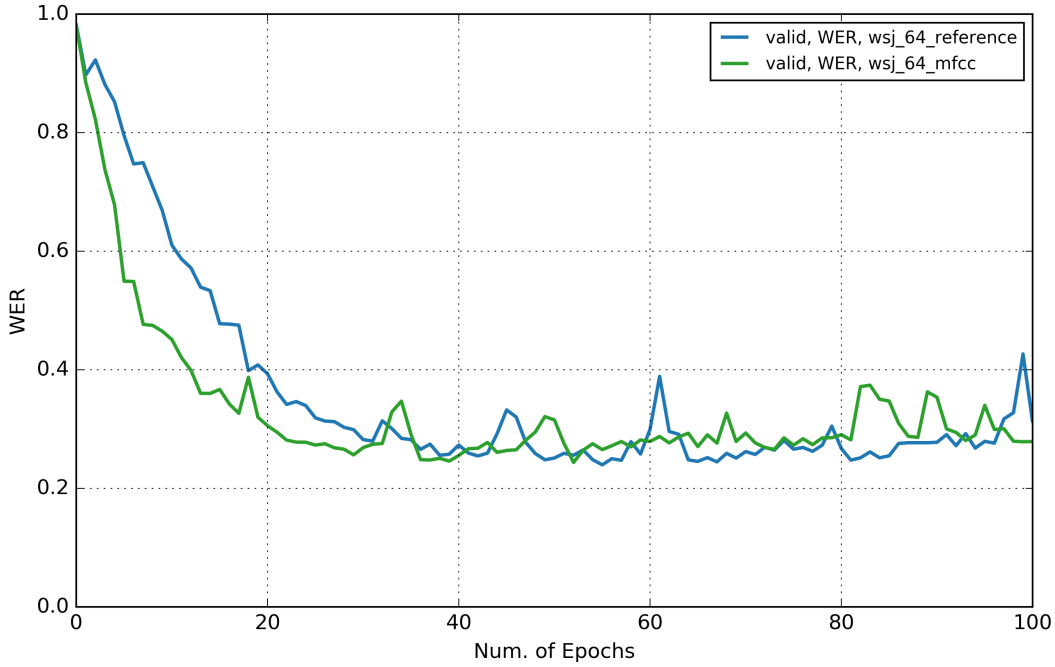


Figure 20: MFCC and log-filter bank features: no significant difference can be observed for our setup.

9 RESULTS FOR DIFFERENT OUTPUT TARGET SIZES

In this chapter, the main findings are presented. In the first section, we describe the recorded evaluation corpus. Then we introduce several systems that are trained with different amounts of words (output targets) and evaluate their WuW performance.

9.1 EVALUATION CORPUS

The evaluation corpus was recorded in a conference room with a size of 6 x 4 meter. The chosen WuW was "Hallo Computer" with German pronunciation and it was excluded from training. The speakers were sitting in the center of the room at a table. The recording microphone was placed at a height of about 2 meters on the wall opposite to the speaker. This was done to simulate a home-automation environment. All files were sampled with 16 kHz to match the sample-rate of the WSJo corpus.
One file has a duration of about two minutes and the WuW is uttered four times, as shown in table 5. In between, informal conversations in German between the main speaker and a conversation partner are recorded to imitate a real-world scenario. Before and after the WuW was uttered, there are at least two seconds of silence.
The database consists of 3 male and 3 female speakers and for each of them 5 files and 6 WuW-templates were recorded. As we average over three templates, this results in two different templates for each speaker. Altogether the WuW was uttered 120 times and about 40 min of conversational speech were recorded. This data was evaluated for both templates to statistically double the amount of the recorded material.
Note that the templates as well as the evaluation files were recorded in the far field with a speaker to microphone distance of about 2.5 m.
The WuW detection performance is measured in terms of EERs. The EER is defined as a special kind of operating point where the FA and FR rate are equal. All EERs are obtained in the following way: For every speaker, we constantly increase the value of a threshold. All the peaks of the cosine similarity, which are above the threshold, are counted as a WuW occurrence. FRs are missing peaks above the threshold within the labeled range, FAs are all peaks above the threshold, which do not lie within the labeled range.
So per speaker a threshold is swept through the similarity scores and for every threshold the FA and FR rates are recorded. Then the threshold containing the EER is determined per speaker. The results of all speakers are then averaged to obtain the total EER.

Table 5: Structure of the recorded files for the evaluation corpus.

	0:00-0:28 m	0:30 m	0:34-0:58 m	...	2:00 m
File 1-5	conversation	WuW	conversation	...	WuW

9.2 DESCRIPTION OF TRAINED MODELS

In this section we trained several models with the same hyperparameter settings as shown in table 2. The main idea was to find a system that exploits best the data of the WSJo corpus for WuW detection. As described in section 8.1, for each system the most common N words containing at least 5 characters were selected. If the overall number of words trained on is increased (bigger output layer), the average occurrences per word decrease.
Therefore we decided to train 6 systems with successively increased output layer sizes and evaluate the WuW performance for each system. In that way we wanted to find an optimal trade-off between the average occurrences per word and the output layer size. In table 6, the corresponding systems are shown. The output layer sizes of the systems were 32, 64, 128, 256, 512 and 2048 words, respectively. The amount of training data used starts with 0.8 hours for the smallest system to 5.7 hours for the largest system. The average number of occurrences per word varies from 256 to 22. Note that the values of wsj_64 and wsj_128 are averaged over five training repetitions (in *italics*).

The achieved WERs indicate that the more occurrences are available per word, the better the system learns to recognize the words. We would like to remind the reader that we use a frame based WER. So given a certain WER, we can't discriminate if a system recognizes a few words correctly very early in time or if it recognizes many words correctly but only in the last frames. A higher WER only tells that the system recognizes less frames on average. So if we add new words (with less occurrences) to the training data, it might be that the system learns the old words equally well and the new ones worse but it might also be that it recognizes all the words worse.

In table 6 it is also observable that convergence times (column "@ epoch") increase for increased output layer sizes with less average occurrences per word. For a small output layer size and many occurrences per word, the training algorithm is provided with similar constellations of data during one epoch and within consecutive epochs. In addition, frequent words are seen more often within one epoch, which both leads to fast convergence. For a huge output layer size with many rare words, there are many unique constellations of data within one epoch and in every new epoch there are new unseen constellations of the data which leads to slow convergence.
System wsj_2048 already shows its best WER at epoch 22 which is not consistent with the previous tendency. This is because the training process did not converge anymore. Here it is obvious that the output layer size was too large compared to the number of occurrences per word to learn a meaningful parameter setting.

Table 6: WER results for systems with increasing output layer size and decreasing average occurrences per word in the training data. The more occurrences are available per word, the better the system learns to recognize the words (lower WER). Convergence times increase for bigger output layer sizes. Wsj_2048 does not converge anymore, due to the disproportion of output layer size and average occurrences per word. Values of systems in *italics* are averaged over 5 repetitions.

System (# words)	**WER [%]**	**@ epoch**	**training hours [h]**	**Avg. occur. per word**
wsj_32	18.78	27	0.81	256.0
wsj_64	*23.54*	*58*	1.21	181.0
wsj_128	*30.39*	*85*	1.71	124.7
wsj_256	38.13	108	2.45	84.7
wsj_512	44.52	110	3.36	56.0
wsj_2048	80.89	22	5.65	22.0

9.3 WUW PERFORMANCE OF TRAINED MODELS

In this section, we analyze and present the achieved results. First we analyze the achieved EER per speaker in detail. Then we present a method to improve the results and finally we evaluate the overall performance of the systems presented in the previous section.

In table 7, a detailed overview of the results for the best system, wsj_128, is given. For every repetition (WH0-WH4), the EER per speaker is given. The average or total EER of every repetition is shown in the last row. The average performance per speaker over all repetitions for the wsj_128 and the wsj_64 system is shown in the last two columns, respectively.
Except for the first repetition (WH0) the results vary highly between the speakers within a certain repetition. For the repetition WH4 with the worst total EER of 0.43, the best speaker achieves an EER of 0.02 and the worst has an EER of 0.85. So the system can work almost perfectly for one certain speaker and not at all for another one. The better the total EER, the weaker this trend. So for the best repetition (WH0), the performance between speaker differs only slightly.
The average performance per speaker over all repetitions is quite different, which is not a desirable property. Here, too, it can be shown that the lower the average EER per speaker is, the lower the variations are from repetition to repetition.
As a comparison, the average EER per speaker of the wsj_64 system is shown in the last column. The same speaker performs equally well on both systems and for every speaker the wsj_128 outperforms the wsj_64 system.
This leads to two important findings: First, this shows that the two models with different output layer sizes are comparable because results are consistent. Second, and more importantly, the wsj_128 outperforms the wsj_64 system for every single speaker. This shows that the ratio of output target size and average number of occurrences is favorable for the wsj_128 system. A bigger output layer, where all representations of the words were learned well, leads to better generalization, which itself leads to a more robust representation of the WuW.
We assume that one answer for the huge differences in WuW performance for repetitions of the same model can be found in the SGD algorithm. For every repetition, it takes another downward path on the error surface. This results in different parameter settings of the model from repetition to repetition but leads to similar WERs, because there are many possible parameter combinations to fit the training data equally well and to achieve good performance on the validation set.
In other words, our network is far too flexible for the amount of training data provided. We assume that results become more consistent if the amount of training data is increased.

In figure 21, the similarity scores of one file from system wsj_128 are shown. This is an example of speaker w58 with an EER of 0.85, so hardly any WuW occurrence is detected correctly. By looking at a couple of files in this way, we observed that the slope of a WuW occurrence is steeper than any other peak. So we tried to take advantage of this by differentiating the similarity scores and obtained huge improvements in terms of EERs. For this selected speaker, for example, the EER improves from 0.85 without differentiation to 0.13 with differentiation of the similarity scores.

Table 7: Detailed overview of the EERs per speaker for the system wsj_128. Wsj_128 outperforms wsj_64 on average and for every single speaker due to a better ratio of output target size and average number of occurrences per word.

	EER for wsj_128						EER for wsj_64
Speaker	**WH0**	**WH1**	**WH2**	**WH3**	**WH4**	**∅ /speaker**	**∅ /speaker**
m24	0.12	0.34	0.12	0.26	0.79	0.33	0.45
m26	0.05	0.20	0.05	0.35	**0.02**	0.13	0.14
w22	0.04	0.12	0.12	0.10	0.05	**0.09**	0.12
w31	0.07	0.41	0.09	0.17	0.22	0.19	0.32
w58	0.12	0.56	0.51	0.56	0.85	0.52	0.64
m25	0.19	0.09	0.15	0.57	0.67	0.33	0.40
total EER	**0.10**	0.29	0.17	0.34	0.43	0.27	0.34

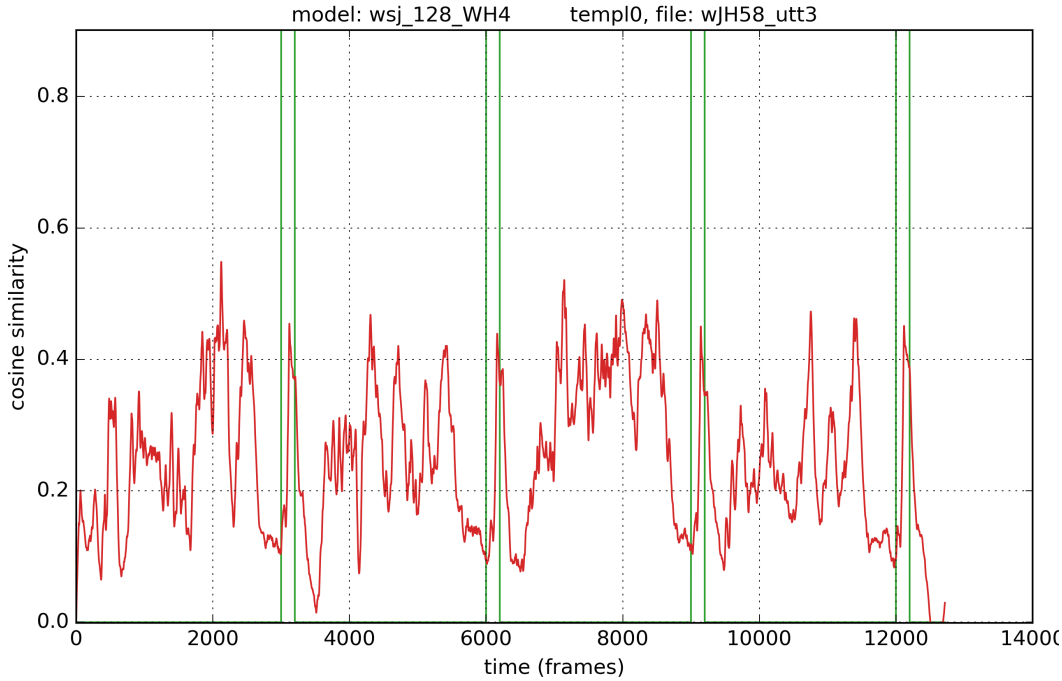


Figure 21: Similarity scores of speaker w58 with an EER of 0.85. The slope of a WuW occurrence is steeper than any other peak. Differentiating the similarity scores leads to an improved EER of 0.14 for this speaker.

In table 8, the total EER for both the plain and the differentiated similarity scores are shown for the system wsj_128. By differentiating the similarity scores, the average performance improves from an EER of 0.27 to 0.14. We differentiated the similarity scores for the system wsj_64 as well and observed a similar behavior.
For the best repetition of wsj_128, differentiation did not improve the results anymore, as can be seen in the first row of table 8. This is an overall trend, the better the EER, the less effect differentiation of the similarity scores has.

By differentiating the similarity scores, we exploit the fact that the WuWs were recorded with a few seconds silence before them. This leads to a particularly large and steep increase of the similarity scores. So we assume that without this preceding silence, results would not improve that much.

Table 8: Under the condition of a preceding silence before the WuW, differentiation of the similarity scores helps to improve the results dramatically. The better the EER, the less effect differentiation of the similarity scores has.

System	WER [%]	EER	diff. EER
wsj_128_WH0	30.69	**0.10**	**0.14**
wsj_128_WH1	29.32	0.29	0.12
wsj_128_WH2	28.92	0.17	0.14
wsj_128_WH3	31.86	0.34	0.18
wsj_128_WH4	31.17	0.43	0.12
avg.	30.39	0.27	0.14

Now we will discuss the overall results of the systems described in 9.2 *Description of trained models*. In figure 22, the performances of the systems with increased output layer size are shown in terms of EER with (light green) and without (blue) differentiation. Note that for the systems wsj_64 and wsj_128, EERs are averaged over 5 repetitions, while for the other systems only one repetition was done. Nevertheless, the performance of the system wsj_128 is the best regarding both the EER with and without differentiation. Systems with less and more output targets have rising error rates. The worst performing system is wsj_2048, which did not converge in training anymore.
In table 9, these results are connected with achieved WERs and the average occurrences per word. We conclude that it is not possible to perform well on WuW detection tasks if the training set has too many words that only have few occurrences per word and that it is beneficial to exclude very short words from training. The bigger the output layer and the more occurrences are available per word, the better and the more robust the WuW detection performance.

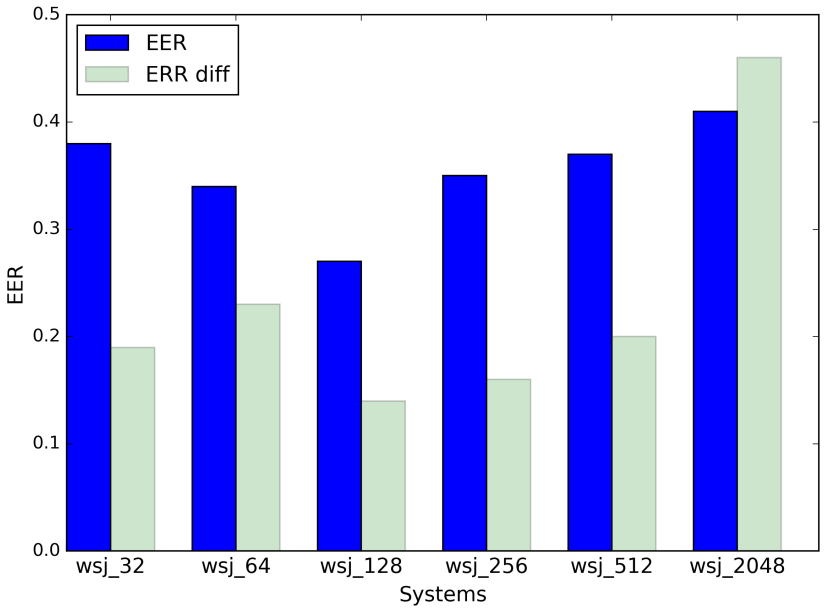


Figure 22: Performance of the systems with increased output layer size in terms of EER with (light green) and without (blue) differentiation. The system wsj_128 performs best. Systems with less and more output targets have rising error rates.

Table 9: Connection of EER with average occurrences per word and achieved WER. The bigger the output layer and the more occurrences are available per word, the better and the more robust the WuW detection performance. Letters in *italics* indicate averaged performances

System (# words)	**WER [%]**	**Avg. occur. per word**	**EER**	**diff. EER**
wsj_32	18.78	256.0	0.38	0.19
wsj_64	*23.54*	181.0	*0.34*	*0.23*
wsj_128	*30.39*	124.7	***0.27***	***0.14***
wsj_256	38.13	84.7	0.35	0.16
wsj_512	44.52	56.0	0.37	0.20
wsj_2048	80.89	22.0	0.41	0.46

10 RESULTS FOR REDUCED AVERAGE WORD LENGTH

In 8.1 *Data Selection*, we decided to restrict the training data to the most common N words and a minimum word length of 5 characters. In the previous chapter, we examined the influence of varying the number N, whereas in this chapter we included shorter words in the training data by setting the minimum word length to 3 characters. This has two important consequences: First, the network also learns to recognize shorter words, which might influence the WuW detection performance. Second, the number of average occurrences per word rises, because there are many short words that occur frequently.

10.1 DESCRIPTION OF TRAINED MODEL

For this experiment, we chose to use the 128 most common words for training as this model had previously performed best. However, we excluded some of the most occurring short words in order to obtain a similar range of average occurrences per word as in the wsj_64 system (approx. 80 - 500 occurrences). The resulting amount of training data is summarized and compared to wsj_64 and wsj_128 (with min. 5 characters) in table 10. Compared to wsj_128, it has the same number of different words, but more occurrences per word. The average word duration in wsj_128_3chars drops as expected due to the inclusion of shorter words.

Table 10 also shows the average WER over 4-5 training repetitions. Wsj_128_3chars performs slightly worse than wsj_128 which is surprising at first glance as more training data per word is available. However, the WER is defined per frame. The system might thus recognize the words at the same time frame or even earlier, but on average there are less correctly classified frames left due to the shorter words. This results in a lower WER. In light of these considerations, it is likely that for a word-based WER, wsj_128_3chars would outperform wsj_128.

Table 10: Training data overview and WER averaged over 4-5 training repetitions for the following systems: Two of them were trained on 64 and 128 longer words, whereas one was trained on 128 shorter words (wsj_128_3chars). The latter has a similar number of occurrences per word as wsj_64. Despite more training data, the achieved WER drops slightly due to its frame-wise definition.

System (# words)	**WER [%]**	**@ epoch**	**training hours [h]**	**Avg. occur. per word**	**Avg. word duration [s]**
wsj_64	23.97	55	1.21	181.0	0.38
wsj_128	30.69	85	1.71	124.7	0.38
wsj_128_3chars	31.88	67	2.06	184.0	0.31

10.2 WUW PERFORMANCE

In the WuW performance evaluation, wsj_128_3chars outperforms all previously reported systems with an averaged EER of 17%. Table 11 shows the achieved EERs per speaker for 4 repetitions of the training process as well as averages over the repetitions and for individual speakers. Regarding the training performance, the 4 repetitions lie within 1.2% WER (absolute), but like in the previous chapter, the WuW performance varies a lot between the repetitions: The corresponding EERs range from 8% for the best system to 30%.
For a given speaker, some systems still work better than others, but there is only one significant outlier with an EER above 32%. Figure 23 shows an example of the cosine similarity output for the best performing speaker m24 of the best system. The WuWs are clearly separated from the conversational speech in between.

Table 11: WuW performance per speaker for 4 repetitions of the training process. Averages for each speaker and each system are also given. Like before, the average results vary a lot between different speakers and systems. The best system achieves an EER of 8% which is better than all of the previously reported systems.

EER for wsj_128_3chars					
Speaker	**WH0**	**WH1**	**WH2**	**WH3**	**∅ /speaker**
m24	**0.00**	0.24	0.26	0.22	0.18
m26	0.05	0.05	0.22	0.01	**0.08**
w22	0.02	0.11	0.19	0.05	0.09
w31	0.07	0.22	0.16	0.02	0.12
w58	0.10	0.22	0.64	0.12	0.27
m25	0.25	0.29	0.32	0.18	0.26
total EER	**0.08**	0.19	0.30	0.10	0.17

Table 12 compares the WuW detection results of wsj_64, wsj_128 and wsj_128_3chars averaged over 4-5 repetitions of the training process. wsj_128_3chars clearly outperforms the other two systems with 17% absolute and 50% relative improvement for wsj_64 and 10% absolute and 37% relative improvement for wsj_128. The huge performance gain compared to wsj_64 confirms our assumption that training a model on more different words that can be learned equally well results in a more powerful acoustic model. The performance gain compared to wsj_128 underlines the importance of sufficient training data as an increase from 125 to 185 occurrences per word yields a significantly superior result. It would be necessary to further examine the influence of the shorter words in the training data of wsj_128_3chars on these improvements. Unfortunately, such experiments could not be conducted due to time constraints. However, we expect its impact to be less significant than the increased amount of training data.

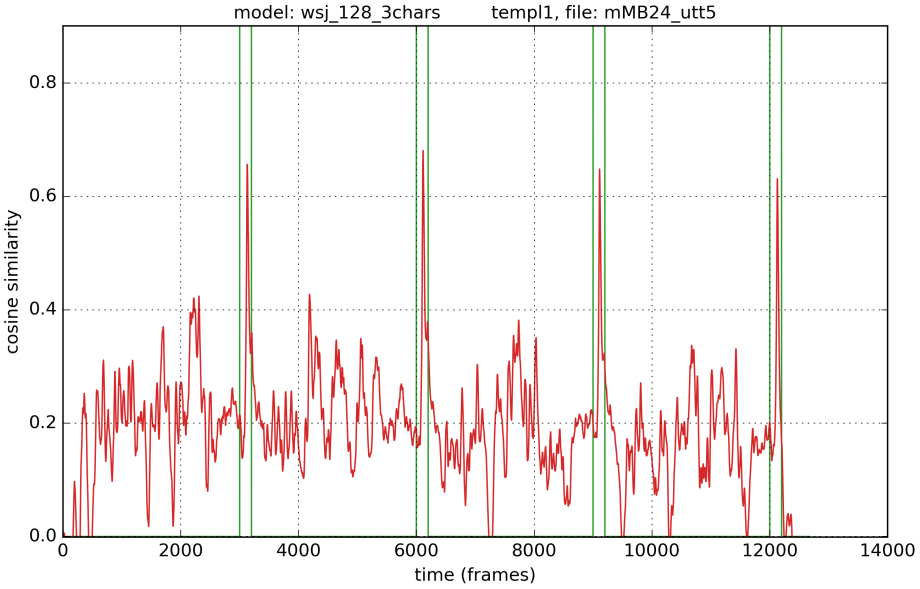


Figure 23: WuW detection with the best performing system wsj_128_3chars. This example from the evaluation corpus shows that the WuWs are clearly separable from the conversational speech in between.

Table 12: WuW performance comparison of the systems already shown in table 10. wsj_128_3chars outperforms both wsj_64 and wsj_128 with 17% absolute and 50% relative improvement for wsj_64 and 10% absolute and 37% relative improvement for wsj_128.

System (# words)	**WER**	**EER**	**Avg. occur. per word**
wsj_64	23.55	0.34	181.0
wsj_128	30.39	0.27	124.7
wsj_128_3chars	31.88	0.17	184.0

11 CONCLUSIONS AND OUTLOOK

11.1 CONCLUSIONS

We successfully re-implemented the approach presented by Chen et al. [CPS15] and recorded a corpus to evaluate the trained models regarding their WuW detection performance. The evaluation confirmed that WuW detection is possible even with extremely limited training resources of about 2 hours. Even though we trained on an English corpus, it appeared to be no problem to use a German WuW and evaluation corpus. For some systems, surprisingly good results with EERs as low as 8% could be achieved despite far-field conditions.
Our results indicate that training the models on 128 different words with about 180 occurrences per word is optimal for our setup. We found that using more different words with few occurrences is not beneficial. This is particularly important when working with small and medium-size databases.
Due to the extremely small training sets, the performance variations between individual speakers and repetitions of the training process were considerably high. However, a tendency that these variations vanish for well-performing systems was observed.
Further performance gains and increased robustness in real-world applications are expected if the network is trained on more different words with a sufficient number of occurrences.

11.2 OUTLOOK

The work presented in this thesis provides a solid basis for future improvements and further investigations. In the following points, we summarize the most important suggestions:

- For improved error rates and more robust real-world performance, the model needs to be trained on a larger speech database. Assuming that we want to train on 5000 words with 180 occurrences each and assuming an average word duration of 0.4 s, about 100 hours of appropriate speech data are required. Note that depending on the database, only parts of it can be used, if some words do not occur often enough or are too short.

- For increased robustness, it could be beneficial to also train with noisy or reverberated speech data. E.g., the parameters of a model, trained on a clean speech corpus, could be used to continue training with noisy or reverberated versions of the same data.

- The training process of LSTMs still requires some further investigations:
  The optimal training data preparation should be re-verified. In particular,

more detailed investigation of word-wise vs. sentence-wise training and the optimal minimum word length are necessary.
The spikes in the training process curves need to be examined and possible solutions such as proper gradient clipping should be found and evaluated.

- For more meaningful WuW performance results, the evaluation corpus should be expanded with more test persons and different WuWs. Additional investigations regarding performance degradation in near-field vs. far-field and in the presence of white and bubble noise could be performed.

BIBLIOGRAPHY

[BBB+10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun 2010.

[BFHC03] Yassine Benayed, Dominique Fohr, Jean-Paul Haton, and Gérard Chollet. Confidence measures for keyword spotting using support vector machines. *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, 1:I–588, 2003.

[BGA00] S E Bou-Ghazale and A O Asadi. Hands-free voice activation of personal communication devices. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, volume 3, pages 1735–1738 vol.3, 2000.

[Bis06] Christopher M Bishop. *Pattern Recognition*. 2006.

[Bot91] L Bottou. Stochastic Gradient Learning in Neural Networks. *Proceedings of Neuro-Nımes*, 91(8), 1991.

[BRS11] M S Barakat, C H Ritz, and D a Stirling. Keyword spotting based on the analysis of template matching distances. *Conference on Signal Processing and Communication Systems*, pages 1–6, 2011.

[CKSK11] Namgook Cho, Taeyoon Kim, Sangwook Shin, and Eun-Kyoung Kim. Voice activation system using acoustic event detection and keyword/speaker recognition. *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, pages 21–22, 2011.

[CPH14] G Chen, C Parada, and G Heigold. Small-footprint keyword spotting using deep neural networks. *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference*, (i):1–5, 2014.

[CPS15] G. Chen, C. Parada, and T.N. Sainath. Query-by-example keyword spotting using Long Short Term Memory Networks. *International Conference on Acoustics, Speech, and Signal Processing*, pages 1–5, 2015.

[dF15] Nando de Freitas. Machine Learning: 2014-2015, 2015. URL: https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/, [online; accessed 2016-04-07].

[Fal13] A.G Fallis. Efficient BackProp. *Journal of Chemical Information and Modeling*, 53; 9:1689–1699, 2013.

[FG01] Zheng Fang and Zhang Guoliang. Comparison of Different Implementations of MFCC. *Journal of Computer Science and Technology*, 16(6):582–589, 2001.

[FGS07] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. *The 17th international conference on Artificial neural networks*, pages 220–229, 2007.

[GC00] Felix a Gers and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural computation*, 12(10):2451–2471, 2000.

[GFGS06] Alex Graves, Santiago Fernandez, Faustino Gomez, and Jurgen Schmidhuber. Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks. *Proceedings of the 23rd international conference on Machine Learning*, pages 369–376, 2006.

[GLF+93] John S Garofalo, Lori F Lamel, William M Fisher, Johnathan G Fiscus, David S Pallett, and Nancy L Dahlgren. The DARPA TIMIT acoustic-phonetic continuous speech corpus CD-rom. NIST speech disc 1-1.1. *NASA STI/Recon Technical Report N 93*, 93, 1993.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech Recognition With Deep Recurrent Neural Networks. *ICASSP*, 3:6645–6649, 2013.

[Gra08] Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks. *Image Rochester NY*, page 124, 2008.

[Gra13] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, pages 1–43, 2013.

[GS00] F.A. Gers and J. Schmidhuber. Recurrent nets that time and count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 1:189–194 vol.3, 2000.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HSH+97] Sepp Hochreiter, Jürgen Schmidhuber, Sepp Hochreiter, Jürgen Schmidhuber, and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–80, 1997.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, pages 1–15, 2014.

[KGB09] Joseph Keshet, David Grangier, and Samy Bengio. Discriminative keyword spotting. *Speech Communication*, 51(4):317–329, 2009.

[KK09] V. Z. Këpuska and T. B. Klein. A novel Wake-Up-Word speech recognition system, Wake-Up-Word recognition task, technology and evaluation. *Nonlinear Analysis, Theory, Methods and Applications*, 71(12):e2772–e2789, 2009.

[KK14] Martin Kleinsteuber and Stefan Krüger. Sense of Hearing, 2014. URL: http://recognize-speech.com/speech/sense-of-hearing, [online; accessed 2016-04-04].

[KL14] Martin Kleinsteuber and Michael Lutter. Mel-Frequency Cepstral Coefficients, 2014. URL: http://recognize-speech.com/feature-extraction/mfcc, [online; accessed 2016-04-04].

[LCYK09] Hyeopwoo Lee, Sukmoon Chang, Dongsuk Yook, and Yongserk Kim. A voice trigger system using keyword and speaker recognition for mobile devices. *IEEE Transactions on Consumer Electronics*, 55(4):2377–2384, 2009.

[Lev83] Man Mohan Levinson, Stephen E and Rabiner, Lawrence R and Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, 1983.

[LYHG12] Jinyu Li, Dong Yu, Jui Ting Huang, and Yifan Gong. Improving wideband speech recognition using mixed-bandwidth training data in CD-DNN-HMM. *2012 IEEE Workshop on Spoken Language Technology, SLT 2012 - Proceedings*, pages 131–136, 2012.

[MR81] C. Myers and L. Rabiner. Connected word recognition using a level building dynamic time warping algorithm. In *ICASSP '81. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 6; 2, pages 951–955. Institute of Electrical and Electronics Engineers, 1981.

[O'D15] Jim O'Donoghue. Introducing: Blocks and Fuel – Frameworks for Deep Learning in Python, 2015. URL: http://www.kdnuggets.com/2015/10/blocks-fuel-deep-learning-frameworks.html, [online; accessed 2016-03-29].

[PB92] Douglas B Paul and Janet M Baker. The Design for the Wall Street Journal - based CSR Corpus. 1992.

[SKMR13] Tara N. Sainath, Brian Kingsbury, Abdel-rahman Mohamed, and Bhuvana Ramabhadran. Learning filter banks within a deep neural network framework. *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013*, pages 297–302, 2013.

[SP15] Tara N Sainath and Carolina Parada. Convolutional Neural Networks for Small-Footprint Keyword Spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[Uni14a] Université de Montréal. Welcome to Blocks' documentation, 2014. URL: https://blocks.readthedocs.org/en/latest/, [online; accessed 2016-03-29].

[Uni14b] Université de Montréal. Welcome to Fuel's documentation, 2014. URL: https://fuel.readthedocs.org/en/latest/, [online; accessed 2016-03-29].

[web16] Theano 0.8.0 documentation, 2016. URL: http://deeplearning.net/software/theano, [online; accessed 2016-03-29].

[WMSS11] Martin Wöllmer, Erik Marchi, Stefano Squartini, and Björn Schuller. Multi-stream LSTM-HMM decoding and histogram equalization for noise robust keyword spotting. *Cognitive Neurodynamics*, 5(3):253–264, 2011.

[WSR13] Martin Wöllmer, Björn Schuller, and Gerhard Rigoll. Keyword spotting exploiting Long Short-Term Memory. *Speech Communication*, 55(2):252–265, 2013.

[ZG09] Yaodong Zhang and James R. Glass. Unsupervised spoken keyword spotting via segmental DTW on Gaussian posteriorgrams. *Proceedings of the 2009 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2009*, pages 398–403, 2009.

[ZHP14] A Zehetner, M Hagmüller, and F Pernkopf. Wake-Up-Word Spotting for Mobile Systems. In *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European*, pages 1472–1476. IEEE, 2014.

[ZW95] David Zipser and Ronald J Williams. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. *Back-propagation: Theory, Architectures and Applications*, pages 433–486, 1995.