Master Thesis

Speech Recognition - A Transfer Learning Approach

Mapping Feature Representations of state-of-the-art Speech Recognition Frameworks

conducted at the
Signal Processing and Speech Communications Laboratory
Graz University of Technology, Austria

in co-operation with
LEFTSHIFT ONE Software GmbH
Graz, Austria

by
Raphael Schlüsselbauer, 01430197

Supervisors:
Dipl.-Ing. Dr.techn. Matthias Zöhrer

Assessors/Examiners:
Univ.-Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf

Graz, October 28, 2020

Acknowledgements

First, I would like to express my gratitude to my supervisor Matthias Zöhrer for his time, patience and constant feedback. He has helped me in many ways to complete this work and overcome problems when they occurred. Second, I would like to thank Franz Pernkopf for making this thesis possible and providing relevant literature. Third, I would like to thank my company Leftshift One Software GmbH and especially the CEO Patrick Ratheiser for funding and continued support during this thesis. Last, I would like to thank the system administrator of the Institute for Signal Processing and Speech Communication for helping me with setting up the experiments.

Abstract

Automated speech recognition (ASR) is of major importance as a hands free human-computer interface. Possible applications are voice controlled systems, dialog systems and documentation from dictation. Systems for the English language already have very low word error rates (WERs) due to large corpora being freely available. For the German language there seems to be too little free data available to train an ASR system with comparable accuracy. We suspect that German models with superior accuracy can be trained by leveraging English training data. This hypothesis is evaluated in this work.

Therefore, we use several ASR models: (i) a particular type of hidden Markov model hybrid, i.e. a HMM with a factorized time delay neural network (HMM/TDNN-F) (ii) a transformer network, (iii) the Wav2Letter and (iv) DeepSpeech 2 architecture, in a transfer learning ASR setup. Open source frameworks are used to compare the proposed architectures on the English speech dataset Librispeech and the German Mozilla Common Voice dataset. In particular, transfer learning models are initialized with parameters obtained with the English speech corpus and mapped to the German ASR models. In order to align subword representations we adapt the network's output layer to the vocabulary size and subword units of the German speech corpus. We measure the network's accuracy in terms of WER and character error rate (CER). The transformer architecture trained without a language model (LM) achieves the best WER on the Librispeech dataset, i.e. a WER of 4.9% on Librispeech test-clean was achieved. The same model trained on the German Mozilla Common Voice dataset reached a WER of 39.9%. Using a transfer learning setup including English speech this accuracy could be improved relatively by 16%. We conclude that the performance of German ASR models is improved significantly by using an English model as weight initialization in a transfer learning setup. This effect is stronger when little training data is available. ASR models only using connectionist temporal classification (CTC) reached WERs of 13.43% (Wav2Letter) and 29.39% (DeepSpeech 2) without a LM on the Librispeech corpus. This indicates that the attention mechanism of the transformer architecture is increasing accuracy and reducing the need for a LM. This paves the way for edge implementations, replacing memory demanding LMs.

When analyzing the computational performance of the selected architectures, we compared network inference time on the CPU of both ESPnet and Kaldi using an Intel Xeon E5-2697v3 @ 2.60GHz CPU. For the evaluation in terms of training and inference performance using GPUs an NVIDIA Tesla K40c GPU with 12GB VRAM was used. The Wav2Letter model had the fastest training time with 4.25 hours per training epoch on Librispeech. DeepSpeech2 had the fastest greedy decoding time on the GPU with 1.33 minutes for 1 hour of audio. Comparably the HMM/TDNN-F offered the fastest greedy decoding time on the CPU with 4.72 minutes for 1 hour of audio. Inference time highly depends on the choice of the LM size and beam size for decoding phoneme and character representations obtained by the acoustic models. The evaluation in terms of inference time exhibits that all evaluated models can decode audio faster than real time if the beam size of the decoder is sufficiently small. However, the evaluated models, have to be scaled down significantly in terms of memory and computational complexity to run on edge devices in realtime.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

date	(signature)

Contents

List of Figures

List of Tables

Acronyms

ANN	artificial neural network
ANNs	artificial neural networks
ASR	automated speech recognition
CER	character error rate
CNN	convolutional neural network
CNNs	convolutional neural networks
CTC	connectionist temporal classification
DNN	deep neural network
DNNs	deep neural networks
Mel-scaled filter-bank	Mel-scaled filter-bank
Mel-scaled filter-banks	Mel-scaled filter-bank
FFT	fast Fourier transform
GMM	Gaussian mixture model
GMMs	Gaussian mixture models
HMM	hidden Markov model
HMMs	hidden Markov models
LARS	layer-wise adaptive rate scaling
LER	label error rate
LM	language model
LMs	language models
LPCs	linear prediction coefficients
LSTM	long short-term memory network
LSTMs	long short-term memory networks
MFCC	Mel frequency cepstral coefficient
MFCCs	Mel frequency cepstral coefficients
ML	maximimum likelihood
MMI	maximum mutual information
PLPs	perceptual linear prediction

RNN	recurrent neural network
RNNs	recurrent neural networks
SVD	singular value decomposition
TDNN	time-delay neural network
TDNN-F	factorized time-delay neural network
TDNNs	time-delay neural networks
TTS	text-to-speech
WER	word error rate
WERs	word error rates
WFSTs	weighted finite state transducers

1 Introduction

1.1 History of Speech Recognition

This section provides an overview of the developments in the field of automated speech recognition (ASR) in the previous decades. Automated speech recognition has been an active field of research since the 1950s. Figure 1.1 illustrates a summary of the progress in a timeline.

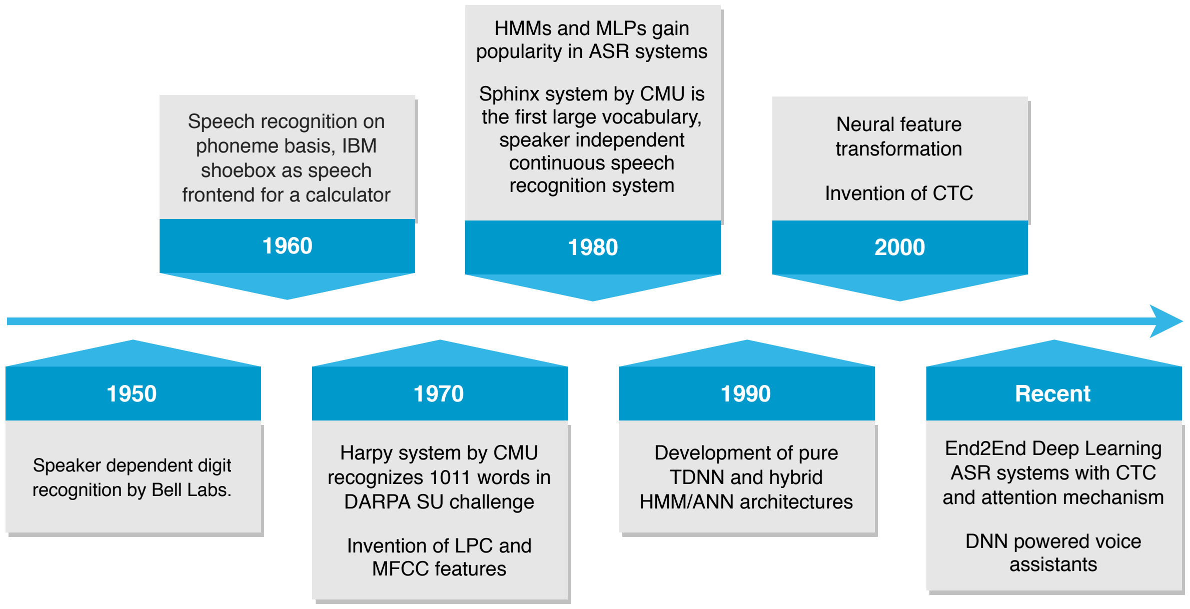

1950
Speaker dependent digit recognition by Bell Labs.
1960
Speech recognition on phoneme basis, IBM shoebox as speech frontend for a calculator
1970
Harpy system by CMU recognizes 1011 words in DARPA SU challenge
Invention of LPC and MFCC features
1980
HMMs and MLPs gain popularity in ASR systems
Sphinx system by CMU is the first large vocabulary, speaker independent continuous speech recognition system
1990
Development of pure TDNN and hybrid HMM/ANN architectures
2000
Neural feature transformation
Invention of CTC
Recent
End2End Deep Learning ASR systems with CTC and attention mechanism
DNN powered voice assistants

Figure 1.1: A timeline of progress in ASR.

1.1.1 1950s

One of the earliest attempts to do automated speech recognition was done by Homer W. Dudley et. al. [3]. They invented a sound printing mechanism that takes speech as input and analyzes the power spectrum of different frequency subbands using band-pass filters to print phonetic characters. In 1952, Davis et al. [4] from Bell Telephone Laboratories invented a speaker dependent recognizer for spoken digits. Recognition was done by performing pattern matching on the frequencies of the first and second formant of the utterance of a single digit.

1.1.2 1960s

Halle et al. [5] suggested a model for speech recognition in 1962 that makes use of an "analysis by synthesis" approach. Using a speech synthesizer phones are synthesized using rules and compared to the input. A control component is used to match phoneme combinations using the best matching phone sequence as a result. The model is only suitable for arbitrary sequences of phones seperated by silence as no language specific constraints are imposed on the output and no segmentation logic is used. In 1963, Sakai et al. [6] propose a mechanical speech recognition system with an unconstrained alphabet. By segmenting the data on a phonetic level they avoid relying on input constraints (e.g. constraining the input to a single word or phones seperated by silence) and can therefore recognize speech continuously.

1.1.3 1970s

The Hearsay system [7] introduced in 1973 aims to provide continuous speech recognition of multiple speakers with small delay. The main idea is to leverage knowledge from multiple sources to improve the recognition accuracy. A source of knowledge is any module that can produce and rate hypotheses, i.e. an acoustic recognizer, a syntactic recognizer, a semantic recognizer. Knowledge sources produce possible hypotheses given the input. It is assumed that errors happen at every analysis stage. Therefore, produced hypotheses can be rejected, accepted or reprioritized by an ensemble of knowledge sources. The system is modular so that sources of knowledge can be added and removed without breaking it. In 1975, Itakura [8] successfully used linear prediction coefficients (LPCs) as input features for dynamic time warping achieving speaker dependent recognition rates of 97.3% in a 200 word challenge. The Dragon system was introduced by Baker in 1975 [9] and was one of the first systems to build on hidden Markov models (HMMs) for recognition. In the 80s HMMs gained traction, were researched increasingly [10] [11] and are still used in state of the art ASR pipelines [12]. Lowerre developed the Harpy speech recognition system [13] in 1976 as an improved version of Hearsay I and Dragon. It was the best performing contribution to the 5 year ARPA SUR challenge [14] and could recognize a vocabulary of over 1000 words. It represents knowledge in a graph structure and performs speech recognition by doing a graph search. Harpy improves on Hearsay-I's and Dragon's search algorithms by using a beam search that decreases the search space by pruning hypotheses with a probability below a certain threshold. In 1976, Paul Mermelstein introduced Mel frequency cepstral coefficients (MFCCs) [15], an effective feature representation of voice signals that is still used in modern speech recognition systems [16, 17]. An example of MFCCs is shown in Figure 1.2 and the corresponding waveform is shown in Figure 1.3. In 1980, Davis et al. [18] showed the superiority of MFCCs compared to LPCs and linear frequency cepstrum coefficients in a speech recognition task leading to a rise in popularity of the technique.

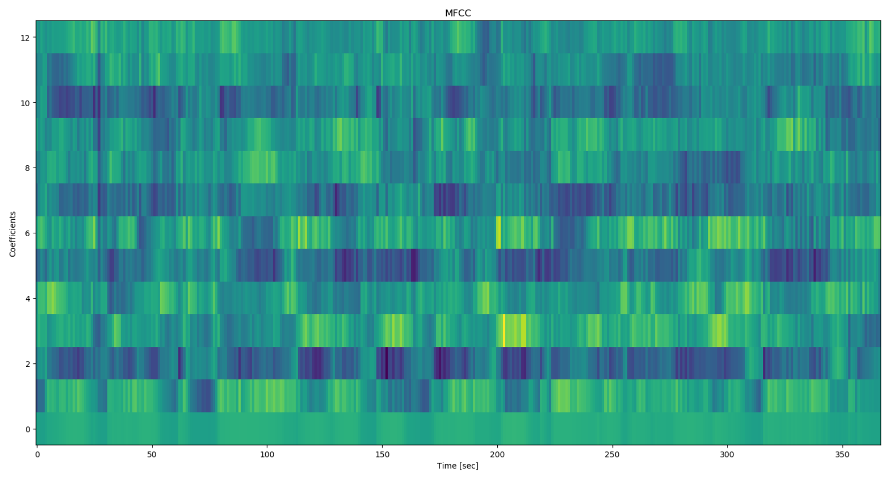


Figure 1.2: MFCC features of the sentence "The quick brown fox jumps over the lazy dog".

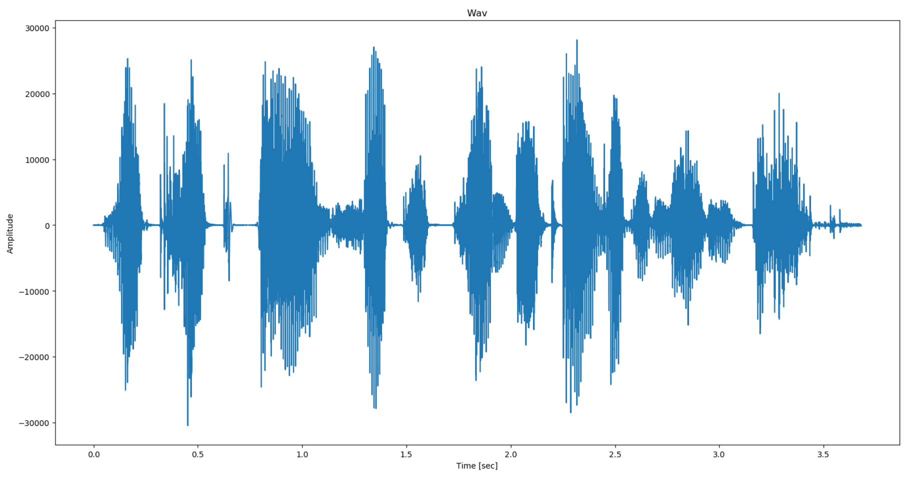


Figure 1.3: Raw waveform of the sentence "The quick brown fox jumps over the lazy dog" sampled at 44100Hz.

1.1.4 1980s

Although HMMs have been used in speech recognition by Baker et al. [9] and Jelinek et al. [19], HMMs only gained popularity in the early 1980s. HMMs model the variability of speech as well as pronunciation [20]. In particular, algorithms like the Baum-Welch algorithm (cf. Section 2.2.1) for training the parameters of HMMs and the Viterbi algorithm, an algorithm finding the most probable state sequence given a model and an observation sequence [21], are powerful tools to improve speed and accuracy. In particular, the Baum-Welch algorithm solves the alignment problem, a central problem in speech recognition described in detail in Section 2.2. Another field of research in the late 80s were artificial neural networks (ANNs). Speech recognizers heavily relied on high level language models (LMs) [22]. Therefore, Lippmann concluded there was need for improvement of low level feature matching. Neural networks have been compared to conventional approaches in constrained settings like classification of three different phonemes [23] or classification of 9 similiar letters of the alphabet [24]. Speech recognition systems at the time worked because they imposed constraints on one or more of the following factors [25]: (i) Speaker independence, (ii) vocabulary size, (iii) continuous speech and (iv) grammar perplexity. Grammar perplexity describes the average number of possible word choices at any point by a language model. With the SPHINX system, Lee et al. [26] introduced the first successful large vocabulary, speaker independent continuous speech recognition system that relaxed all the aforementioned constraints. At the time it had the best speaker independent results for the DARPA 997 word resource management task with accuracies of 71%, 94% and 96% at grammar perplexities 997, 60 and 20 [25]. The SPHINX system uses HMMs for phone, word and sentence models, is trained with the Baum-Welch algorithm and decoded with a Viterbi beam search. In 1986, Bahl et al. [27] suggest training HMM parameters to maximize mutual information MMI between audio sequence and transcription sequence instead of using a maximimum likelihood (ML) criterion and training parameters that maximize the probability of observing the training audio sequences. In a speaker-dependent word recognition task the proposed MMI criterion performed 18% better than the classic ML criterion. The method can only be applied to pretrained models and has a lot of hyperparameters that need to be tuned [28].

1.1.5 1990s

Typically HMM based systems used GMMs as acoustic model. Seperate GMMs are trained per phone and can calculate the probability of audio features representing the phone modeled by the GMM. In ASR GMM-HMMs suffer from several shortcomings [29] such as poor discrimination between GMMs due to maximum likelihood training, assumption that speech is a first order Markov chain of audio frames and assumptions that phones can be represented by GMMs. Therefore, researchers tried to use ANNs as ASR framework instead of HMMs. In particular, the ANNs used were time-delay neural networks (TDNNs) [30] and recurrent neural networks (RNNs) for their ability to model sequences. Examples of pure ANN architectures are Alpha nets [31] and Viterbi nets [32] that emulated HMM behaviour, but did not improve on it. Hybrid HMM/ANN approaches were proposed in the 90s to combine the discrimination capabilities of ANNs with the solution to the alignment problem the HMM provides [33, 34]. In these examples ANNs are used as an estimator for emission probabilities of the HMM. In 1997, Juang et al. propose a substantial improvement to statistic methods for pattern matching in speech recognition [35]. They argue that the widely used Bayes theorem is not optimal for speech recognition, because it is required to estimate a distribution from limited data which leads to suboptimal results. Instead they suggest using classification error rate as a cost function and minimizing it. The resulting algorithm is called generalized probabilistic descent and produces 30-50% lower error rate (relative) than the commonly used ML classifier. In 1997 Hochreiter and Schmidhuber layed the ground work for modern deep learning approaches with the invention of long short-term memory networks (LSTMs) [36]. Long short-term memory networks are RNNs that do not suffer from the vanishing gradient problem [37] and can therefore make use of long term dependencies in sequences necessary for speech recognition.

1.1.6 Recent developments

Hermansky et al. [38] publish an alternative approach to hybrid HMM/ANN systems in 2000 that improves the error rates on the Aurora [39] noisy continuous digits task by 35%. Compared to common hybrid systems that use a neural network as acoustic models they feed feature vectors into a neural network and use the resulting vectors as input for a GMM acoustic model. The invention of connectionist temporal classification (CTC) by Graves et al. [40] in 2006 provided an alternative to HMMs for solving the alignment problem in ASR. With CTC it is possible to train an RNN on unsegmented and unaligned data paving the way for end-to-end deep learning ASR solutions [41–43].

In 2011, Povey et al. [16] released the Kaldi speech recognition toolkit based on weighted finite state transducers (WFSTs) [44] with support for deep neural network (DNN) acoustic models and predefined recipes for ASR datasets. The toolkit is still very popular in the research community for hybrid HMM/DNN architectures [45–47] and achieves competitive results. However, conventional systems have some drawbacks that researches are trying to avoid by building end-to-end ASR systems: They need to be trained in multiple steps instead of one streamlined procedure with a single training objective [48]. When training a new DNN acoustic model a HMM with GMMs needs to be trained first to obtain initial alignments. HMM/DNN systems also need linguistic information that can introduce errors like hand engineered phonetic dictionaries and phonetic context decision trees. Another drawback is that conventional systems assume conditional independence of the current frame and its label given the previous input frames, which is not true for ASR [48]. The different modules in such a system normally do not share the same objective function that is optimized resulting in modules that do not fit together perfectly.

In 2012, Graves introduced the RNN Transducer [41] that extends CTC by a prediction network, that calculates the probability of an output based on all previous outputs analogous to a language model (LM) integrated into the model architecture. In 2015, Chorowski et al. [49] successfully applied an encoder-decoder architecture with attention mechanism to speech recognition as an alternative to CTC. The encoder takes audio frames and generates a sequence of feature vectors. At each decoding step the attention mechanism generates a weighted sum over all outputs of the encoder. This sum and the decoder state is then used to generate the transcription. Compared to CTC, the encoder-decoder with attention mechanism does not rely on the conditional independence assumption [50], but it is unsuitable for realtime decoding because it needs to see the whole input sequence before decoding [51]. Since then new technologies have been proposed to apply attention in a streaming online ASR setting [51–53]. The transformer architecture introduced by Vaswani et al. [54] improves on encoder-decoder with attention by removing RNNs from the architecture and therefore allowing increased parallelization. It produces state of the art results in neural machine translation tasks and was successfully applied to ASR by Zhou et al. [55].

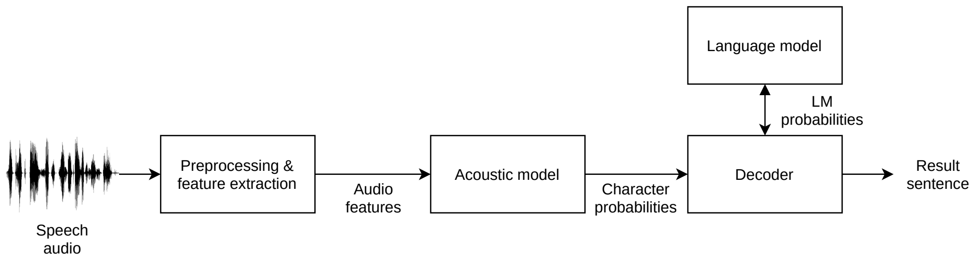


Figure 1.4: Diagram of a conventional ASR pipeline.

Figure 1.4 shows the modules of a conventional ASR pipeline, that can also be found in many end-to-end ASR systems. Firstly, the raw audio is preprocessed with noise removal, voice activity detection and pre-emphasis for higher frequencies [56]. Secondly, it is split into short overlapping frames (about 20ms) and a window function might be applied to emphasise a specific part of the frame and reduce discontinuity at beginning and end of the frame. Finally, the audio frames are mapped from the time domain to the time frequency domain using a discrete Fourier transform [57]. An example of the resulting spectogram can be seen in Figure 1.5.

Features that compress the essential information of the preprocessed audio are extracted with one of various feature extraction techniques including MFCCs (cf. Section 2.1.1), Mel-scaled filterbanks (cf. Section 2.1.1) or perceptual linear prediction (PLPs) [58] features. These features of audio frames are then fed to the acoustic model to get a probability distribution over a unit of transcription that can include graphemes, subword units or (context dependent) phonemes. The

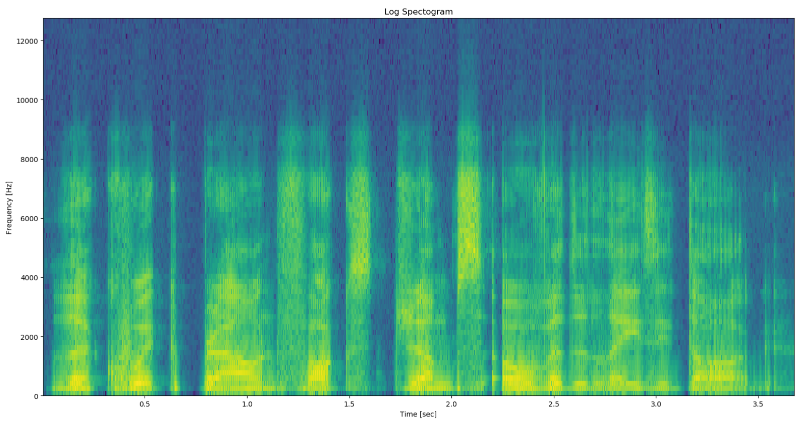


Figure 1.5: Log-Spectogram of the sentence "The quick brown fox jumps over the lazy dog".

acoustic model is responsible to discriminate between the possible outputs regardless of background noise and interference. Acoustic model architectures include GMMs, RNNs, TDNNs, long short-term memory networks (LSTMs), sequence-to-sequence models with attention, transformers and convolutional neural networks (CNNs). At training time ASR systems can either use HMMs with Baum-Welch algorithm (cf. Section 2.2.1), connectionist temporal classification (cf. Section 2.2.2) or an attention mechanism (cf. Section 2.2.3) for solving the alignment problem (cf. Section 2.2). At runtime the system produces a probability distribution over the units of transcription for each frame. These can then be decoded with a suitable algorithm including greedy decoding or beam search decoding [40]. The decoder can make use of a LM to restrict the output to predefined words and grammar. Popular options are n-gram [59], LSTM [12], gated CNNs [60] and transformer [12] LMs. Another option for decoding is multi pass decoding where initial transcription candidates, e.g. word lattices [61], are selected with the acoustic model and an efficient LM and rescored with larger LMs in the following pass(es).

1.2 Scope of Thesis

The scope of this thesis consists of the following goals:

1. Performance evaluation of different ASR architectures in terms of WER using two speech corpora, i.e. Librispeech (cf. Section 4.1.1), an English read speech corpus with 940 hours and the German Mozilla Common Voice corpus (June 2019) (cf. Section 4.1.2), a German read speech corpus with 324 hours. Librispeech was selected, because it is commonly used in recent publications, is freely available and has sufficient clean data to provide good results [12, 62]. The German Mozilla Common Voice corpus was selected, because it is the largest freely available German speech corpus. In particular, the evaluated architectures are HMM/TDNN-F, Wav2Letter, DeepSpeech 2 and an ESPnet transformer introduced in more detail in Chapter 3. They were selected based on their recency, good performance and existence of an implementation in open source frameworks.

2. Performance evaluation in terms of WER of a transfer learning setup based on transformer architecture trained on the German Mozilla Common Voice corpus and initialized with the model trained on the Librispeech corpus.

3. Performance evaluation in terms of training and inference speed of the open source ASR frameworks used in (1), i.e. Kaldi, OpenSeq2Seq and ESPnet (cf. Section 3.1).

1.3 Outline of Thesis

This thesis is divided into 5 Chapters.

- In Chapter 2 we provide an overview of the mathematical concepts used in ASR and the ASR systems discussed in this thesis.
- Chapter 3 describes several ASR system architectures in detail. Furthermore, information on implementation frameworks of those architectures is provided.
- Chapter 4 is divided into three parts. Firstly, it provides information on the speech corpora Librispeech and German Mozilla Common Voice. Secondly, it describes the experimental evaluation in terms of WER of ASR architectures selected in the previous Chapter with the Librispeech corpus, the German Mozilla Common Voice corpus and a transfer learning approach. Finally, it compares the inference and training time of each of the ASR architectures as well as features and usability of the selected frameworks.
- Chapter 5 provides a conclusion and future outlook of the thesis.

1.4 Contributions

We provide a performance evaluation in terms of word error rate (WER) of acoustic models with architectures HMM/TDNN-F, Wav2Letter, DeepSpeech 2 and ESPnet transformer on the 960h Librispeech corpus, including a transfer learning setup using an English transformer model as initialization for a German ASR task. In particular, the best achieved WER for the Librispeech corpus without LM is 4.9% on test-clean with the transformer architecture. We show that transfer learning from an English model provides a major accuracy boost when training the transformer architecture on the German Mozilla Common Voice corpus and report the results. In particular, WERs of 37.6% and 33.5% are achieved for the official Mozilla Common Voice split and a larger custom split (cf. Section 4.1.2) which is a relative improvement of 37.5% and 16% over models trained without transfer learning. Finally we provide a comparison of the frameworks and network architectures that were used based on training and inference speed. In particular, we report that Wav2Letter has the fastest training time, DeepSpeech 2 provides the fastest GPU inference and the Kaldi HMM/TDNN-F provides the fastest CPU inference with a common setup.

2 Mathematical Background

2.1 End-to-End Speech Recognition

Due to recent technological advances and availability of large amounts of data it has become possible to reduce complexity of earlier ASR systems and learn speech recognition end-to-end in a single DNN with performance comparable to the state-of-the-art. While HMMs have proven effective for speech recognition an end-to-end system can reduce the complexity of preprocessing, acoustic modeling, building LMs, special handling of out of vocabulary words and the need for human expertise for phonetic dictionaries in HMM based systems. ANNs have already improved HMM performance [29] in hybrid HMM/ANN approaches. In a hybrid HMM/ANN the ANN is used for classifying audio frames and trained to minimize frame classification error. However, minimizing frame classification error does not directly translate to minimized labelling error [28] and is therefore not as effective. End-to-end ASR networks improve on the shortcomings of HMM/ANN hybrids by training acoustic, alignment and language model end to end without the need for multi step procedures or hand engineered features like phonetic dictionaries. They optimize the sequence label error instead of frame label error and therefore maximize the probability of correct sequences. In HMM/ANN hybrids, the HMMs are trained with the Baum-Welch algorithm to find proper alignments for the given data. End-to-end ASR systems replace this component with one of the following: (i) connectionist temporal classification (CTC), (ii) attention mechanism or (iii) hybrid attention/CTC. In the following we describe the components used in a state-of-the-art end-to-end ASR system. In particular, we highlight feature pre-processing, acoustic model, LM and alignment algorithms for finding the most probable state-sequence for speech audio given a set of training examples.

2.1.1 Preprocessing

Feature extraction plays an important role in ASR due to the high dimensionality of raw waveform data. It is possible for DNNs to learn a feature representation directly from raw waveform [63–65], but in terms of accuracy and training time they can still profit from conventional feature representations. Popular feature representations are Mel-scaled filter-bank features and MFCCs [18] that were originally used in combination with HMM/GMM hybrids.

Log Mel-scaled filter-bank features

Log Mel-scaled filter-banks [66] are a common feature representation and the basis for MFCCs [18]. They are short term features based on the frequency spectrum and scaled by the Mel scale [67], a scale for measuring pitch perceived by the human ear. The Mel scale is a logarithmic scale that compresses higher frequencies that cannot be distinguished by humans easily. It is used to maintain the most information relevant for speech and compress the remaining information accordingly. To create Mel-scaled filter-banks the audio is first split into short frames that usually have an overlap. In this example frames of length $25ms$ with a stride of $10ms$ are used, which is a common choice [66, 68]. Then a Hamming window function is applied to each audio frame smoothing the signal at the edges. Given a frame with N samples a Hamming window of the same size is defined as

$$\omega(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \qquad \forall n : 0 \leq n \leq N-1 \ . \tag{2.1}$$

An example of a Hamming window function for 100 samples is shown in Figure 2.1. The smoothed

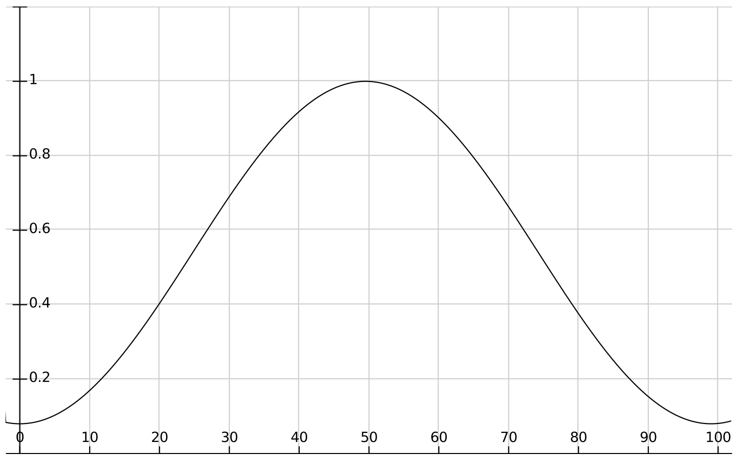


Figure 2.1: Plot of the Hamming window function for a total number of $N = 100$ samples.

frames are transformed from the time domain to the frequency domain with a fast Fourier transform (FFT) [69] resulting in a frequency spectrum for each audio frame. To preserve information important for human speech perception we apply 40 Mel scaled triangular bandpass filters to the frequency spectrum. The width of the filters is evenly spaced in the Mel domain and then transformed back to the Hz domain. The Mel value $m(f)$ of a Hz value f is calculated as

$$m(f) = 2595\log_{10}\left(1 + \frac{f}{700}\right) \ . \tag{2.2}$$

The resulting 40 dimensional vector is our Mel-scaled filter-bank feature vector of which a log Mel-scaled filter-bank vector can be obtained by taking the logarithm. This vector characterizes the speech audio and can be used for ASR applications. However, its values are correlated, because the triangular bandpass filters overlap. Therefore log Mel-scaled filter-bank features can be processed further to obtain decorellated MFCC features.

Mel-Frequency Cepstral Coefficients

Since their introduction by Davis and Mermelstein in 1980 [18], MFCCs have been a dominant feature representation for speech in ASR tasks [70]. They are decorrelated short term features based on log Mel-scaled filter-banks. MFCCs are calculated by taking the discrete cosine transform of the log Mel-scaled filter-bank feature vector. A common choice for the number of coefficients N is 13 as is the default in Kaldi [16]. Given a log Mel-scaled filter-bank vector $\boldsymbol{x}$ with size K an MFCC $\boldsymbol{c}_i$ is calculated by

$$\boldsymbol{c}_i = \sum_{k=1}^{K} \boldsymbol{x}_k cos\left(i \cdot \left(k - \frac{1}{2}\right)\frac{\pi}{20}\right) \ , \qquad i = 1, ..., N \ . \tag{2.3}$$

The first coefficient is usually discarded, because its information is irrelevant to ASR [71]. Coefficients 2-13 are combined in an acoustic vector for each frame as displayed in Figure 2.2 and can be used for ASR. MFCCs only describe the current speech frame, therefore delta features

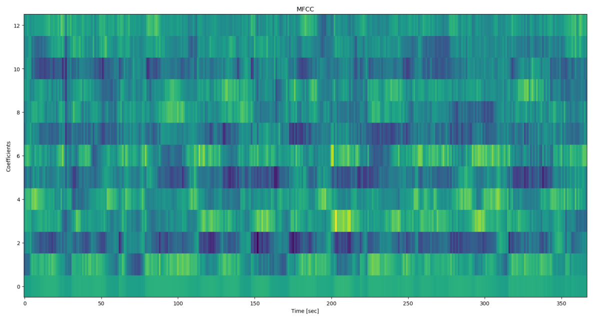


Figure 2.2: Plot of MFCCs of a speech signal.

can be added for each MFCC to describe the change of the coefficients over time [72,73]. A delta feature vector $\boldsymbol{d_t}$ for a frame at timestep t is given by

$$\boldsymbol{d_t} = \frac{\sum_{n=1}^{N} n(\boldsymbol{c}_{t+n} - \boldsymbol{c}_{t-n})}{2\sum_{n=1}^{N} n^2}, \tag{2.4}$$

where $\boldsymbol{c_t}$ is the MFCC vector at timestep t and N is the number of context vectors used to compute the delta feature. Das et. al. [73] recommend setting N to 2, but it can be tuned in the range of positive integers. Additionally, the MFCCs' acceleration (delta-delta features) can be calculated by performing the delta calculation given in Equation 2.4 and replacing the MFCC vector $\boldsymbol{c_t}$ with the delta vector $\boldsymbol{d_t}$. Adding delta and delta-delta features to MFCCs results in a 36 dimensional feature vector that contains frame and temporal information. Furthermore, usually the log energy, the delta log energy and the delta-delta log energy is considered. This results in a total of 39 features for each speech frame.

2.1.2 Acoustic Model

The acoustic model helps with calulating $\text{argmax}_Y P(Y|X)$, the most likely label sequence $Y = (y_1, ..., y_N)$ for an input sequence $X = (\boldsymbol{x_1}, ..., \boldsymbol{x_T})$ of preprocessed audio features. Elements of the label sequence can be, but are not limited to context dependent or independent phones, graphemes and subwords. Using Bayes theorem we can rewrite the problem of finding the most likely label sequence Y^* as

$$Y^* = \underset{Y}{\text{argmax}}\, P(Y|X) = \underset{Y}{\text{argmax}} \frac{P(X|Y) \cdot P(Y)}{P(X)}. \tag{2.5}$$

Furthermore, the term $P(X)$ can be dropped, because it only scales the probability label sequence Y resulting in

$$Y^* = \underset{Y}{\text{argmax}}\, P(Y|X) = \underset{Y}{\text{argmax}}\, P(X|Y) \cdot P(Y)\ . \tag{2.6}$$

For the acoustic model we can differentiate between generative and discriminative modeling [74] of acoustic features. A generative model only models $P(X|Y)$ and additionally uses a LM $P(Y)$ in order to calculate $P(Y|X)$ with Bayes theorem as shown in Equation 2.6. A discriminative model represents $P(Y|X)$ directly as it is usually done in DNN based models. The output of the acoustic model is used to generate transcription hypotheses that can be rescored by a LM. In conventional systems the acoustic model consists of GMMs or DNNs combined with HMMs for alignment. In end-to-end deeplearning models the acoustic model is typically a DNN architecture with CTC or attention mechanism as alignment model (cf. Section 2.2).

2.1.3 Language Model

A language model scores the likelihood of a sequence of words occuring in an utterance of a language. The goal is to have a system that computes $P(Y)$ of a word sequence $Y = (y_1, y_2, ..., y_n)$. Transcription hypotheses generated from the output of the acoustic model can be scored with the weighted probability given by the LM. In this manner implausible word sequences get a lower rating. It also helps with disambiguation of similiar sounding word sequences, if one is more plausible than the other according to the LM. In case of a generative acoustic model, the score of the LM is used to calculate the probability of a transcription sequence given the input as described in Section 2.1.2. A popular LM choice is KenLM [75]. It is a time and memory efficient implementation of an n-gram language model that is used in multiple recent ASR frameworks including OpenSeq2Seq [76] and Wav2Letter++ [77].

2.2 Solutions to the Alignment Problem

The goal of the acoustic model is to transcribe a sequence of audio frames to a sequence of labels. Finding the most probable mapping between input frames and elements of the label sequence is known as the alignment problem. An alignment of audio segments to characters is shown in Figure 2.3. For training an ASR system it is necessary to know the alignment of training audio

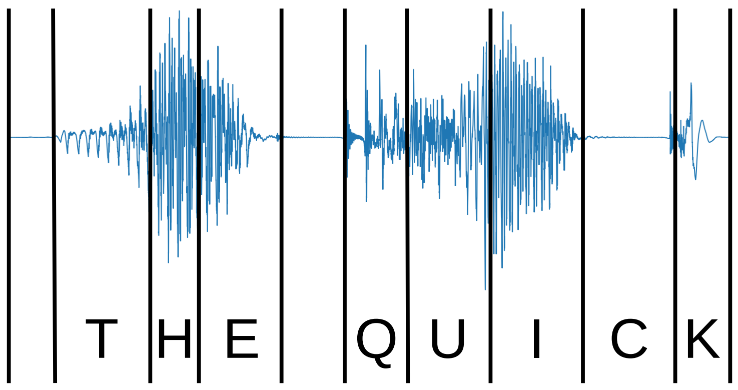


Figure 2.3: Example of a transcription aligned with the corresponding audio segments on the character level.

with its labels [63]. Manually annotating every letter or phoneme in a dataset is a lot of work and therefore often not feasible. Therefore, large datasets such as LibriSpeech [2] provide short audio files with transcription without alignment. Multiple approaches have been developed to learn alignments from data and they will be described in more detail in the sequel.

2.2.1 Baum-Welch algorithm

We define a HMM as $\Theta = (A, B, \boldsymbol{\pi})$ where A is a matrix of transition probabilities between a set of states S, B is a matrix of emission probabilities of the hidden states and $\boldsymbol{\pi}$ is a vector of the initial state distribution [78]. To model an ASR problem with HMMs we assume that the emissions are feature vectors of the audio frames and the hidden states are their corresponding labels. Following that scheme a hierarchy of phone, subword, word and sentence HMMs can be constructed by synthesis of sub-unit HMMs. Finding the correct labels for an utterance is then a problem of finding the most probable state sequence Q^* given an observation sequence, also known as decoding. Formally we want to calculate [78]

$$Q^* = \underset{Q}{\operatorname{argmax}}\, P(Q|X,\Theta) \tag{2.7}$$

where $Q = (q_1, ..., q_T)$ is a state sequence, $X = (x_1, ..., x_T)$ is an observation sequence and Θ are the parameters of the HMM. The solution to this problem can be calculated using the Viterbi algorithm [78]. We have previously made the assumption that we already have a HMM that has the correct parameters to perform speech recognition by decoding a frame sequence. This leads us to the problem of finding parameters that maximize the probability of observing given training sequences also known as training.

Training is defined by Rabiner [78] as finding HMM parameters $\hat{\Theta}$ according to

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmax}}\, P(X|\Theta) \tag{2.8}$$

for an observation sequence X for several sequences. For this problem there is no formal solution for global optimization [78]. However, it is possible to compute a local optimum iteratively with the Baum-Welch algorithm [79], an algorithm based on the principle of expectation maximization [80]. The Baum-Welch algorithm consists of two steps: (i) Calculation of forward and backward probabilities and (ii) parameter update. In the forward step the forward probabilities $\alpha_n(j)$ are calculated

$$\alpha_n(j) = P(x_1, ..., x_n, q_n = s_j|\Theta), \tag{2.9}$$

where $\alpha_n(j)$ is the probability of observing the first n elements of the observation sequence X and being in state s_j at observation step q_n given the parameters of the HMM. They are calculated recursively with a dynamic programming algorithm (known as Forward algorithm):

$$\alpha_1(j) = \pi_j \cdot b_{j,x_1} \quad \forall j = 1, ..., N_s \tag{2.10}$$

$$\alpha_n(j) = \left(\sum_{i=1}^{N_s} \alpha_{n-1}(i) \cdot a_{i,j} \right) \cdot b_{j,x_n} \quad N \geq n > 1 \; \forall j = 1, ..., N_s \tag{2.11}$$

$$P(X|\Theta) = \sum_{j=1}^{N_s} \alpha_N(j) \tag{2.12}$$

For the first observation the forward probabilities are initialized in Equation 2.10 where π_j is the initial probability of being in state s_j and b_{j,x_1} is the probability of emitting the output x_1 in state s_j. The rest of the forward probabilities can be calculated recursively as described in Equation 2.11 where $a_{i,j} = P(q_n = s_j|q_{n-1} = s_i)$ is the transition probability. Figure 2.4 shows how the forward probability depends on the previous forward probabilities of all states, their transition probability to the current state and the emission probability in the current state.

In the backward step we calculate the backward probabilities formally expressed as

$$\beta_n(i) = P(x_{n+1}, ..., x_N|q_n = s_i, \Theta), \tag{2.13}$$

where $\beta_n(i)$ is the probability of observing the remaining emission sequence $x_{n+1}, ..., x_N$ condi-

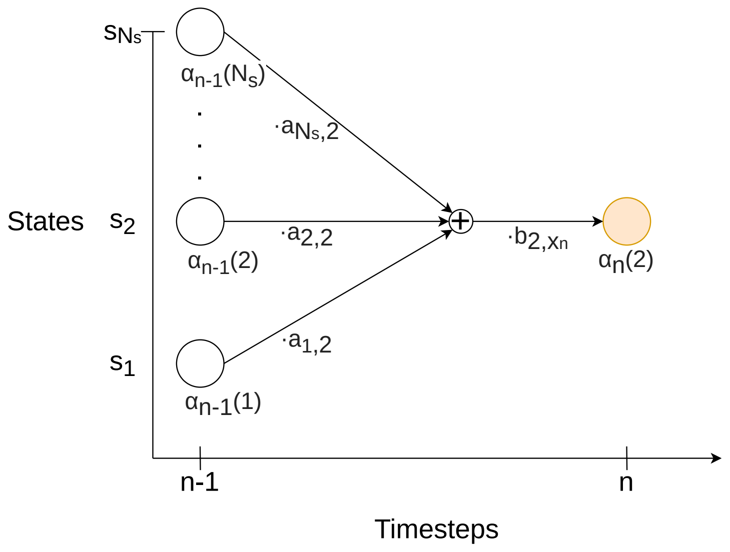


Figure 2.4: The forward probability $\alpha_n(2)$ is the product of the previous forward probability $\alpha_{n-1}(i)$ and the corresponding transition probability $a_{i,2}$ summed over all states i times the emission probability b_{2,x_n} of x_n in state 2.

tioned on being in state s_i at timestep n and the parameters Θ of the HMM.

$$\beta_N(i) = 1 \quad \forall i = 1, ..., N_s \tag{2.14}$$

$$\beta_n(i) = \sum_{j=1}^{N_s} a_{i,j} \cdot b_{j,x_{n+1}} \cdot \beta_{n+1}(j) \quad 1 \leq n < N \; \forall i = 1, ..., N_s \tag{2.15}$$

$$P(X|\Theta) = \sum_{j=1}^{N_s} \pi_j \cdot b_{j,x_1} \cdot \beta_1(j) \tag{2.16}$$

In Equation 2.14 the backward probabilities $\beta_N(i)$ for timestep N are initialized with 1 regardless of state s_i. This can be explained by the probability of observing nothing when being in the last state q_N being equal to 1 regardless of the state. Then the backward probabilities $\beta_n(i)$ of observing $x_{n+1}, ..., x_N | q_n = s_i, \Theta$ are calculated recursively back in time using the previous β_{n+1} values. Specifically the transition probability $a_{i,j}$ of moving from state s_i to s_j is multiplied by $b_{j,x_{n+1}}$, the probability of emitting x_{n+1} in state s_j and the previous backward probability in state s_j $\beta_{n+1}(j)$. This is done for all possible next states and the resulting sum is the backward probability $\beta_n(i)$ as illustrated in Figure 2.5. For any timestep n and any state s_j

$$P(X, q_n = s_j|\Theta) = P(x_1, ..., x_n, q_n = s_j|\Theta) \cdot P(x_{n+1}, ..., x_N|q_n = s_j, \Theta) \tag{2.17}$$

$$= \alpha_n(j) \cdot \beta_n(j) \tag{2.18}$$

is the probability of observing X and being in state s_j at timestep n. Marginalizing over all states we get

$$P(X|\Theta) = \sum_{j=1}^{N} \alpha_n(j) \cdot \beta_n(j) \tag{2.19}$$

the probability of observing sequence X for any timestep n.

In the update step we use the forward and backward probabilities to calculate new HMM

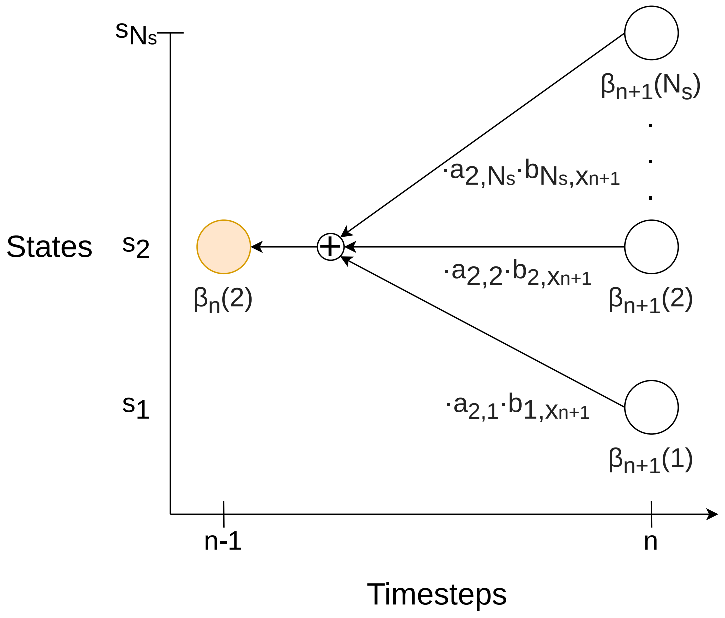

s_{N_s}
$\beta_{n+1}(N_s)$
$\cdot a_{2,N_s}\cdot b_{N_s,x_{n+1}}$
States
s_2
$\beta_n(2)$
$\cdot a_{2,2}\cdot b_{2,x_{n+1}}$
$\beta_{n+1}(2)$
s_1
$\cdot a_{2,1}\cdot b_{1,x_{n+1}}$
$\beta_{n+1}(1)$
n-1
n
Timesteps

Figure 2.5: The backward probability $\beta_n(2)$ is the product of transition probability $a_{2,i}$, emission probability $b_{i,x_{n+1}}$ and backward probability $\beta_{n+1}(i)$ summed over all possible next states i.

parameters that maximize $P(X|\Theta)$. Therefore, we define

$$\gamma_n(i) := P(q_n = s_i|X,\Theta) = \frac{P(q_n = s_i, X|\Theta)}{P(X|\Theta)} = \frac{\alpha_n(i)\beta_n(i)}{P(X|\Theta)} = \sum_{j=1}^{N_s} \xi_n(i,j) \tag{2.20}$$

$$\xi_n(i,j) := P(q_n = s_i, q_{n+1} = s_j|X,\Theta) = \frac{P(q_n = s_i, q_{n+1} = s_j, X|\Theta)}{P(X|\Theta)} = \\ = \frac{\alpha_n(i)\cdot a_{i,j}\cdot b_{j,x_{n+1}}\cdot \beta_{n+1}(j)}{P(X|\Theta)} \tag{2.21}$$

The probability γ_n of being in state s_i at timestep n conditioned on an observation sequence X and the parameters of the HMM Θ is calculated in Equation 2.20. The function $\xi_n(i,j)$ defined in Equation 2.21 is the probability of transitioning from state s_i to s_j at timestep n conditioned on an observation sequence X and the parameters of the HMM Θ. The parameters are updated according to:

$$\bar{\pi}_i = \gamma_1(i) \tag{2.22}$$

$$\bar{a}_{i,j} = \frac{\sum_{n=1}^{N-1} \xi_n(i,j)}{\sum_{n=1}^{N-1} \gamma_n(i,j)} \tag{2.23}$$

$$\bar{b}_{j,k} = \frac{\sum_{n=1}^{N} \gamma_n(j)\cdot 1_{[x_n=v_k]}}{\sum_{n=1}^{N} \gamma_n(j)} \tag{2.24}$$

The initial state distribution π_i is updated with $\bar{\pi}_i$, the probability of being in state i at timestep 1 (cf. Equation 2.22). Transition probability $a_{i,j}$ is updated with $\bar{a}_{i,j}$, the summed transition probabilities from i to j at each timestep divided by the sum over the probabilities of being in state s_i at each timestep (cf. Equation 2.23). Emission probability $b_{j,k}$ is updated with $\bar{b}_{j,k}$, the summed probabilites of being in state s_j and emitting v_k at each timestep n divided by the summed probabilities of being in state s_j at each timestep n (cf. Equation 2.24), where $1_{[x_n=v_k]}$ is the indicator function, i.e. it is 1 if the condition is true, otherwise it is zero. This way parameters are updated iteratively to increase the likelihood of the training sequence X towards a local optimum until an early stopping criterion is met.

2.2.2 Connectionist Temporal Classification

Graves et. al. [40] define connectionist temporal classification (CTC) as an extension of temporal classification to connectionist networks. Temporal classification is the classification of unsegmented sequences to minimize some error measure. The error measure used by Graves et. al. is the label error rate (LER), the normalized Levenshtein distance between predicted and true labels of a test set. Graves et. al. propose an extension to connectionist networks, by showing how to use a modified forward backward algorithm and backpropagation for training a neural network and how to decode the network outputs to get the most probable label sequence. The trained network is time synchronous, meaning that it provides an output at every timestep.

Classification

We want to map an input sequence of m-dimensional audio features $X = (\boldsymbol{x_1}, ..., \boldsymbol{x_T}) \quad \boldsymbol{x_i} \in \mathbb{R}^m$ with arbitrary length T to a label sequence $Z \in \mathcal{L}^{\leq T} = (\boldsymbol{z_1}, ..., \boldsymbol{z_U})$. $\mathcal{L}$ is the set of labels used to represent all valid transcriptions. Therefore, the alphabet and the space character are a suitable choice so that $\mathcal{L} = \{A, B, ..., Z, < SPACE >\}$. Assume we have a network trained with CTC defined as $DNN : (\mathbb{R}^m)^T \to (\mathbb{R}^{|\mathcal{L}|})^T$ that produces a sequence of T probability distributions $Y = (\boldsymbol{y_1}, ..., \boldsymbol{y_T}) \quad \boldsymbol{y_i} \in \mathbb{R}^{|\mathcal{L}|}$ over all $|\mathcal{L}|$ labels for a sequence of T input frames. When classifying a frame sequence X with the network it is assumed that the number of labels produced must be smaller or equal to the number of input frames, but the system produces outputs for every frame in the input sequence. Therefore we define a mapping function

$$A : \mathcal{L}^T \to \mathcal{L}^{\leq T} \tag{2.25}$$

that reduces a sequence of labels by merging adjacent repeated labels. This mapping function has the drawback of removing valid repeated letters as in the following example:

$$A(h, e, e, e, l, l, l, o) = (h, e, l, o) \tag{2.26}$$

To be able to map repeated characters Graves et al. [40] define the lables that can be assigned to a frame as $\mathcal{L}' = \mathcal{L} \cup blank$. The *blank* label is used to classify noisy frames where no other label is suitable and to seperate repeated labels from each other. We define an extended mapping function

$$B : (\mathcal{L}')^T \to \mathcal{L}^{\leq T} \tag{2.27}$$

that merges all repeated labels and then removes all *blank* elements. Equation 2.28 shows an example of the application of function B to labels of speech frames displayed in Figure 2.6:

$$B(h, blank, blank, e, e, e, l, l, blank, l, blank, o, o, o, o, o, o, blank) = (h, e, l, l, o) \tag{2.28}$$

For classification of input frames we define a network $DNN' : (\mathbb{R}^m)^T \to (\mathbb{R}^{|\mathcal{L}'|})^T$ that maps a sequence of input features X to a sequence of probability distributions Y over the extended alphabet $\mathcal{L}'$ conditioned on X. Let y_t^k be the conditional probability of symbol k at timestep t given the input. Define any symbol sequence of length T as path $\Pi = (\boldsymbol{\pi_1}, ..., \boldsymbol{\pi_T}) \quad \pi_i \in \mathcal{L}'$ and let π_t denote the label of Π at timestep t. We can describe the probability of observing any symbol sequence Π given the input sequence X by multiplying the probabilities of the labels of

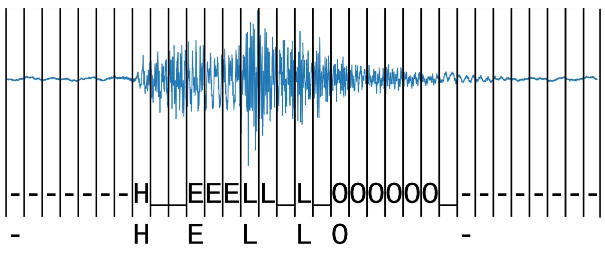

-|-|-|-|-|-|-|-|H|_|_|E|E|E|L|L|_|L|_|O|O|O|O|O|O|_|-|-|-|-|-|-|-|-
- H E L L O -

Figure 2.6: Repeated characters are collapsed to one character because a single character can be recognized in multiple adjacent frames. Then blank symbols denoted by _ are removed from the label sequence resulting in the label sequence "HELLO". Space symbols are denoted by −.

Π at each timestep t.

$$p(\Pi|X) = \prod_{t=1}^{T} y_t^{\pi_t}, \forall \Pi \in \mathcal{L}'^T \tag{2.29}$$

Figure 2.7 shows all possible paths that correspond to the same label sequence given an input

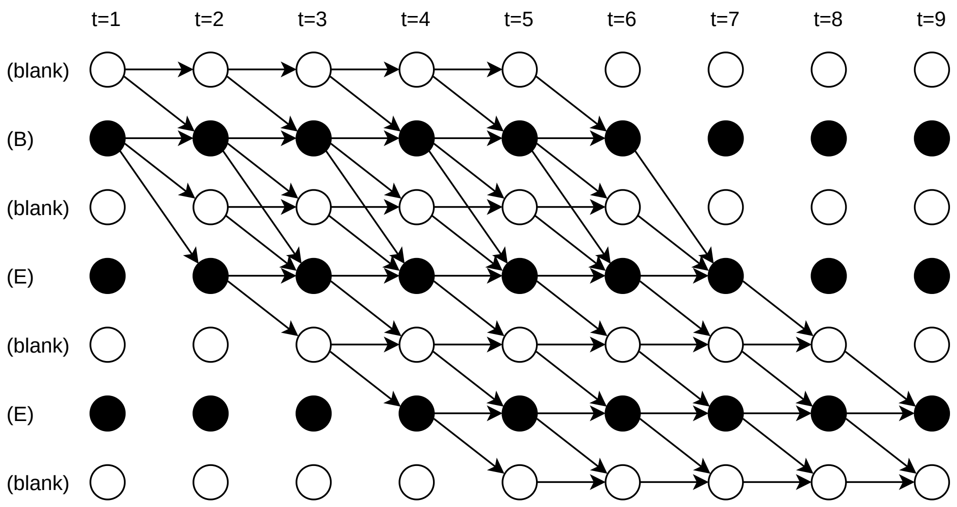


Figure 2.7: Trellis diagram visualizing the possible CTC paths corresponding to the word "bee" for an input sequence of 9 frames.

sequence X. The conditional probability of a final label sequence $L \in \mathcal{L}^{\leq T}$ given X is the sum of conditional probabilities of all paths that can be mapped to L using the mapping function B given X.

$$p(L|X) = \sum_{\Pi \in B^{-1}(L)} p(\Pi|X) \tag{2.30}$$

To get the label sequence with the highest probability we want to find the label sequence L that maximizes Equation 2.30 such that

$$h(X) = \operatorname*{argmax}_{L \in \mathcal{L}^{\leq T}} p(L|X) \ . \tag{2.31}$$

A trivial approach to approximate this complicated calculation is greedy or best path decoding. We approximate the most probable label with the most probable path, by picking the most probable symbol at each timestep. However, this does not result in the optimal solution because the probability of a label does not only depend on the most probable path, but on all paths corresponding to it (cf. Equation 2.30). Prefix search decoding is guaranteed to find the most probable label by calculating the total probability of the label starting with each symbol and then expanding the most probable prefix as visualized in Figure 2.8. However, this is not feasible for long sequences due to exponential growth in the search space. Among other decoding approaches a commonly used solution that provides a tradeoff between computational complexity and label quality is a beam search [81, 82].

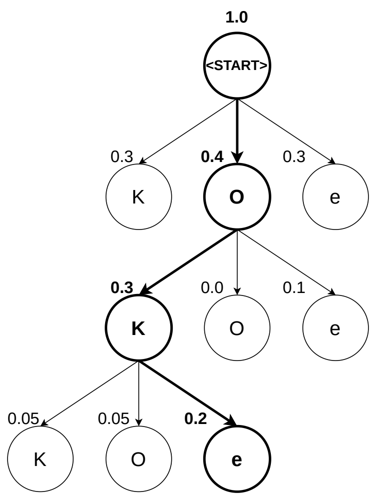


Figure 2.8: A prefix search through the alphabet $\{O, K\}$ where e represents the end of the sequence. The numbers above the states represent the total probability of all paths corresponding to the label starting with the prefix up to that state. The prefix with the highest probability is expanded with all symbols of the alphabet and the end token e. The prefix is expanded until the most probable prefix ends with an e. Then the prefix, in this case "OK", is the most probable label.

Training

For training Graves et al. derive an objective function that can be trained with gradient descent. When minimized, the objective function maximizes the log likelihood of the correct label sequences in the training set $\mathcal{S}$ or of a single correct label sequence $Z = (z_1, ..., z_U) \quad z_i \in \mathcal{L},\ U \leq T$ corresponding to a training sequence X, i.e.

$$L(\mathcal{S}) = -\sum_{(X,Z)\in\mathcal{S}} \ln(p(Z|X)) \tag{2.32}$$

$$L(X, Z) = -\ln(p(Z|X)) \ . \tag{2.33}$$

To train the neural network with backpropagation [83] and gradient descent [84] we need the partial derivatives of the loss with regard to the network outputs y_t^k given by

$$\frac{\partial L(X,Z)}{\partial y_t^k} = -\frac{\partial \ln(p(Z|X))}{\partial y_t^k} = -\frac{1}{p(Z|X)}\frac{\partial p(Z|X)}{\partial y_t^k} \tag{2.34}$$

and the partial derivatives of the loss with regard to the unnormalized network activations a_t^k of neuron k at timestep t before the softmax

$$\frac{\partial L(X,Z)}{\partial a_t^k} = -\sum_{k'} \frac{\partial L(X,Z)}{\partial y_t^{k'}}\frac{\partial y_t^{k'}}{\partial a_t^k} \ . \tag{2.35}$$

In the following we describe the efficient calculation of $p(Z|X)$ and use it to calculate the derivative given by Equation 2.35. As stated in Equation 2.30, the conditional probability $p(L|X)$

depends on all paths that can be mapped to L. The amount of possible paths grows rapidly with the size of the input sequence X. Therefore, we use a modified forward backward algorithm to efficiently calculate the probability of any label sequence L conditioned on the input X. First we define the label sequence $L' = (l'_1, ..., l'_{2|L|+1}) \quad l'_i \in \mathcal{L}'$ as L with a blank in the beginning and after every element, because blanks could be observed at any point in the label sequence and are removed with the mapping function B. For each element in L' at each timestep we calculate the forward and backward parameter.

Let $U_{1:v}$ of any sequence U denote the first v elements of the sequence. The forward parameter $\alpha_t(s)$ is the total probability of $L'_{1:s}$ at frame t given the input. This corresponds to the probability of all valid paths up to frame t that end in symbol L'_s, i.e.

$$\alpha_t(s) = \sum_{\Pi \in \mathcal{L}'^T : B(\Pi_{1:t}) = L_{1:s}} \prod_{t'=1}^{t} y_{t'}^{\pi_{t'}} \ . \tag{2.36}$$

To calculate the forward parameter recursively we set the initial values to

$$\alpha_1(1) = y_1^{l'_1} \tag{2.37}$$
$$\alpha_1(2) = y_1^{l'_2} \tag{2.38}$$
$$\alpha_1(s) = 0, \forall s > 2 \, . \tag{2.39}$$

In Equations 2.37 and 2.38 the forward parameters for the first two states are set to their corresponding symbol probabilities according to the network output at frame 1, i.e. $P(y_1|X)$. The other states described in Equation 2.39 are unreachable at timestep 1 because all paths that can be mapped to L need to either start at the first letter l'_2 or at the initial blank symbol l'_1. The recursion is defined as

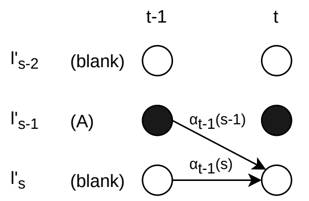
$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y_t^{l'_s} & \text{if } l'_s = \text{blank or } l'_{s-2} = l'_s \quad (2.40) \\ (\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)) y_t^{l'_s} & \text{otherwise} \quad (2.41) \end{cases}$$

$$\bar{\alpha}_t(s) = \alpha_{t-1}(s) + \alpha_{t-1}(s-1) \tag{2.42}$$

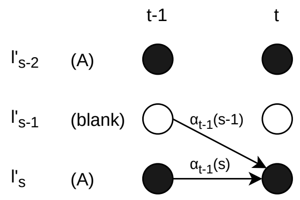
The term $\bar{\alpha}_t(s)$ defined in Equation 2.42 is the sum of probabilities of the paths at timestep $t-1$ being equal to $L'_{1:s}$ or $L'_{1:(s-1)}$. If l'_s is a blank or a repeated character the forward parameter $\alpha_t(s)$ is the sum of probabilities of prefixes $L'_{1:s}$ and $L'_{1:(s-1)}$ conditioned on the input times the network activation $y_t^{l'_s}$ of observing l'_s at timestep t as described in Equation 2.40. Figures 2.9a and 2.9b illustrate why $\alpha_{t-1}(s-2)$ is not part of the calculation of $\alpha_t(s)$ in these cases. Otherwise the symbol l'_s is a non blank and non repeated character as illustrated in Figure 2.9c. Therefore, $L'_{1:(s-2)}$ is also a valid prefix, because the blank symbol l'_{s-1} can be skipped and the sequence could still be mapped to L. In this case the total probability of valid prefixes of $\alpha_t(s)$ given the input is the sum of $\alpha_{t-1}(s)$, $\alpha_{t-1}(s-1)$ and $\alpha_{t-1}(s-2)$. This probability multiplied with the conditional probability of observing l'_s at timestep t according to the network activation $y_t^{l'_s}$ is the forward parameter described in Equation 2.41.

The backward parameter $\beta_t(s)$ is the conditional probability of all valid paths starting at frame $t+1$ that are preceded by any path in $\alpha_t(s)$. This is formalized as Equation 2.43 [85]:

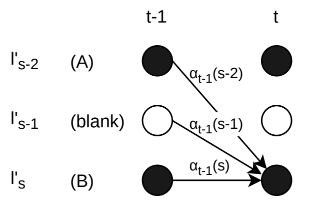
$$\beta_t(s) = \sum_{\Pi \in \mathcal{L}'^T : B(\Pi_{t:T}) = L_{s:|L|}} \prod_{t'=t+1}^{T} y_{t'}^{\pi_{t'}} \ . \tag{2.43}$$



(a) Valid transitions to a blank symbol l'_s either come from that blank symbol or from the previous symbol l'_{s-1}. A transition from l'_{s-2} would skip the symbol A and is therefore invalid.



(b) Valid transitions to a repeated symbol l'_s (where $l'_s = l'_{s-2}$) either come from that l'_s or from the previous blank symbol l'_{s-1}. A transition from l'_{s-2} would skip the blank symbol that seperates repeated characters and is therefore invalid.



(c) Valid transitions to a non blank non repeated symbol l'_s can also come from l'_{s-2} skipping the blank symbol l'_{s-1}.

Figure 2.9: Calculation of the forward parameter depending on the current symbol l'_s.

To calculate the backward parameter recursively we set the initial values to

$$\beta_T(|L'|) = 1 \tag{2.44}$$
$$\beta_T(|L'|-1) = 1 \tag{2.45}$$
$$\beta_T(s) = 0, \forall s < |L'|-1 \ . \tag{2.46}$$

In Equations 2.44 and 2.45 we initialize the backward parameters of the last two symbols (blank and $l_{|l|}$) with 1, because they are valid end states for L. Equation 2.46 sets the other symbols to 0 because they cannot be the last symbol of any path that is mappable to L. The recursion is defined as

$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s) + \bar{\beta}_t(s+1) & \text{if } l'_s = \text{blank or } l'_{s+2} = l'_s & (2.47) \\ \bar{\beta}_t(s) + \bar{\beta}_t(s+1) + \bar{\beta}_t(s+2) & \text{otherwise} & (2.48) \end{cases}$$

$$\bar{\beta}_t(s) = \beta_{t+1}(s) y_{t+1}^{l'_s} \tag{2.49}$$

The term $\bar{\beta}_t(s)$ is the sum of probabilities of valid paths at timestep $t+1$ equal to $L'_{s:|L'|}$ scaled by the network activation of being in state l'_s at timestep $t+1$. If l'_s is a blank or a repeated character at timestep t the only legal transitions are to l'_s and l'_{s+1} and therefore its probability is calculated with Equation 2.47. Otherwise the transition to l'_{s+2} is also possible skipping a blank symbol in the process. The resulting probability is calculated in Equation 2.48. The forward and backward parameters get increasingly smaller as the algorithm progresses increasing the chance of underflows. Graves et. al. [85] therefore recommend working in the log domain in practice. The total probability of a label given an input sequence can now be calculated as a sum of the probabilities of paths ending in the final blank or final character at timestep T. All valid paths that can be mapped to L need to end in the final blank $l'_{|L'|}$ or the final character $l'_{|L'|-1}$:

$$p(L|X) = \alpha_T(|L'|) + \alpha_T(|L'|-1) \ . \tag{2.50}$$

When multiplying the forward and backward parameter of the same element s of the label sequence at timestep t we get the total probability of all paths going through l'_s at t.

$$\alpha_t(s)\beta_t(s) = \sum_{\Pi \in B^{-1}(L):\pi_t = l_s} p(\Pi|X) \ . \tag{2.51}$$

An example of these paths is shown in Figure 2.10. As all paths go through some symbol at any

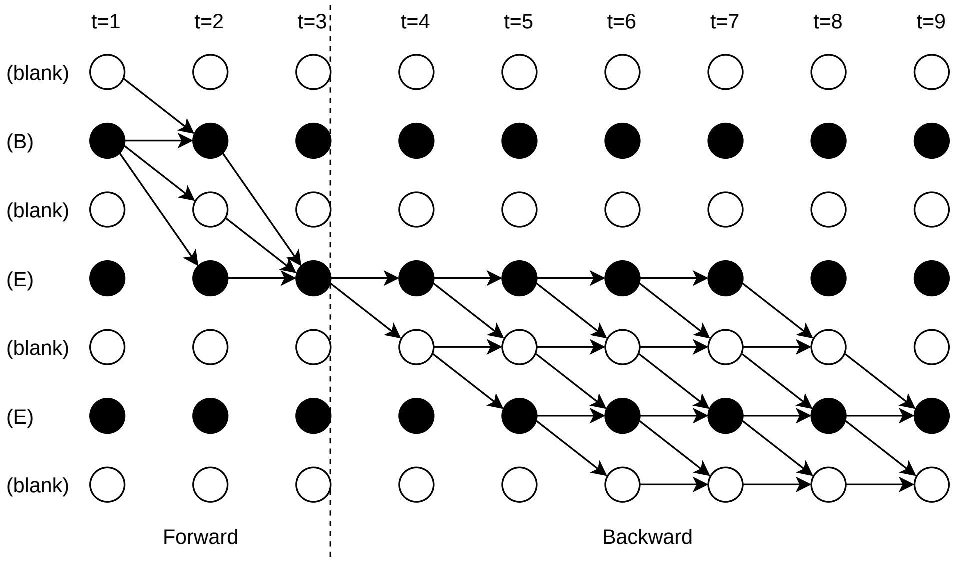


Figure 2.10: An example of all paths going through l'_4 at timestep $t = 3$.

timestep of the input, summing the probability of all paths going through all symbols of L' at timestep t results in the total probability of L given the input X. So for any timestep $t \in 1, ..., T$

$$p(L|X) = \sum_{s=1}^{|L'|} \alpha_t(s)\beta_t(s) \ . \tag{2.52}$$

To derive Equation 2.52 with respect to y_t^k we look at all paths going through the label k at timestep t. The label sequence L' might contain zero or more instances of label k that are reachable at time t, therefore Graves et. al. define a function $lab(L,k) = \{s : l'_s = k\}$ that returns a set of all positions of k in the label sequence L'. Deriving 2.52 yields

$$\frac{\partial p(L|X)}{\partial y_t^k} = \frac{1}{y_t^k} \sum_{s \in lab(L,k)} \alpha_t(s)\beta_t(s) \ . \tag{2.53}$$

When we set $L = Z$ this can be substituted into the partial derivative of the loss function in Equation 2.34 resulting in

$$\frac{\partial L(X,Z)}{\partial y_t^k} = -\frac{1}{p(Z|X)y_t^k} \sum_{s \in lab(Z,k)} \alpha_t(s)\beta_t(s) \ . \tag{2.54}$$

Substituting the partial derivative of the softmax and Equation 2.54 into Equation 2.35 yields the error that can be used for training the weight matrices of the network with backpropagation:

$$\frac{\partial L(X,Z)}{\partial a_t^k} = y_t^k - \frac{1}{p(Z|X)} \sum_{s \in lab(Z,k)} \alpha_t(s)\beta_t(s) \ . \tag{2.55}$$

2.2.3 Attention Models

Attention is a mechanism used to exploit context of the whole input at each decoding step and can be used in many different model architectures [54, 86]. Initially, attention in ASR was used with an encoder decoder sequence to sequence model [49]. Therefore, we will shortly explain encoder decoder sequence to sequence models and then extend them with an attention mechanism. Encoder decoder models can be used to model sequence to sequence learning tasks where the input and output sequence do not necessarily have the same length [87]. The encoder reads the whole input sequence and compresses it into a single vector of fixed size also known as context vector. Using an LSTM encoder the last hidden state is used as the context vector. To mark the end of a sequence the last symbol of each trained sequence must be an end of sequence token (<EOS>). The decoder emits a softmax over the whole alphabet at each emission step based on its hidden state, its previous output and the context vector. The decoder's hidden state is conditioned on the context vector, the previous hidden state and its previous emission. When the decoder emits an <EOS> symbol it has finished decoding the sequence. Figure 2.11 shows

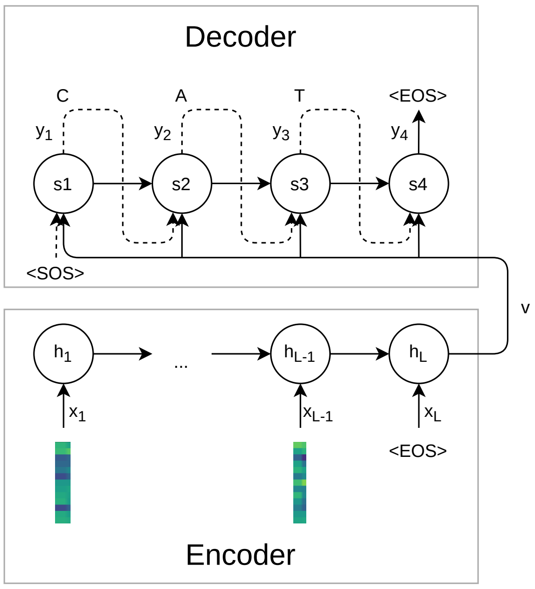


Figure 2.11: Encoder decoder architecture encoding L audio features into a fixed size context vector $\boldsymbol{v}$*. The decoder then transcribes the input by emitting symbols from a defined alphabet.*

how L audio feature vectors are encoded to a fixed size context vector $\boldsymbol{v}$. At each decoding step $\boldsymbol{v}$, the previous state s_{i-1} and the previous emission y_{i-1} are used to calculate the next hidden state s_i and the corresponding emission y_i. This approach suffers when the input sequence is large [88], because it cannot compress variably sized sequences into a fixed size vector without losing information. To mitigate this problem the encoder decoder model is extended by an attention mechanism [49, 89]. Chorowski et al. [49] argue that Bahdanau attention [89] is not suitable for ASR due to high noise in the input and very long input sequences. Therefore, they propose an extension of Bahdanau attention for speech recognition:

$$\alpha_i = Attend(s_{i-1}, \alpha_{i-1}, H) \tag{2.56}$$

$$g_i = \sum_{j=1}^{L} \alpha_{i,j} h_j \tag{2.57}$$

$$y_i = Generate(s_{i-1}, g_i) \tag{2.58}$$

$$s_i = f(s_{i-1}, g_i, y_i) \ . \tag{2.59}$$

Equation 2.56 shows the attention mechanism calculating an alignment α_i at decoding step i from the previous decoder state s_{i-1}, the previous alignment α_{i-1} and the sequence of outputs of the encoder $H = (h_1, ...h_L)$. We calculate g_i, a glimpse of the encoder output at decoding step i, as the sum over all elements in the encoder output sequence H weigthed by the alignment vector α_i (cf. Equation 2.57). Intuitively a glimpse is a part of the encoder output that the decoder attends to at the current decoding step. Then the decoder produces an output y_i based on the previous decoder state s_{i-1} and the glimpse of the encoder output g_i. The next decoder state is then calculated from the previous decoder state s_{i-1}, the current glimpse g_i and the current output y_i (cf. Equation 2.59). Conditioning on the output sequence H of the encoder state is known as content based attention and conditioning on the previous alignment α_{i-1} is known as location based attention. Chorowski et al. [49] combine the two types of attention to minimize their respective weaknesses. The hybrid attention mechanism in Equation 2.56 uses the alignment a_{i-1} to preselect elements in H, scores them with content based attention, i.e.

$$e_{i,j} = Score(s_{i-1}, h_j) \tag{2.60}$$

and normalizes the score with a softmax.

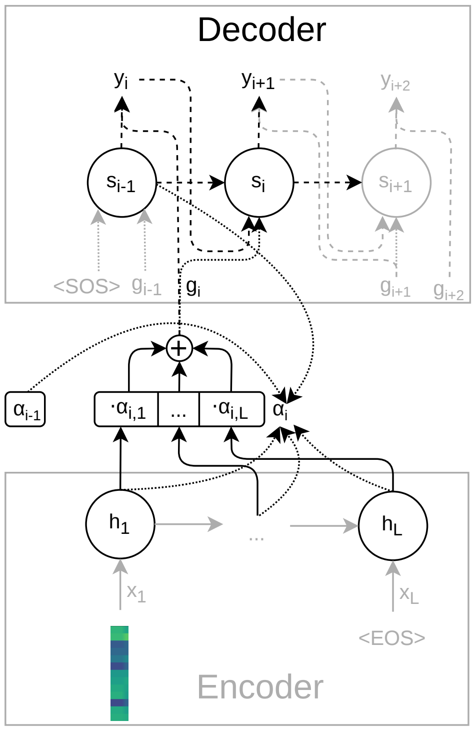


Figure 2.12: Visualization of encoder decoder architecture with attention.

Figure 2.12 shows the different steps of the algorithm proposed by Chorowski et al. [49]. The attend step that calculates the attention vector in Equation 2.56 is displayed by the dotted lines. The calculation of the glimpse from the attention vector and all outputs of the encoder described in Equation 2.57 is plotted as solid lines. The dashed lines represent the generation of an output y_i (cf. Equation 2.58) and the calculation of the next state s_i (cf. Equation 2.59). Figure 2.13 shows how the attend function produces attention vectors for every encoder output, effectively aligning the input data for transcription.

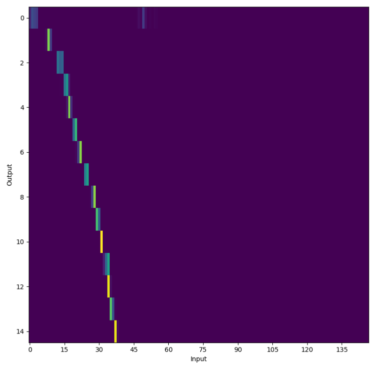


Figure 2.13: Visualization of attention vectors plotted as lines below each other producing the alignment of inputs on the x axis with outputs on the y axis.

3 An overview of state-of-the-art Speech Recognition Frameworks

3.1 An overview of state-of-the-art Speech Recognition Frameworks

This section introduces modern open source ASR frameworks and describes acoustic model architectures that deliver state-of-the-art performance. The frameworks were selected based on their recency, reported accuracy and availability as open source projects. The described model architectures are used in the experiments of Chapter 4, where they are evaluated without the use of a language model.

3.1.1 Kaldi HMM/TDNN-F architecture

In this section we describe the framework Kaldi which provides an implementation of the acoustic model architecture HMM/factorized time delay neural network (TDNN-F) [90]. Additionally, the HMM/TDNN-F acoustic model architecture used for evaluation in Chapter 4 is described. Kaldi is a feature rich modular toolkit for building ASR systems, especially acoustic models based on HMMs, GMMs, DNNs and weighted finite state transducers [16]. It is open source software written in C++ and Bash provided under the Apache 2.0 license. Kaldi runs on Linux and Windows and its source code is available on Github[1].

For preprocessing it has support for many feature extraction methods including Mel-scaled filter-banks (cf. Section 2.1.1), MFCCs (cf. Section 2.1.1) and PLPs [91] and includes data augmentation such as time warping and frame shifting. Forced alignment can be done with either HMM-GMM or HMM-DNN hybrids. Weighted finite state transducers are used as a common framework to represent and combine commonly used probabilistic models in speech recognition like HMMs, word and phone models, LMs and phonetic dictionaries [92]. Mohri et. al. [92] provide algorithms for efficient combination and optimization of models that have been translated to the domain of finite state transducers. For parallelization Kaldi has multi-GPU and multi-CPU support and runs on cluster workload managers like Sun GridEngine or Slurm. In particular, an ASR pipeline (cf. Section 1.1.6) is configured with a Bash script calling the necessary tools for preprocessing, feature extraction, LM training, acoustic model training and evaluation. These Bash scripts chaining tools together to create a pipeline are called recipes. Kaldi comes with many recipes defining ASR pipelines for popular datasets.

In the following we shortly describe a TDNN [30] and its factorized version TDNN-F. Then we summarize the TDNN-F architecture implemented in Kaldi used for the experiments in Chapter 4. A TDNN is a neural network that leverages temporal context by performing convolutions over fixed size time windows [30]. Due to their feed forward nature TDNNs do not suffer of high training times of RNNs [1]. By stacking multiple TDNN layers on top of each other a network can use short term context in lower layers and long term context in upper layers as

[1] https://github.com/kaldi-asr/kaldi

seen in Figure 3.1. A TDNN-F is a factorized version of a TDNN that reduces the amount of

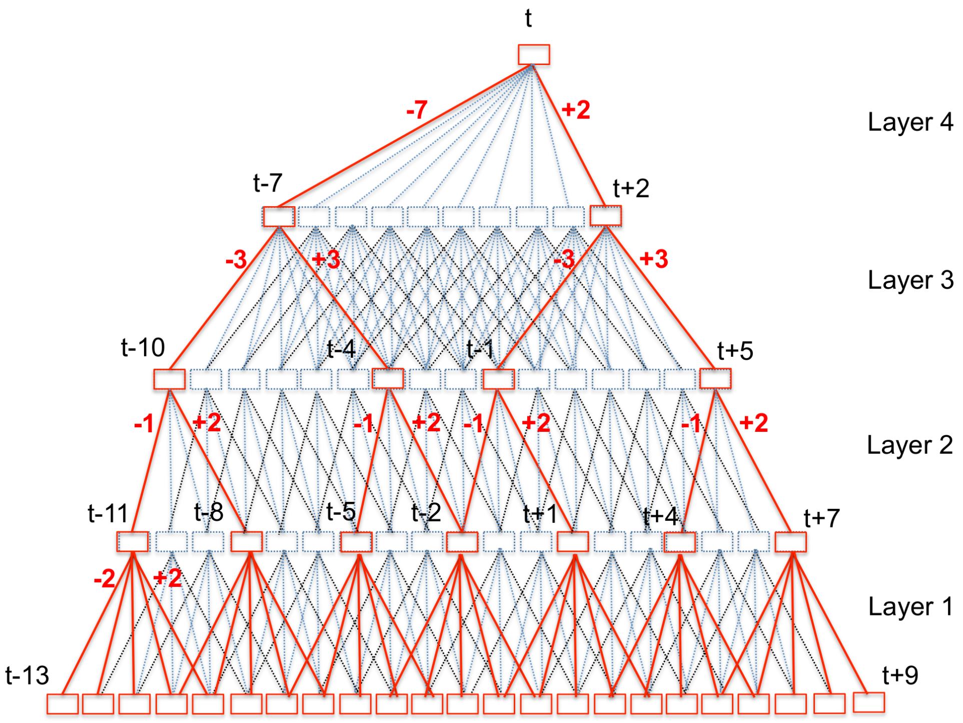


Figure 3.1: Visualization of a multi layer TDNN by Peddinti et al. [1].

parameters that need to be trained compared to a TDNN [90]. A TDNN-F makes use of singular value decomposition (SVD) to decompose the weight matrix W of a TDNN into the product of two smaller matrices AB. A semiorthogonal constraint is imposed on B and its k lowest values are set to 0 decreasing the amount of parameters with a minimal effect on accuracy. A and B are then used as the weight matrices of two seperate TDNN-F sub-layers. Therefore, a single TDNN-F has an input dimension, a dimension after the transformation with the first weight matrix and an output dimension after the transformation with the second weight matrix. Additionally, there are skip connections connecting TDNN sub-layers for a more direct dataflow. The HMM/TDNN-F architecture evaluated in Chapter 4 is illustrated in Figure 3.2. The input features consist of 40-dimensional MFCCs [18] with a window of 3 frames concatenated with a 100 dimensional i-Vector [93] of the current frame. Local context information is gathered by stacking multiple TDNN-F layers with a small context window of [-1,+1]. These are followed by TDNN-F layers with a larger context window of [-3,+3]. This way previous and future context can be leveraged without the overhead of a Bidirectional RNN. Then there is a linear layer followed by a NaturalGradientAffine layer which is a feed forward layer with an inverse Fisher matrix as learning rate matrix [94]. The final part of the network consists of a linear layer followed by another NaturalGradientAffine layer with a softmax activation function. The outputs are observation probabilities for the phonetic states of the HMM (cf. Chapter 2). All layers use a ReLU activation function and employ batch normalization. The objective function optimized in training is the log likelihood of the correct phone sequence. Decoding is performed with a beam search algorithm.

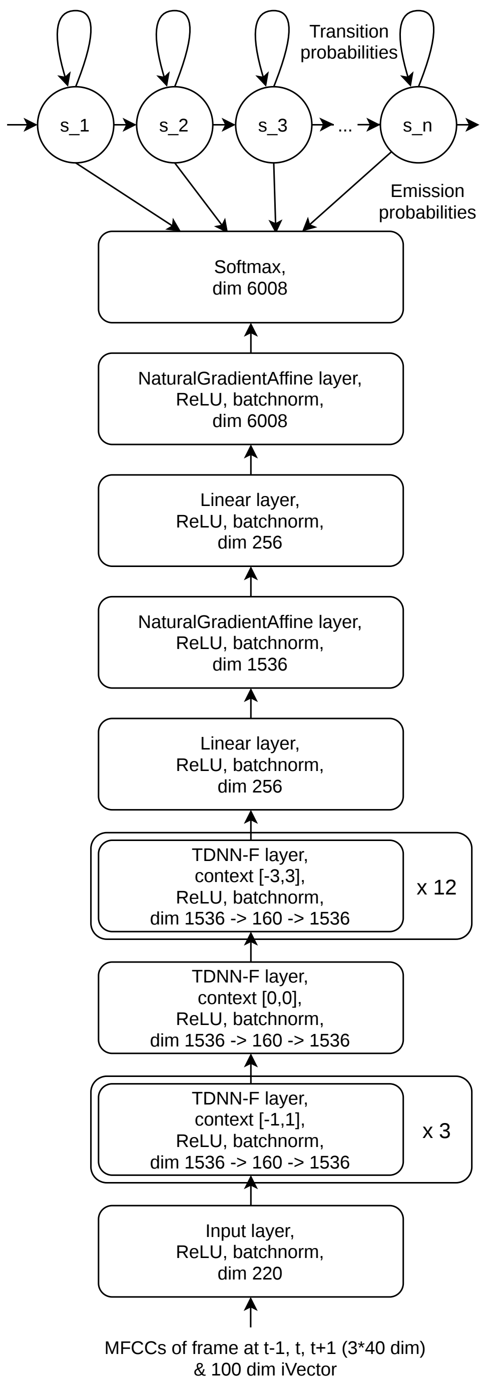

Transition probabilities
s_1
s_2
s_3
...
s_n
Emission probabilities
Softmax, dim 6008
NaturalGradientAffine layer, ReLU, batchnorm, dim 6008
Linear layer, ReLU, batchnorm, dim 256
NaturalGradientAffine layer, ReLU, batchnorm, dim 1536
Linear layer, ReLU, batchnorm, dim 256
TDNN-F layer, context [-3,3], ReLU, batchnorm, dim 1536 -> 160 -> 1536
x 12
TDNN-F layer, context [0,0], ReLU, batchnorm, dim 1536 -> 160 -> 1536
TDNN-F layer, context [-1,1], ReLU, batchnorm, dim 1536 -> 160 -> 1536
x 3
Input layer, ReLU, batchnorm, dim 220
MFCCs of frame at t-1, t, t+1 (3*40 dim) & 100 dim iVector

Figure 3.2: Architecture of HMM/TDNN acoustic model used in the experiments.

3.1.2 ESPnet

ESPnet [17] provides an implementation of an acoustic model architecture based on a transformer [54] among other neural architectures. ESPnet is an ASR and text-to-speech (TTS) toolkit focused on end-to-end deeplearning models based on the neural network frameworks PyTorch and Chainer. It is open source software written in Python and Bash and provided under the Apache 2.0 license. ESPnet runs on Linux and its source code is available on Github[2]. The ESPnet transformer acoustic model architecture is used for evaluation in Chapter 4.

For preprocessing and feature extraction it uses the provided tools from the Kaldi toolkit. Acoustic modeling is done by DNNs with CTC, attention or hybrid CTC and attention. The supported DNN architectures are CNN with Bidirectional RNN, subsampling bidirectional RNNs and transformers. For parallelization ESPnet has multi-GPU and multi-CPU support and runs on cluster workload managers like Sun GridEngine or Slurm. Comparable to Kaldi, ESPnet comes with a large set of Bash recipes for popular datasets that chain tools available in ESPnet to define ASR architectures.

In the following we shortly describe ESPnet's transformer ASR architecture illustrated in Figure 3.3 used for the experiments in Chapter 4. As input features 80 dimensional Mel-scaled filter-bank features are concatenated with pitch information, which are passed into convolutional layers. The resulting vector is fed to a transformer with 12 encoding layers, 6 decoding layers, 4 attention heads and 2048 feed forward units each. The previous output is also embedded and passed to the transformer. A self attention layer returns a sum of the inputs weighted by an attention vector. Multiheaded self attention combines multiple result vectors by calculating multiple attention vectors and reducing the dimensionality by multiplying the result with a trained matrix. The add and norm layers sum up the input and output of the previous layer and apply a normalization technique called layer normalization [95]. Compared to RNNs, the transformer has no information at which position the current input is. Therefore, all embeddings are enhanced with positional information by adding a positional encoding [54]. The output activations of the transformer encoder are fed to a 5002 dimensional feed forward layer with CTC loss and a softmax activation function. The output activations of the transformer decoder are fed to a 5002 dimensional linear layer with softmax activation function. Compared to other models the output vocabulary is more sophisticated than the alphabet: Using SentencePiece [96] we generate a vocabulary of 5002 subword units from the training data whose probability is predicted jointly by the CTC output layer and transformer output layer. During training hybrid CTC/attention is done by optimizing a weighted sum of CTC loss (cf. Section 2.2.2) and attention cross entropy. Decoding is done with a beam search combining weighted CTC and attention scores without a LM.

[2] https://github.com/espnet/espnet

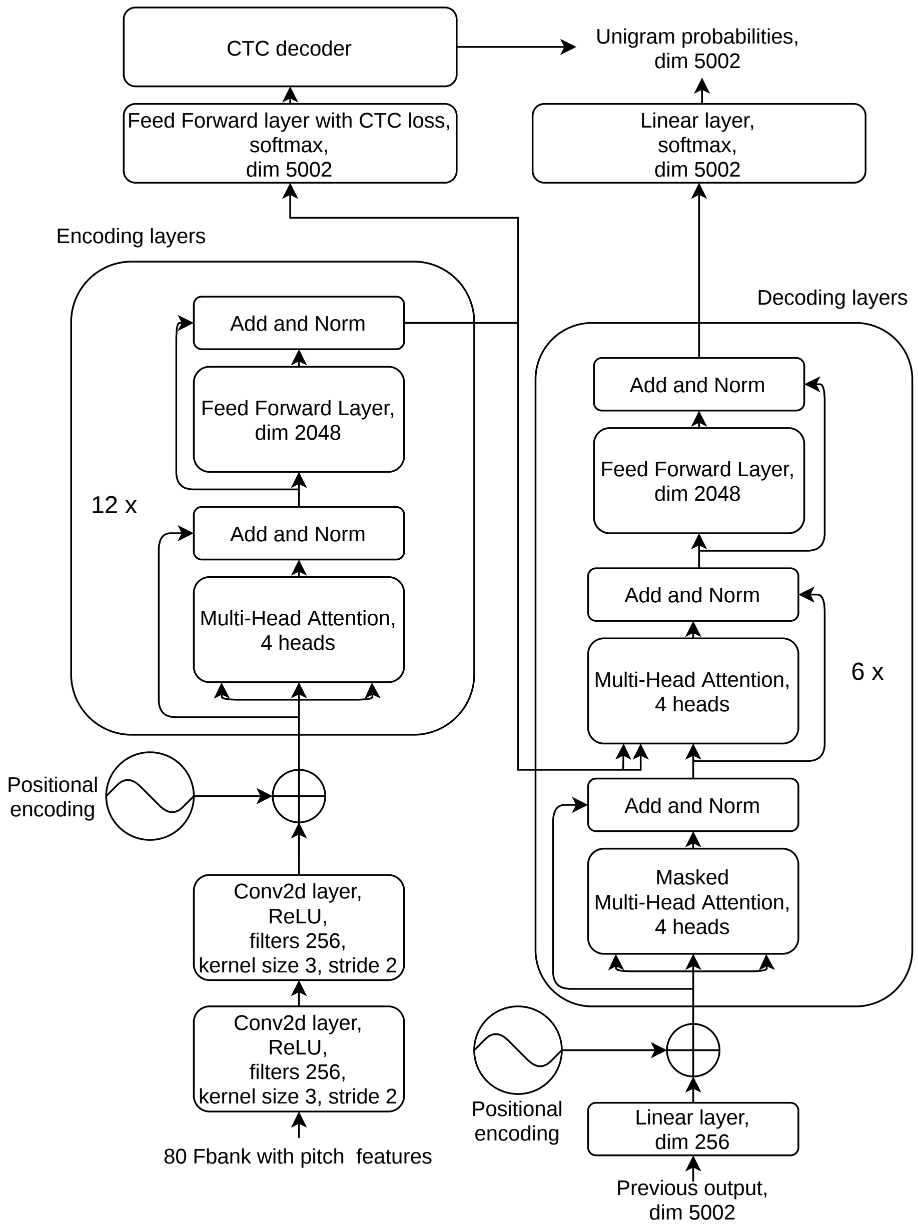

CTC decoder
Unigram probabilities, dim 5002
Feed Forward layer with CTC loss, softmax, dim 5002
Linear layer, softmax, dim 5002
Encoding layers
Decoding layers
Add and Norm
Feed Forward Layer, dim 2048
12 x
Add and Norm
Multi-Head Attention, 4 heads
Add and Norm
Feed Forward Layer, dim 2048
Add and Norm
Multi-Head Attention, 4 heads
6 x
Add and Norm
Masked Multi-Head Attention, 4 heads
Positional encoding
Conv2d layer, ReLU, filters 256, kernel size 3, stride 2
Conv2d layer, ReLU, filters 256, kernel size 3, stride 2
80 Fbank with pitch features
Positional encoding
Linear layer, dim 256
Previous output, dim 5002

Figure 3.3: Architecture of transformer acoustic model used in the experiments.

3.1.3 DeepSpeech 2

In this section we describe the frameworks Mozilla Deep Speech and OpenSeq2Seq which provide an implementation of the acoustic model architecture DeepSpeech 2 [97]. The DeepSpeech 2 framework is used for evaluation in Chapter 4. Mozilla Deepspeech is an ASR framework for end-to-end deep learning models based on Baidu's Deep Speech 2 paper [97], building on the neural network framework TensorFlow. It is open source software written in C++ and C with Python and NodeJS bindings and is published under the Mozilla Public License 2.0. DeepSpeech runs on Linux, Windows and macOS and is available on Github[3].

For preprocessing it supports MFCC [18] feature extraction and various audio and spectogram augmentation algorithms. As training criterion it supports CTC and it is compatible with n-gram LMs trained with KenLM [75]. It has implemented automatic mixed precision training which leads to training speedup. Additionally, there is the option to export considerably smaller TFLite models that can be used for mobile inference. Deep Speech has multi-GPU support that uses hybrid parallel optimization where gradients for minibatches are averaged.

Another framework that provides an implementaiton of the DeepSpeech 2 architecture is Nvidia's OpenSeq2Seq [76]. OpenSeq2Seq is an ASR framework for training Seq2Seq models using TensorFlow as neural network framework. It is open source software written in Python and C++ provided under the Apache 2.0 license. OpenSeq2Seq runs on Linux and its source code is available on Github[4]. OpenSeq2Seq comes with distributed multi GPU and multi node training with Horovod [98], a distributed training framework for deep learning, as well as mixed precision training with Tensor Cores for improved training speed. For preprocessing it implements several feature extraction methods including spectograms, Mel scaled spectograms or MFCCs [18] as well as data augmentation with speed perturbation and time/frequency masks. It provides recipes and pretrained models for multiple domains including ASR, neural machine translation, speech synthesis, LMs, sentiment analysis and image classification. Supported encoder architectures are (i) DeepSpeech 2 [97], (ii) 1D CNNs like Wav2Letter [63] or Jasper [99], and (iii) 1D time-channel-seperable CNNs [100].

The architecture used for the experiments in this thesis is based on Baidu's Deep Speech 2 paper [97]. Its implementation in the OpenSeq2Seq framework is illustrated in Figure 3.4. We use 160 dimensional spectogram features derived from time stretch augmented audio as input for 3 convolution layers over the time and frequency domain. They are followed by 3 unidirectional GRU layers with 50% dropout and a lookahead convolution layer with batch normalization. The idea of the lookahead convolution layer is to take limited future context into account without the overhead of a bidirectional RNN. Next there is a feed forward layer with 2048 units followed by a 29 dimensional feed forward layer with softmax assigning probabilities to one of the 29 letters of our alphabet including *blank*, *silence* and *space*. The network is trained with CTC loss (cf. Section 2.2.2) and decoding is performed with a greedy decoder.

[3] https://github.com/mozilla/DeepSpeech
[4] https://github.com/NVIDIA/OpenSeq2Seq

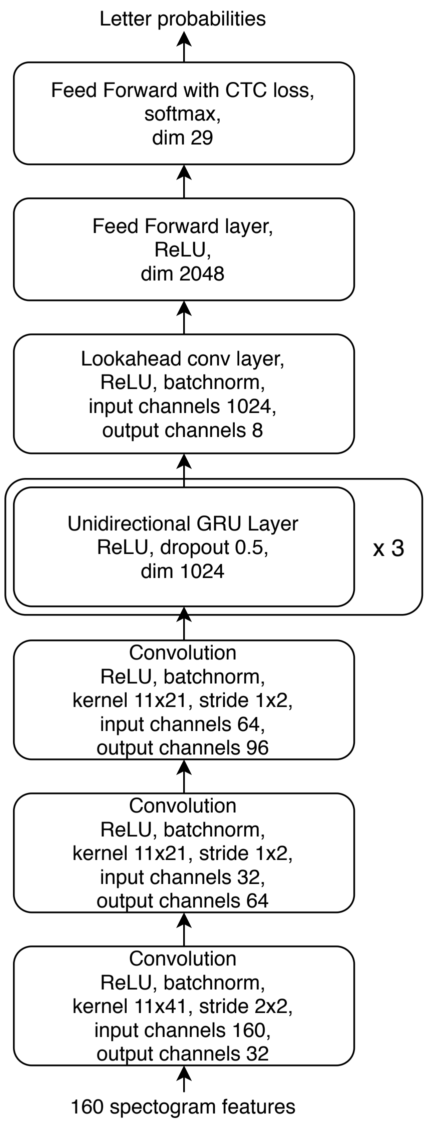

Letter probabilities
Feed Forward with CTC loss, softmax, dim 29
Feed Forward layer, ReLU, dim 2048
Lookahead conv layer, ReLU, batchnorm, input channels 1024, output channels 8
Unidirectional GRU Layer ReLU, dropout 0.5, dim 1024
x 3
Convolution ReLU, batchnorm, kernel 11x21, stride 1x2, input channels 64, output channels 96
Convolution ReLU, batchnorm, kernel 11x21, stride 1x2, input channels 32, output channels 64
Convolution ReLU, batchnorm, kernel 11x41, stride 2x2, input channels 160, output channels 32
160 spectogram features

Figure 3.4: Architecture of DeepSpeech 2 acoustic model used in the experiments.

3.1.4 Wav2Letter

The framework Wav2Letter++ provides an implementation of the acoustic model architecture Wav2Letter [63]. Wav2Letter++ is an ASR framework for end-to-end models by Facebook [77] using Flashlight as underlying machine learning library and ArrayFire as tensor library. It is open source software written in C++ provided under the BSD 3-clause license. Wav2Letter++ runs on Linux and is available on Github[5].

For preprocessing it supports several feature extraction methods including MFCCs [18], log Mel-scaled filter-banks and power spectrum features as well as data augmentation with SpecAugment [101]. Acoustic modeling in Wav2Letter++ can be done with 1D CNNs [63], time-depth separable convolutions [102] and transformers [62]. The available training criterions are CTC, Seq2Seq and AutoSegmentation Criterion [63]. It provides support for n-gram LMs trained with KenLM [75]. Wav2Letter++ comes with English pretrained models and recipes for popular ASR datasets, i.e. WSJ, Librispeech and Timit.

The architecture used for the experiments in this thesis is based on Wav2Letter [63]. Its implementation in the OpenSeq2Seq framework is illustrated in Figure 3.5. As input it takes 64 log Mel-scaled filter-bank features per frame, feeds them to several 1D convolution layers. The kernels of the 1D convolution layers increase in size, therefore the temporal context increases similiar to TDNNs. At the top is a feed forward layer with softmax activation function for classification to the alphabet. All convolutional layers employ batch normalization and dropout of 20-40% and use a ReLU activation function clipped at 20. The training criterion is CTC loss (cf. Section 2.2.2) and decoding is performed greedily without a LM.

[5] https://github.com/facebookresearch/wav2letter

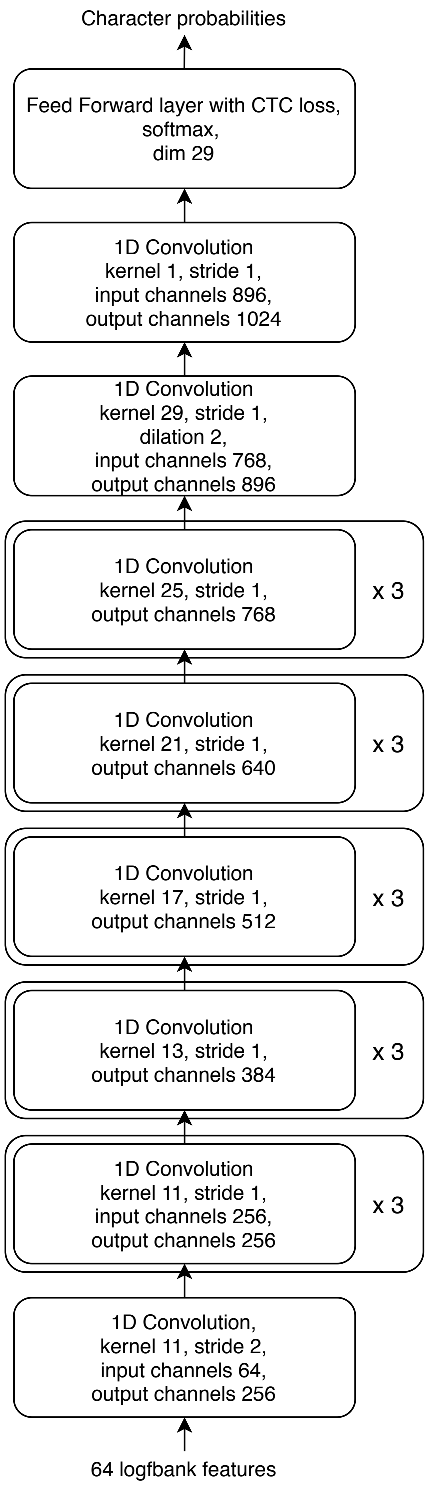

Character probabilities
Feed Forward layer with CTC loss, softmax, dim 29
1D Convolution kernel 1, stride 1, input channels 896, output channels 1024
1D Convolution kernel 29, stride 1, dilation 2, input channels 768, output channels 896
1D Convolution kernel 25, stride 1, output channels 768
x 3
1D Convolution kernel 21, stride 1, output channels 640
x 3
1D Convolution kernel 17, stride 1, output channels 512
x 3
1D Convolution kernel 13, stride 1, output channels 384
x 3
1D Convolution kernel 11, stride 1, input channels 256, output channels 256
x 3
1D Convolution, kernel 11, stride 2, input channels 64, output channels 256
64 logfbank features

Figure 3.5: Architecture of Wav2Letter model used in the experiments.

4 Experimental Evaluation

4.1 Database

In this Section we describe the databases used for the experiments in Section 4.2. We selected Librispeech [2] as dataset due to its large amount of clean speech data to evaluate the architectures described in Chapter 3. For training a German speech recognizer we selected the German Mozilla Common Voice Corpus (June 2019 release) due to its considerable size compared to other freely available datasets in German [103,104]. Both corpora are introduced in the following sections.

4.1.1 Librispeech Corpus

The Librispeech corpus [2] is based on English read speech from public domain audiobooks in the LibriVox project. It is freely available[6] under the permissive CC BY 4.0 license. In total the dataset contains approximately 980 hours of audio. Each entry in the dataset is a sentence in flac format sampled at 16kHz with its transcription, speaker id and speaker gender. Panayotov et al. [2] used an ASR system to align the speech audio on sentence level. Each sentence is assigned to a speaker and for every speaker the gender is known. This way Panayotov et al. were able to enforce gender balance on a speaker level over the whole dataset. Chapters read by multiple speakers and chapters that contained too much noise were filtered out. With an acoustic model trained on a part of the WSJ corpus all sentences were transcribed and the WER scores were measured. Roughly 50% with the lower WER were used as 'clean' datasets, the rest was classified as 'other'. From the 'clean' data they randomly sampled gender balanced testing and validation sets. The rest of the clean data was randomly split into training sets with 100 and 360 hours. In order to generate more challenging subsets testing and validation sets were sampled from the third quartile of the data ranked increasingly by WER. The rest of the other data was used for a training set with 500 hours. For every set they enforced a limit on minutes of speech per speaker to avoid bias towards certain speakers. For exact numbers refer to Table 4.1 [2].

subset	**hours**	**per-speaker minutes**	**female speakers**	**male speakers**	**total speakers**
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1166

Table 4.1: Librispeech subsets from Panayotov et al. [2].

[6] https://www.openslr.org/12/

In addition to the speech data they provide text of 14500 books from Project Gutenberg [105] with 803 million tokens and 900000 unique words for LM creation. These texts are filtered so that they do not contain any transcriptions from the validation or test sets.

4.1.2 German Mozilla Common Voice Corpus

The Mozilla Common Voice corpus [106] is based on read speech by volunteers of the Common Voice Project[7]. It is freely available under a permissive CC0 license. As of February 2020, the dataset contains over 4250 recorded hours and over 3400 validated hours of speech audio in 40 different languages[8]. Every entry in the dataset is a sentence in MP3 format sampled at 48000Hz with its transcription, speaker id and optional information about speaker age, gender and dialect. For our experiments we used FFmpeg[9] to sample 16000Hz wav files from the MP3s. The data for the voice corpus is collected with a multi step crowd sourcing approach. First a text corpus is initialized with sentences from Wikipedia articles. It can then be extended with other sentences by community members. Sentences proposed by community members are added after being approved by two other members. Then users can record sentences of the text corpus displayed to them by web or iPhone app. The recorded sentences need to be approved by other members before being added to the train/dev/test split.

set	hours	female speakers	male speakers	unknown speakers	total speakers	utterances
train	10.2	37	331	184	552	8519
dev	7	37	348	625	1010	5634
test	7.7	22	281	1598	1901	5634
total validated	324.3	173	1555	3124	4852	281208

Table 4.2: Statistics of the German Common Voice corpus released in June 2019.

As can be seen in Table 4.2 the train/dev/test split for German only contains a small amount of all validated audio. This happens because there is a lot of sentence overlap between speakers meaning that different speakers often speak the same sentences. This reduces the overall dataset size if the test and dev set should not have any sentence or speaker overlap with the training set and with each other. The splits are randomly sampled and generally have a high gender imbalance biased towards male speakers. The dataset used in the experiments in Section 4.2 is the German Common Voice June 2019 release. Since a lot of data is needed for training deep learning models, we built a larger training set from all validated utterances by relaxing the speaker overlap constraint. In the new test set we allowed speaker overlap between the train, test and dev set, but kept the constraint of no sentence overlap between the sets. This was done by taking all validated utterances and removing all utterances with sentences from the test and dev set. We will refer to the official training set with CV_A and to our large constructed dataset with CV_B. Details of the two sets are displayed in Table 4.3. CV_B contains almost 7 times the amount of utterances of the official training set. Previous reported results for this dataset were a CER of 12.8% in the ESPnet repository [10] with a language model and a custom split that allows sentence overlap, but no speaker overlap and a CER of 43.76% for an older and smaller version of the corpus [106].

[7] https://voice.mozilla.org
[8] https://voice.mozilla.org
[9] https://www.ffmpeg.org
[10] https://github.com/espnet/espnet/blob/master/egs/commonvoice/asr1/RESULTS.md

set	hours	utterances	speaker overlap	sentence overlap
CV_A	10.2	8519	no	no
CV_B	66.7	57817	yes	no

Table 4.3: The two subsets of the German Common Voice Corpus we use in the evaluation.

4.2 ASR Accuracy Evaluation

In this section several experiments performed with the ASR architectures presented in Chapter 3 (without LM) are evaluated in terms of accuracy. In particular, the commonly used character error rate (CER) and word error rate (WER) is used as a metric for accuracy. The CER is also known as Levenshtein distance [107] between the transcription hypothesis and its reference normalized by the number of characters in the reference transcription. It is defined as

$$CER = \frac{I_C + D_C + S_C}{N_{C_{Ref}}}\,, \tag{4.1}$$

where I_C is the number of characters only present in the hypothesis (insertions), D_C is the number of characters only present in the reference (deletions) and S_C is the number of substituted characters in the hypothesis when compared to the reference (substitutions). $N_{C_{Ref}}$ is the number of characters in the reference. Like CER, WER is based on Levenshtein distance, but instead of characters the number of inserted, deleted and substituted words between the sequences is counted. Therefore, the word error rate of a transcription hypothesis and its reference is defined as [108]

$$WER = \frac{I_W + D_W + S_W}{N_{W_{Ref}}}\,, \tag{4.2}$$

where I_W is the number of words only present in the hypothesis (insertions), D_W is the number of words only present in the reference (deletions) and S_W is the number of substituted words in the hypothesis when compared to the reference (substitutions). $N_{W_{Ref}}$ is the number of words in the reference. For the evaluation we used untuned and unoptimized framework code as provided by the corresponding repositories.

4.2.1 Evaluation with the Librispeech Corpus

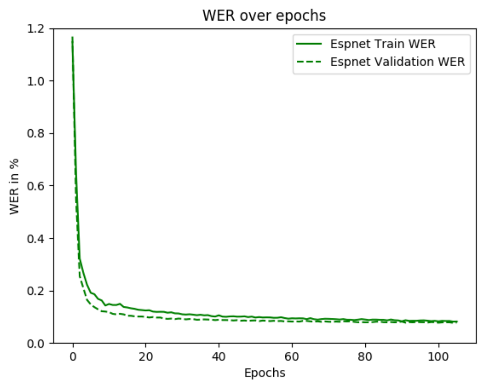
The training of the acoustic model architectures presented in Chapter 3 is evaluated using the Librispeech corpus (cf. Section 4.1.1).

Setup In the following, we shortly describe the training and decoding parameters for the ESPnet transformer architecture (cf. Section 3.1.2) trained on the Librispeech dataset. The ESPnet transformer was trained on a single Tesla K40 GPU with a batch size of 6 and gradient accumulation factor [109] of 4. The gradient accumulation factor specifies the number of batches for which gradient updates are accumulated to simulate a larger batch size on systems with little memory. In our case a gradient with accumulation factor 4 and a batch size of 6 results in a simulated batch size of 24. As optimizer we used a modified Adam [110] described by Vaswani et al. [54] with a learning rate of 5. The transformer is configured to 25000 warmup iterations with a decreased learning rate to make initial training with a high learning rate more stable. The objective function is the sum of hybrid attention scaled by 0.7 and CTC scaled by 0.3 with a label smoothing [111] factor of 0.1. The transformer encoder's dropout rate is set to 0.1. For decoding we use a joint attention and CTC beam search decoder [82] with a beamsize of 60.

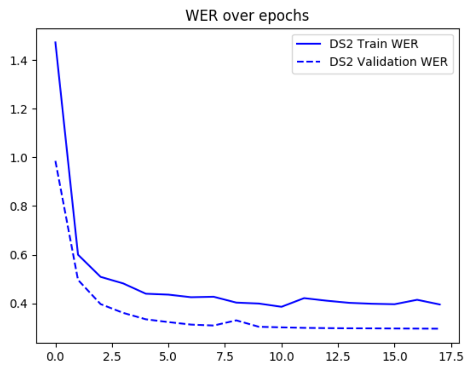
The DeepSpeech 2 architecture was trained on 4 Tesla K40 GPUs with a batch size of 10 per GPU. As optimizer we used Adam [110] with an initial learning rate of 0.00002 and a polynomial decrease learning rate policy [112] setting the learning rate lr to $initial\_lr \times (1 - \frac{iteration}{max\_iterations})^{0.5}$ at each iteration. We apply L^2 regularization [113] with $\lambda = 0.0005$ to all convolutional and feed forward layers. For decoding we use a greedy GPU based CTC decoder [11] with batch size 1.

The Wav2Letter architecture was trained on 4 Tesla K40 GPUs with a batch size of 8 per GPU. As optimizer we used Stochastic Gradient Descent with Momentum [114] of 0.9 with an initial learning rate of 0.05 and a polynomially decreasing learning rate policy [112] setting the learning rate lr to $initial\_lr \times (1 - \frac{iteration}{max\_iterations})^2$ at each iteration. We apply L^2 regularization [113] with $\lambda = 0.001$. We also employ layer-wise adaptive rate control (LARC[12]), an approach for adapting the learning rate separately for each layer based on LARS [115]. For decoding we use a greedy GPU based CTC decoder [13] with batch size 1.

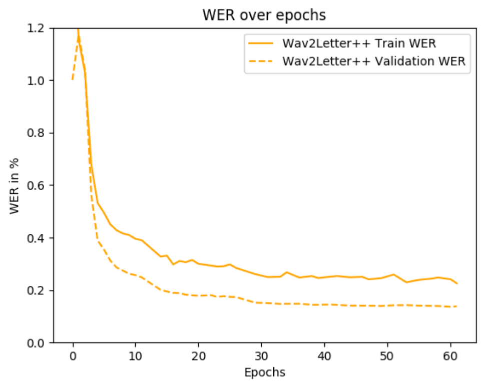
The TDNN-F/HMM architecture was trained on 1 Tesla K40 GPU with a batch size of 64. As optimizer we used Stochastic Gradient Descent with an initial learning rate of 0.00015 and an exponentially decreasing learning rate policy with a minimum of 0.000015. For regularization we use a leaky HMM coefficient [116] of 0.1, which enables the HMM to forget context over time. A leaky HMM coefficient allows to restart the HMM in any state at any timestep with a probability of the coefficient times the initial probability of that state. For decoding we use a beam search with a beam size of 15.



(a) ESPnet transformer training on Librispeech.



(b) DeepSpeech 2 training on Librispeech.



(c) Wav2Letter training on Librispeech.

Figure 4.1: Plots of WER accross training and validation sets over epochs.

[11] https://github.com/baidu-research/warp-ctc
[12] https://docs.nvidia.com/deeplearning/frameworks/caffe-user-guide/index.html#larc
[13] https://github.com/baidu-research/warp-ctc

Figure 4.1 plots the Word Error Rate of the different models for the combined training set train-clean-100, train-clean-360 and train-clean-500 and the combined validation set dev-clean and dev-other. The validation WER in Figure 4.1 is lower than the training accuracy. This makes sense because augmented data is part of the training data and batch normalization and dropout are only disabled for the evaluation of the validation set, but not for the training set. The ESPnet transformer architecture performed best in terms of WER including fast training times compared with the other models. Therefore, the ESPnet transformer architecture was chosen for the experiments in Section 4.2.2 and 4.2.3. Table 4.4 shows the WER scores for

Model	WER [%]			
	Without LM		With LM	
	dev-clean	test-clean	dev-clean	test-clean
DeepSpeech 2	28.19	29.39	-	6.85[14]
Wav2Letter	13.57	13.43	-	9.4 [63]
ESPnet transformer	**4.7**	**4.9**	3.7 [17]	4.0 [17]
Kaldi HMM/TDNN-F	5.44	5.90	**3.58**	**3.97**

Table 4.4: Accuracy of different acoustic models trained on Librispeech decoded without LM-rescoring and with LM-rescoring (reported by other papers).

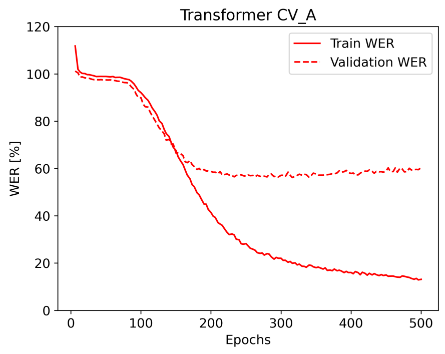
the Librispeech test-clean and dev-clean datasets. DeepSpeech 2 has a WER score of 29.39% on test-clean, but has been reported to achieve 6.85% WER on test-clean[15] with an n-gram LM. The Wav2Letter model trained on Mel-scaled filter-bank features (cf. Section 2.1.1) achieved a WER score of 13.43% on test-clean. Collobert et al. [63] report a relative improvement of 30% for their model trained on spectograms by using a 4-gram LM during decoding. The scores we achieved with ESPnet without LM rescoring are very low. In the ESPnet git repository[16] a relative improvement of 18% for test-clean is reported when rescoring with a transformer LM. The error rates for Kaldi HMM/TDNN-F with 4-gram LM rescoring improve on the HMM/TDNN-F without rescoring by relative 79% and are negligibly close to 3.29% WER on dev-clean and 3.80% WER on test-clean reported in the recipe published in the Kaldi Git repository[17].

4.2.2 Evaluation with the Mozilla Common Voice Corpus

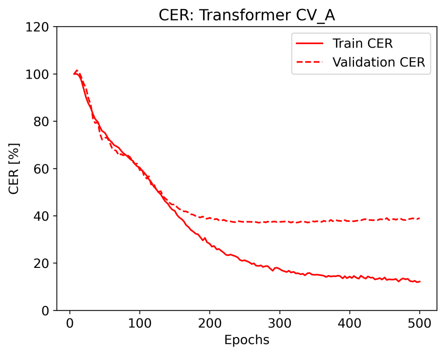
In this section we describe and evaluate the training of the acoustic model architecture ESPnet transformer from Section 3.1.2 on the German Mozilla Common Voice corpus cf. Section 4.1.2. The experiment was performed on a single NVIDIA Tesla K40 GPU and conducted without LM rescoring. As described in Section 3.1.2 the model uses subword units generated with SentencePiece [96] for its output layer. Since the German Mozilla Common Voice corpus does not have enough data to properly train 5000 subword units used in the Librispeech setup, the size of the subword units were set to 150, similiar to the Common Voice recipe provided by ESPnet. Figure 4.2 shows the accuracy of the transformer architecture on CV_A and on CV_B. The training set CV_B leads to considerably faster convergence compared to CV_A. This can be attributed to the fact that the data seen during one epoch of training on the set CV_B is almost seven times as much as is seen during an epoch of CV_A (cf. Section 4.1.2). The model trained on the CV_B reaches superior accuracy, but overfits around 100 epochs.

Table 4.5 reports the WER and CER scores for the ESPnet transformer architecture trained with the sets CV_A and CV_B (cf. Table 4.3) of the German Mozilla Common Voice corpus without LM. The CER of 41.3% of CV_A is superior to the CER of 43.76% reported by Ardila

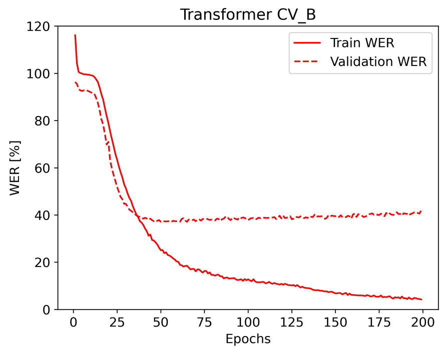
[14] https://github.com/PaddlePaddle/DeepSpeech
[15] https://github.com/PaddlePaddle/DeepSpeech
[16] https://github.com/espnet/espnet/blob/master/egs/librispeech/asr1/RESULTS.md
[17] https://github.com/kaldi-asr/kaldi/blob/master/egs/librispeech/s5/local/chain/tuning/run_TDNN-F_1d.sh



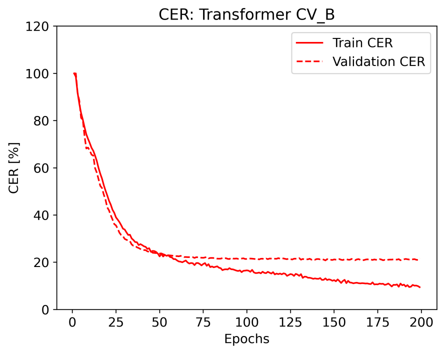
(a) WER of ESPnet transformer trained on CV_A.



(b) CER of ESPnet transformer trained on CV_A.



(c) WER of ESPnet transformer trained on CV_B.



(d) CER of ESPnet transformer trained on CV_B.

Figure 4.2: Plots of WER and CER accross training and validation sets of German Common Voice datasets (cf. Section 4.1.2).

Dataset	WER [%]		CER [%]	
	Validation	Test	Validation	Test
CV_A	56.2	60.2	37.1	41.3
CV_B	36.9	39.9	20.8	23.6

Table 4.5: Accuracy of ESPnet transformer architecture trained on different datasets without LM-rescoring.

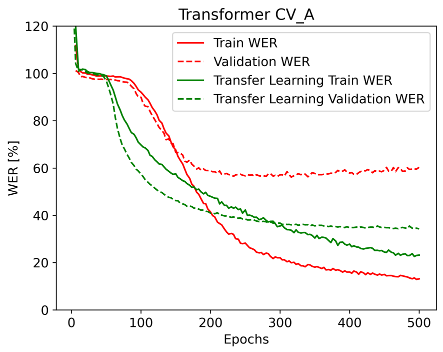
et al. [106], due to the use of an updated corpus. In the Common Voice recipe[18] a relative improvement of 38% is reported when decoding with RNN LM rescoring.

4.2.3 Evaluation of Transfer Learning Setups

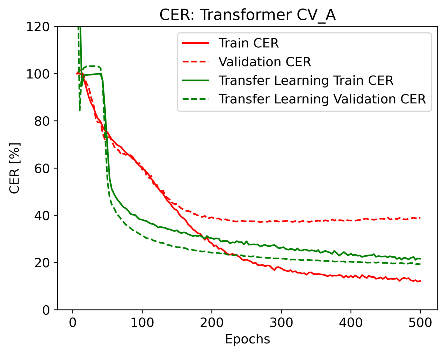
In this Section we describe and evaluate the training of the acoustic model architecture ESPnet transformer from Section 3.1.2 on the German Mozilla Common Voice corpus starting with a model pretrained on the Librispeech corpus (cf. Section 4.2.1). The experiment was performed on a single NVIDIA Tesla K40 GPU and conducted without LM rescoring. For transfer learning from a model pretrained on Librispeech we face the problem of special characters in German. The model trained on Librispeech can only transcribe one of the 5000 subwords in its output layer and they only contain English characters. Therefore, 150 new subwords are generated from the German Common Voice training set with SentencePiece [96] that we can use to replace the

[18] https://github.com/espnet/espnet/blob/master/egs/commonvoice/asr1/RESULTS.md

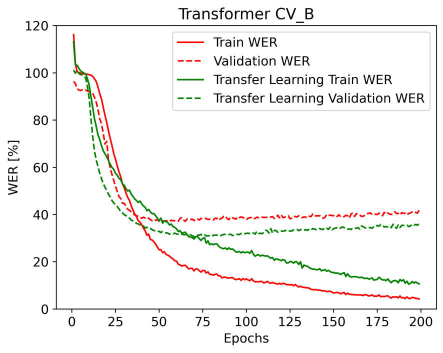
output layer. We use 150 subwords because it worked well in the previous experiment with the German Common Voice corpus in Section 4.2.2 and better than 5000 subwords used in the Librispeech recipe. To those subwords we add a blank and an unknown symbol resulting in 152 subword units. When the pretrained model is loaded the transformer's final softmax layer and the CTC softmax layer are randomly initialized and their output dimensions are adjusted to 152. The output embedding layer (used as input for the first decoding layer) is also randomly initialized and its input dimension is adjusted to 152. Then the model is trained without freezing any of the layers. Figure 4.3 shows the accuracy of the transformer architecture with transfer learning trained on the datasets CV_A and CV_B.



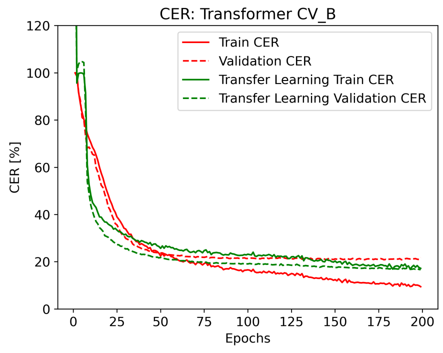
(a) WER of ESPnet transformer trained on CV_A dataset with and without TL.



(b) CER of ESPnet transformer trained on CV_A with and without TL.



(c) WER of ESPnet transformer trained on CV_B with and without TL.



(d) CER of ESPnet transformer trained on CV_B with and without TL.

Figure 4.3: WER and CER of transformer trained on German Common Voice datasets (cf. Section 4.1.2) starting from a model trained on Librispeech (cf. Section 4.2.1).

Dataset	WER [%]		CER [%]	
	Validation	Test	Validation	Test
CV_A	34.3	37.6	19.3	22.1
CV_B	30.6	33.5	16.7	19.3

Table 4.6: Accuracy of ESPnet transformer architecture trained on different datasets without LM-rescoring starting from a model pretrained on Librispeech.

The best WER and CER values for the two models are reported in Table 4.6. Compared to the experiment in Section 4.2.2 the transfer learning setup leads to a relative improvement in WER on test-clean of 37.5% for training set CV_A and 16% for set CV_B. As can be seen in Figure 4.3 the transfer learning setup has a steeper training curve. Training on CV_B converges considerably faster than on CV_A. This can be attributed to the fact that the data seen during one epoch of training on CV_B is almost seven times as much as is seen during an epoch of CV_A (cf. Section 4.1.2). After 200 epochs the model trained on CV_B has better accuracy than the one trained on CV_A, but WER starts increasing around 60 epochs.

4.3 Qualitative Comparison of ASR Frameworks

In this section we qualitatively compare the ASR frameworks Kaldi, ESPnet and OpenSeq2Seq used for the experiments. Table 4.7 shows the qualitative comparison of ASR frameworks. OpenSeq2Seq and ESPnet are both used to train and decode end-to-end (E2E) deep learning models with CTC and attention while Kaldi mainly builds on HMMs combined with DNNs or GMMs. All three frameworks support data augmentation, but they have different approaches to preprocessing and feature extraction. ESPnet and Kaldi perform preprocessing and feature extraction as a separate step in their recipe before training, while OpenSeq2Seq extracts features on the fly during training. This results in a disadvantage when comparing training time. All three toolkits can train on multi GPU setups for decreased training time. OpenSeq2Seq supports multi node training with Horovod [98] or distributed Tensorflow, while ESPnet does not support multi node training yet. Kaldi comes with scripts that can run distributed training on parallelization engines like Sun GridEngine, slurm and Tork. OpenSeq2Seq is the only framework of the three to support mixed precision training [117] for lower memory footprint and training speedup. Kaldi and OpenSeq2Seq both implement realtime inference and are therefore suitable for a setup as cloud service. Out of the box none of the frameworks provide options for inference on mobile devices, but there have been efforts by Gaida et al. [118] to run Kaldi model inference on the Android mobile platform. The Kaldi toolkit has a mixed codebase with binaries written in C++ chained together by Bash and Python scripts. ESPnet's codebase is comparable, with its main code written in Python chained together by Bash scripts. The code of OpenSeq2Seq is purely written in Python with the exception of the CTC decoder that is written in C++. Configuration for training, evaluation and inference in Kaldi and ESPnet is stored in a configuration folder, but additional parameters are adjustable in the recipes and Bash scripts used in the recipe. The OpenSeq2Seq toolkit keeps the whole configuration for preprocessing, training, decoding and evalution in a single configuration file per architecture.

Framework	**OpenSeq2Seq**	**ESPnet**	**Kaldi**
Approach	E2E Deeplearning	E2E Deeplearning	Hybrid HMM/DNN or HMM/GMM
Data Augmentation	Yes	Yes	Yes
Preprocessing & Feature Extraction	On the fly	Before training	Before training
Multi GPU Training	Yes	Yes	Yes
Multi Node Training	Yes	No	Yes
Mixed Precision Training	Yes	No	No
Realtime Inference	Yes	No	Yes
Mobile Inference	No	No	No
Codebase	Python	Python & Bash	C++, Bash & Python
Configuration	Single config file	Short recipe and multiple config files	Long recipe and multiple config files

Table 4.7: Comparison of the ASR frameworks used in this chapter.

4.4 Training and Inference Costs of ASR Frameworks

In the following we evaluate the training and inference costs of the model architectures described in Chapter 3. We measure and compare the time needed for training for one epoch on the Librispeech corpus and the time needed for inference of the test-clean set of the Librispeech corpus for each model. For the evaluation we used untuned and unoptimized framework code as provided by the corresponding repositories. All experiments are performed on a single NVIDIA Tesla K40c GPU with 12GB VRAM and an Intel Xeon E5-2697 v3 @ 2.60GHz CPU. Training is performed on the GPU and inference is performed either with or without GPU depending on what is supported by the framework. Table 4.8 shows the results of the training experiment. The

Model	CPUs	GPUs	Total epochs trained	Batch of 50 [sec]	Epoch [h]	Total [h]
DeepSpeech 2	1	1	16	7.17	11.2	179.2
Wav2Letter	1	1	90	**2.72**	**4.25**	382.5
ESPnet transformer	1	1	106	4.13	6.46	684.76
Kaldi HMM/TDNN-F	1	1	4	89.24	139.43	557.72

Table 4.8: Temporal performance of different acoustic models for training on Librispeech.

number of CPUs and GPUs used for training of the corresponding model and the total number of epochs trained is shown in each row. The respective training times for a batch of 50 utterances in seconds, a whole epoch in hours and the total training time in hours are reported. The total training time does not necessarily align with the epoch training time, because the models were not trained for the same amount of epochs due to time constraints. For every training experiment a single CPU and a single GPU were used.

Model	GPUs	Beam Size	Inference time for 1h [min]	test-clean [h]	Realtime capable
DeepSpeech 2	1	1	**1.33**	**0.12**	✓
Wav2Letter	1	1	1.78	0.16	✓
ESPnet transformer	1	60	527.33	47.46	✕
		1	3.33	0.3	✓
	0	60	441.89	39.77	✕
		1	34.56	3.11	✓
Kaldi HMM/TDNN-F	1	15	2.78	0.25	✓
		1	1.67	0.15	✓
	0	15	14.03	1.26	✓
		1	4.72	0.43	✓

Table 4.9: Temporal performance of different acoustic models for inference without LM-rescoring.

Table 4.9 shows the results of the inference experiment: The number of GPUs used for inference of the corresponding model is reported in each row. The respective inference times for an hour of audio data in minutes and the whole test set Librispeech test-clean in hours are reported. All models have been measured with a greedy decoder indicated by the beam size of 1. Additionally the timings of the transformer and the HMM/TDNN-F were measured with the beamsize used in the evaluation in terms of accuracy. As Kaldi and ESPnet support both GPU and CPU decoding we report both numbers for comparison. All decoding experiments are run in a single thread and all GPU based decoding experiments use a batch size of 1. The Kaldi HMM/TDNN-F has a very high training time per epoch, but compared to the end-to-end systems it only needs a small number of epochs for respectable WER scores so that its total training time is not considerably different than that of its alternatives. The CPU and GPU based inference of the transformer with a beam size of 60 is very slow with 39.77 and 47.46 hours for 5.4 hours of audio, rendering it unsuitable for realtime uses. However, with greedy decoding it provides performance faster than realtime. DeepSpeech2 provides the fastest inference time by decoding 1 hour of audio data in 1.33 minutes, but its implementation in OpenSeq2Seq does not support CPU decoding which can be a problem on devices where GPUs are not available. Wav2Letter is a close competitor to DeepSpeech 2 in terms of speed, but also does not support CPU decoding in the OpenSeq2Seq framework implementation. When decoded on the GPU the Kaldi HMM/TDNN-F provides competitive numbers for greedy decoding, taking the second best position in the ranking. The Kaldi HMM/TDNN-F decoded on the CPU also provides acceptable numbers, especially for greedy decoding where it takes 358% of the time the fastest model, DeepSpeech 2, takes on the GPU.

5 Conclusion

5.1 Conclusion and Future Outlook

In this work modern ASR models based on different technologies including hybrid HMM/TDNN-F, transformers, convolutional neural networks and RNNs were presented. In particular, the presented models are HMM/TDNN-F, ESPnet transformer, DeepSpeech 2 and Wav2Letter. For conducting the evaluations the frameworks Kaldi, ESPnet and OpenSeq2Seq implementing those models were selected. The frameworks were used to train ASR models on the English Librispeech corpus containing 980 hours of read speech. The resulting models were then compared in terms of word error rate. ESPnet transformer, the best performing end-to-end model, was used for evaluation on the German Mozilla Common Voice corpus with 324 hours of read speech. In particular, the transformer was trained with the German Common Voice corpus with and without transfer learning setup. Without the transfer learning setup it was trained from scratch with randomly initialized weights. For the transfer learning setup the model previously trained on the Librispeech corpus was used as weight initialization. Additionally, its output layer was replaced with a layer of a size suitable for classifying frames into German subword units. Following the training we reported and evaluated the results of the experiment. Furthermore, the used frameworks OpenSeq2Seq, Kaldi and ESPnet were compared based on their technology, features and usability. Additionally, the different models were compared in terms of training time and inference time.

This work provides an evaluation in terms of WER of the ASR architectures HMM/TDNN-F, transformer, Wav2Letter and DeepSpeech2 using the Librispeech corpus. Additionally, we verified that ESPnet transformer model accuracy can benefit from transfer learning. In particular, training a German model with a model pretrained on English Librispeech as initialization is superior to training with random initialization for the ESPnet transformer. Furthermore, the training and inference performance of the trained models as well as the feature sets and usability of the the frameworks OpenSeq2Seq, Kaldi and ESPnet were compared. The first evaluation in terms of accuracy was performed using the Librispeech dataset. The ESPnet transformer reached the best accuracy with a WER score of 4.9 on Librispeech test-clean. The second evaluation in terms of WER was performed using the German Common voice corpus CV_B. The ESPnet transformer achieved WER scores of 39.9 and 33.5 on the test set for regular training and transfer learning respectively. In the comparison between regular training and transfer learning using the German Common Voice corpus CV_A the ESPnet transformer reached WER scores of 60.2 and 37.6 on the test set respectively. This shows that initialization with a model pretrained on Librispeech improves the accuracy of the ESPnet transformer model trained on the German Common Voice corpus. The relative improvement due to transfer learning is larger when training on the small dataset CV_A than with the large dataset CV_B. Therefore it appears that transfer learning has a higher impact on accuracy when the amount of training data is small.

The performance evaluation in terms of inference time shows that DeepSpeech2 is the most efficient of the evaluated architectures in terms of inference speed on the GPU. For CPU inference the HMM/TDNN-F is the most efficient. Furthermore, the evaluation shows that the beam search decoder implemented in ESPnet is very slow and can only be used in realtime scenarios with greedy decoding. The performance evaluation of inference time allows the conclusion that DeepSpeech2, Wav2Letter, ESPnet transformer and Kaldi HMM/TDNN-F are suitable for realtime use. However, the beam size of the ESPnet transformer decoder has to be reduced from 60 to 1. Hence, the accuracy suffers under the realtime constraint. All frameworks could be deployed in a cloud environment, but none of them support running on mobile devices out of the box.

This research indicates that pure CTC architectures are in need of LMs to boost classification accuracy. The transformer architecture reduces the need for a LM and is therefore an interesting model for future research. Although much progress has been made in end-to-end ASR the quality of the model is still highly dependent on the amount of training data in the target language. We have improved a transformer model with transfer learning which is a good candidate for leveraging data in other languages. Decoding in realtime is an important factor in perceived latency of an ASR system. To use the ESPnet transformer in realtime an attention mechanism with limited lookahead needs to be used and a framewise decoder must be implemented. Overall we observe that all evaluated models are suitable as cloud solutions. However, when it comes to edge deployment additional research regarding resource efficiency of acoustic- and language models needs to be conducted.

Bibliography

[1] V. Peddinti, D. Povey, and S. Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[2] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[3] H. W. Dudley, "Sound printing mechanism," Mar. 26 1940, uS Patent 2,195,081.

[4] K. H. Davis, R. Biddulph, and S. Balashek, "Automatic recognition of spoken digits," *The Journal of the Acoustical Society of America*, vol. 24, no. 6, pp. 637–642, 1952.

[5] M. Halle and K. Stevens, "Speech recognition: A model and a program for research," *IRE transactions on information theory*, vol. 8, no. 2, pp. 155–159, 1962.

[6] T. Sakai and S. Doshita, "The automatic speech recognition system for conversational sound," *IEEE Transactions on Electronic Computers*, no. 6, pp. 835–846, 1963.

[7] D. Reddy, L. Erman, and R. Neely, "A model and a system for machine recognition of speech," *IEEE Transactions on Audio and Electroacoustics*, vol. 21, no. 3, pp. 229–238, 1973.

[8] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

[9] J. Baker, "The dragon system–an overview," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 24–29, 1975.

[10] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi, "An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition," *Bell System Technical Journal*, vol. 62, no. 4, pp. 1035–1074, 1983.

[11] M. Russell and R. Moore, "Explicit modelling of state occupancy in hidden markov models for automatic speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 10. IEEE, 1985, pp. 5–8.

[12] C. Lüscher, E. Beck, K. Irie, M. Kitza, W. Michel, A. Zeyer, R. Schlüter, and H. Ney, "Rwth asr systems for librispeech: Hybrid vs attention," *INTERSPEECH*, pp. 231–235, 2019.

[13] B. T. Lowerre, "The harpy speech recognition system," Carnegie-Mellon University Pittsburgh Department of Computer Science, Tech. Rep., 1976.

[14] W. A. Lea and J. E. Shoup, "Review of the arpa sur project and survey of current technology in speech understanding." Speech Communications Research Lab Los Angeles CA, Tech. Rep., 1979.

[15] P. Mermelstein, "Distance measures for speech recognition, psychological and instrumental," *Pattern recognition and artificial intelligence*, vol. 116, pp. 374–388, 1976.

[16] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. CONF. IEEE Signal Processing Society, 2011.

[17] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen *et al.*, "Espnet: End-to-end speech processing toolkit," *arXiv preprint arXiv:1804.00015*, 2018.

[18] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE transactions on acoustics, speech, and signal processing*, vol. 28, no. 4, pp. 357–366, 1980.

[19] F. Jelinek, "Continuous speech recognition by statistical methods," *Proceedings of the IEEE*, vol. 64, no. 4, pp. 532–556, 1976.

[20] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 2, pp. 179–190, 1983.

[21] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[22] R. P. Lippmann, "Review of neural networks for speech recognition," *Neural computation*, vol. 1, no. 1, pp. 1–38, 1989.

[23] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition: neural networks vs. hidden markov models vs. hidden markov models," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).* IEEE, 1988, pp. 107–110.

[24] D. J. Burr, "Speech recognition experiments with perceptrons," in *Neural Information Processing Systems*, 1988, pp. 144–153.

[25] K.-F. Lee, *Automatic speech recognition: the development of the SPHINX system.* Springer Science & Business Media, 1988, vol. 62.

[26] K.-F. Lee, H.-W. Hon, and R. Reddy, "An overview of the sphinx speech recognition system," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 1, pp. 35–45, 1990.

[27] L. R. Bahl, P. F. Brown, P. V. De Souza, and R. L. Mercer, "Maximum mutual information estimation of hidden markov model parameters for speech recognition," in *ICASSP*, vol. 86, 1986, pp. 49–52.

[28] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International conference on machine learning*, 2014, pp. 1764–1772.

[29] H. A. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach.* Springer Science & Business Media, 1994, vol. 247.

[30] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[31] J. S. Bridle, "Alpha-nets: a recurrent 'neural'network architecture with a hidden markov model interpretation," *Speech Communication*, vol. 9, no. 1, pp. 83–92, 1990.

[32] R. Lippmann and B. Gold, "Neural classifiers useful for speech recognition," *Neural Networks, pp. IV-417, San Diego, CA*, 1987.

[33] M. Franzini, K.-F. Lee, and A. Waibel, "Connectionist viterbi training: a new hybrid method for continuous speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing.* IEEE, 1990, pp. 425–428.

[34] H. Bourlard and N. Morgan, "A continuous speech recognition system embedding mlp into hmm," in *Advances in neural information processing systems*, 1990, pp. 186–193.

[35] B.-H. Juang, W. Hou, and C.-H. Lee, "Minimum classification error rate methods for speech recognition," *IEEE Transactions on Speech and Audio processing*, vol. 5, no. 3, pp. 257–265, 1997.

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[37] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[38] H. Hermansky, D. P. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional hmm systems," in *IEEE International Conference on Acoustics, Speech, and Signal Processing.*, vol. 3. IEEE, 2000, pp. 1635–1638.

[39] D. Pearce, "Aurora project: Experimental framework for the performance evaluation of distributed speech recognition front-ends," ETSI working paper, Tech. Rep., 1998.

[40] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning.* ACM, 2006, pp. 369–376.

[41] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[42] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE international conference on acoustics, speech and signal processing.* IEEE, 2013, pp. 6645–6649.

[43] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[44] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[45] P. Cosi, "A kaldi-dnn-based asr system for italian," in *International Joint Conference on Neural Networks (IJCNN).* IEEE, 2015, pp. 1–5.

[46] I. Kipyatkova and A. Karpov, "Dnn-based acoustic modeling for russian speech recognition using kaldi," in *International Conference on Speech and Computer.* Springer, 2016, pp. 246–253.

[47] P. Upadhyaya, O. Farooq, M. R. Abidi, and Y. V. Varshney, "Continuous hindi speech recognition model based on kaldi asr toolkit," in *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET).* IEEE, 2017, pp. 786–789.

[48] S. Watanabe, T. Hori, S. Kim, J. R. Hershey, and T. Hayashi, "Hybrid ctc/attention architecture for end-to-end speech recognition," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 8, pp. 1240–1253, 2017.

[49] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in neural information processing systems*, 2015, pp. 577–585.

[50] L. Wu, T. Li, L. Wang, and Y. Yan, "Improving hybrid ctc/attention architecture with time-restricted self-attention ctc for end-to-end speech recognition," *Applied Sciences*, vol. 9, no. 21, p. 4639, 2019.

[51] C.-C. Chiu and C. Raffel, "Monotonic chunkwise attention," *arXiv preprint arXiv:1712.05382*, 2017.

[52] N. Jaitly, Q. V. Le, O. Vinyals, I. Sutskever, D. Sussillo, and S. Bengio, "An online sequence-to-sequence model using partial conditioning," in *Advances in Neural Information Processing Systems*, 2016, pp. 5067–5075.

[53] R. Fan, P. Zhou, W. Chen, J. Jia, and G. Liu, "An online attention-based model for speech recognition," *arXiv preprint arXiv:1811.05247*, 2018.

[54] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[55] S. Zhou, L. Dong, S. Xu, and B. Xu, "Syllable-based sequence-to-sequence speech recognition with the transformer in mandarin chinese," *arXiv preprint arXiv:1804.10752*, 2018.

[56] Y. A. Ibrahim, J. C. Odiketa, and T. S. Ibiyemi, "Preprocessing technique in automatic speech recognition for human computer interaction: An overview," *Annals. Computer Science Series*, vol. 15, no. 1, 2017.

[57] S. Winograd, "On computing the discrete fourier transform," *Mathematics of computation*, vol. 32, no. 141, pp. 175–199, 1978.

[58] F. Hönig, G. Stemmer, C. Hacker, and F. Brugnara, "Revising perceptual linear prediction (plp)," in *Ninth European Conference on Speech Communication and Technology*, 2005.

[59] B. Roark, M. Saraclar, and M. Collins, "Discriminative n-gram language modeling," *Computer Speech & Language*, vol. 21, no. 2, pp. 373–392, 2007.

[60] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR, 2017, pp. 933–941.

[61] F. Richardson, M. Ostendorf, and J. R. Rohlicek, "Lattice-based search strategies for large vocabulary speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 1995, pp. 576–579.

[62] G. Synnaeve, Q. Xu, J. Kahn, E. Grave, T. Likhomanenko, V. Pratap, A. Sriram, V. Liptchinsky, and R. Collobert, "End-to-end asr: from supervised to semi-supervised learning with modern architectures," *arXiv preprint arXiv:1911.08460*, 2019.

[63] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *arXiv preprint arXiv:1609.03193*, 2016.

[64] A. Graves, "Hierarchical subsampling networks," in *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, pp. 113–116.

[65] D. Palaz, R. Collobert, and M. M. Doss, “Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks,” *arXiv preprint arXiv:1304.1018*, 2013.

[66] C.-T. Do and Y. Stylianou, “Improved automatic speech recognition using subband temporal envelope features and time-delay neural network denoising autoencoder.” in *INTERSPEECH*, 2017, pp. 3832–3836.

[67] S. S. Stevens, J. Volkmann, and E. B. Newman, “A scale for the measurement of the psychological magnitude pitch,” *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[68] A.-r. Mohamed, G. Hinton, and G. Penn, “Understanding how deep belief networks perform acoustic modelling,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2012, pp. 4273–4276.

[69] H. J. Nussbaumer, “The fast fourier transform,” in *Fast Fourier Transform and Convolution Algorithms.* Springer, 1981, pp. 80–111.

[70] B. Logan *et al.*, “Mel frequency cepstral coefficients for music modeling.” in *ISMIR*, vol. 270, 2000, pp. 1–11.

[71] M. R. Hasan, M. Jamil, M. Rahman *et al.*, “Speaker identification using mel frequency cepstral coefficients,” *Proceedings of the 3rd International Conference on Electrical and Computer Engineering (ICECE 2004)*, vol. 1, no. 4, 2004.

[72] L. Muda, M. Begam, and I. Elamvazuthi, “Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques,” *arXiv preprint arXiv:1003.4083*, 2010.

[73] P. P. Das, S. M. Allayear, R. Amin, and Z. Rahman, “Bangladeshi dialect recognition using mel frequency cepstral coefficient, delta, delta-delta and gaussian mixture model,” in *Eighth International Conference on Advanced Computational Intelligence (ICACI).* IEEE, 2016, pp. 359–364.

[74] L. Deng and X. Li, “Machine learning paradigms for speech recognition: An overview,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 5, pp. 1060–1089, 2013.

[75] K. Heafield, “Kenlm: Faster and smaller language model queries,” in *Proceedings of the sixth workshop on statistical machine translation.* Association for Computational Linguistics, 2011, pp. 187–197.

[76] O. Kuchaiev, B. Ginsburg, I. Gitman, V. Lavrukhin, J. Li, H. Nguyen, C. Case, and P. Micikevicius, “Mixed-precision training for nlp and speech recognition with openseq2seq,” 2018.

[77] V. Pratap, A. Hannun, Q. Xu, J. Cai, J. Kahn, G. Synnaeve, V. Liptchinsky, and R. Collobert, “wav2letter++: The fastest open-source speech recognition system,” *arXiv preprint arXiv:1812.07625*, 2018.

[78] L. R. Rabiner, “A tutorial on hidden markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[79] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, “A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains,” *The annals of mathematical statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[80] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[81] X. Huang, A. Acero, H.-W. Hon, and R. Foreword By-Reddy, *Spoken language processing: A guide to theory, algorithm, and system development.* Prentice hall PTR, 2001.

[82] T. Hori, S. Watanabe, and J. R. Hershey, "Joint ctc/attention decoding for end-to-end speech recognition," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 2017, pp. 518–529.

[83] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[84] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[85] A. Graves, "Connectionist temporal classification," in *Supervised Sequence Labelling with Recurrent Neural Networks.* Springer, 2012, pp. 61–93.

[86] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition." in *INTERSPEECH*, 2017, pp. 939–943.

[87] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[88] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[89] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[90] D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, M. Yarmohammadi, and S. Khudanpur, "Semi-orthogonal low-rank matrix factorization for deep neural networks." in *INTERSPEECH*, 2018, pp. 3743–3747.

[91] H. Hermansky, "Perceptual linear predictive (plp) analysis of speech," *the Journal of the Acoustical Society of America*, vol. 87, no. 4, pp. 1738–1752, 1990.

[92] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Springer Handbook of Speech Processing.* Springer, 2008, pp. 559–584.

[93] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2010.

[94] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of dnns with natural gradient and parameter averaging," *arXiv preprint arXiv:1410.7455*, 2014.

[95] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[96] T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *arXiv preprint arXiv:1808.06226*, 2018.

[97] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, 2016, pp. 173–182.

[98] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[99] J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. T. Gadde, "Jasper: An end-to-end convolutional neural acoustic model," 2019.

[100] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions," *arXiv preprint arXiv:1910.10261*, 2019.

[101] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[102] A. Hannun, A. Lee, Q. Xu, and R. Collobert, "Sequence-to-sequence speech recognition with time-depth separable convolutions," 2019.

[103] S. Radeck-Arneth, B. Milde, A. Lange, E. Gouvêa, S. Radomski, M. Mühlhäuser, and C. Biemann, "Open source german distant speech recognition: Corpus and acoustic model," in *International Conference on Text, Speech, and Dialogue.* Springer, 2015, pp. 480–488.

[104] B. Milde and A. Köhn, "Open source automatic speech recognition for german," in *Speech Communication; 13th ITG-Symposium.* VDE, 2018, pp. 1–5.

[105] M. Hart, "The history and philosophy of project gutenberg," *Project Gutenberg*, vol. 3, pp. 1–11, 1992.

[106] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common voice: A massively-multilingual speech corpus," *arXiv preprint arXiv:1912.06670*, 2019.

[107] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[108] I. A. McCowan, D. Moore, J. Dines, D. Gatica-Perez, M. Flynn, P. Wellner, and H. Bourlard, "On the use of information retrieval measures for speech recognition evaluation," IDIAP, Tech. Rep., 2004.

[109] T. Hayashi, R. Yamamoto, K. Inoue, T. Yoshimura, S. Watanabe, T. Toda, K. Takeda, Y. Zhang, and X. Tan, "Espnet-tts: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit," *arXiv preprint arXiv:1910.10909*, 2019.

[110] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[111] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[112] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," *arXiv preprint arXiv:1506.04579*, 2015.

[113] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[114] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *International conference on machine learning*, 2013, pp. 1139–1147.

[115] Y. You, I. Gitman, and B. Ginsburg, “Large batch training of convolutional networks,” *arXiv preprint arXiv:1708.03888*, 2017.

[116] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, “Purely sequence-trained neural networks for asr based on lattice-free mmi.” in *INTERSPEECH*, 2016, pp. 2751–2755.

[117] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, “Mixed precision training,” *arXiv preprint arXiv:1710.03740*, 2017.

[118] C. Gaida, R. Petrick, and D. Suendermann-Oeft, “Kaldi goes android,” in *INTERSPEECH*, 2016.