BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

THE BEST POSSIBLE SPEECH RECOGNIZER ON YOUR OWN DATA

CO NEJLEPŠÍ ROZPOZNÁVAČ ŘEČI NA VLASTNÍCH DATECH

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **TOMÁŠ SÝKORA**
AUTOR PRÁCE

SUPERVISOR **Ing. IGOR SZŐKE, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Abstract

Many state-of-the-art results in different machine learning areas are presented on day-to-day basis. By adjusting these systems to perform perfectly on a specific subset of all general data, huge improvements may be achieved in their resulting accuracy. Usage of domain adaptation in automatic speech recognition can bring us to production level models capable of transcribing difficult and noisy customer conversations way more accurately than the general models trained on all kinds of language and speech data. In this work I present 17% word error rate improvement in our speech recognition task over the general domain speech recognizer from Google. The improvement was achieved by both very precise annotation and preparation of domain data and by combining state-of-the-art techniques and algorithms. The described system was successfully integrated into a production environment of the Parrot transcription company, where I am a member of the initial team, which drastically increased performance of the human transcribers.

Abstrakt

Denno-denne vzniká množstvo špičkových objavov v oblasti strojového učenia. Prispôsobením týchto systémov tak, aby čo najlepšie fungovali iba na obmedzenej podmnožine všeobecných dát, môžu byť dosiahnuté výrazné zlepšenia v prenosti. Prispôsobením automatického rozpoznávača reči na doménovo špecifické dáta je možné vytvoriť produkt dosahujúci omnoho lepšie výsledky ako rozpoznávač reči natrénovaný na všeobecných dátach. Táto práca prezentuje 17-percentné zlepšenie chybovosti prepísaných slov oproti automatickému rozpoznávaču reči ponúkaného službou Google Speech. Toto zlepšenie bolo dosiahnuté precíznou anotáciou a prípravou doménových dát a kombináciou špičkových techník a algoritmov v oblasti automatického rozpoznávania reči. Popísaný systém bol úspešne nasadený do výrobného prostredia transkripčnej spoločnosti Parrot, ktorej súčasťou som od jej začiatku. Nasadený systém výrazne zvýšil efektivitu zamestancov používajúcich výstup popísaného rozpoznávača.

Keywords

automatic speech recognition, domain data, kaldi, dataset, speech data cleaning

Kľúčové slová

automatické rozpoznávanie reči, doménové dáta, kaldi, dataset, čistenie rečových dát

Reference

SÝKORA, Tomáš. *The Best Possible Speech Recognizer on Your Own Data.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Igor Szőke, Ph.D.

The Best Possible Speech Recognizer on Your Own Data

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Igor Szőke Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Tomáš Sýkora
June 3, 2020

Acknowledgements

I would like to thank my supervisor Igor Szőke for his advice during this project.

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Master's Thesis Specification

Student: **Sýkora Tomáš, Bc.**
Programme: Information Technology Field of study: Intelligent Systems
Title: **The Best Possible Speech Recognizer on Your Own Data**
Category: Speech and Natural Language Processing
Assignment:

1. Get familiar with automatic speech recognition (ASR) and Kaldi toolkit.
2. Train a speech recognizer on publicly available data.
3. Prepare your own training data.
4. Add your data into the training process of the speech recognizer and evaluate it.
5. Tweak your data (cleaning, augmentation, etc.) and mix it with public data in various ways. Evaluate your ASR continuously.
6. Discuss achieved results and future work.
7. Make an A2 poster and a short video presenting your work.

Recommended literature:

- according to supervisor's recommendation.

Requirements for the semestral defence:

- Items 1, 2, 3, part of item 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Szőke Igor, Ing., Ph.D.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: June 3, 2020
Approval date: May 27, 2020

Contents

Chapter 1

Introduction

Automatic speech recognition (ASR) has achieved many improvements over the last decade similarly to other artificial intelligence and machine learning areas. New algorithms and more computational power, especially in a form of graphics processing units (GPUs), allowed researchers to feed the systems with much more data than before which markedly boosted results and accuracies of such systems. As a result, machine learning products can be now integrated in various day-to-day scenarios where they can automate the previously applied manual processes and save both time and money of companies using them.

Automatic speech recognition is a system capable of transcribing speech from the audio signal form to text. ASR can improve many existing areas like movie captions generation, or solve completely new problems like automatic meetings and phone calls transcriptions which was too time consuming and inefficient to do so manually in the past. One of the areas opened to automation is the court reporting industry in the USA. All courts, lawyers, attorneys and other persons and entities in the legal industry need transcriptions of depositions, witness statements or phone calls on day-to-day basis. Currently, court reporters take care of their needs and transcribe the audio recordings containing conversations into text manually. But as this job is highly time consuming and there is not enough court reporters to transcribe everything the legal industry needs, the automatic speech recognition systems may be of great help here.

In Parrot we use ASR as a helper tool for our human transcribers. After obtaining a transcription request from a customer and generating the automatic transcription, the human transcribers fix errors in it using a specialised editor tool. This approach is way more efficient than transcribing the whole recordings from scratch manually. To improve the experience and efficiency of the human transcribers we decided to build a more accurate ASR system. As until now we've been using a general speech recognition, I show in this thesis that a domain specific system can outperform a general one. The main reason for this being possible is no need of transcribing all kinds of target domains, situations, acoustic setups and specific language dialects. Instead, the model focuses only on scenarios that occur commonly in the legal space. Although this way our models may perform worse in general, on the other hand they perform way better in our target domain where the speech recognition model knows the data well as it was trained on similar ones.

The goal of this work was to achieve higher transcription accuracy on our legal based audio conversations than the one obtained with a general model. The general model used in our company until now was the one developed by Google[1]. The thesis demonstrates a

[1] https://cloud.google.com/speech-to-text/

contribution of the individual minor and major enhancements implemented to boost the final model performance. In this thesis, I describe how automatic speech recognition systems are built in general in Chapter 2. I explain how I collected and prepared the dataset for training the new domain specific ASR system in Chapter 3 and how I trained the ASR system itself with this dataset in Chapter 4. Finally, I show how the resulting system performed in comparison to the general one and what accuracy it achieved in Chapter 5. Results of different setups are presented and contributions of the individual enhancement features are shown in this chapter.

Chapter 2

Current trends in automatic speech recognition

This chapter describes how automatic speech recognition systems are built in general and what algorithms, approaches and principles are widely used among the researchers to achieve state-of-the-art results. I explain basic terminologies like word error rate (WER), main components of speech recognition systems and major and minor enhancements to these systems improving their final transcription accuracy.

Automatic speech recognition (ASR) is a system capable of recognising word sequences in a given speech signal and converting them into a readable text form. It can be a limited set of words, e.g. commands to control a device where a small set of all recognisable words is specified upfront [40]. Another common use case of ASR systems is recognising long word sequences and continuous speech parts composed of words defined by a large vocabulary or lexicon. While the first use case of the small vocabulary recognition may contain only several dozen words, the other one may have to be capable of recognising a vocabulary of tens to hundreds of thousands words which is obviously a much more difficult challenge. Large vocabulary continuous speech recognition (LVCSR) [35] systems have to be capable of recognising words in day-to-day conversations, meetings or phone calls. Complexity of such tasks increases drastically in comparison to the ones solved by small vocabulary systems. To be more specific, according to WordNet, the English language contains more than 200,000 words [8]. While learning these words by an ASR system, different word pronunciations have to be also considered which increases the lexicon size even more.

To solve this difficult problem, ASR systems often stand on complex architectures composed of several subsystems. Specifically, acoustic models (AM) are trained to match every word in a speech signal to a sequence of acoustic units called phonemes. To do this, Hidden Markov models (HMM) [34] are currently one of the state-of-the-art approaches to solve this sub-problem. However, there are words with similar pronunciation but different transcriptions. Additionally to that, different background noises may degrade the acoustic signal and make the phoneme or word recognising even more challenging. Thinking about how humans percept the speech, we can often understand the specific speech sequence although some words aren't pronounced perfectly or are covered with noise. It is because of a word context. To simulate this human perception step language models (LM) predict probabilities of word sequences in a given language. As an example, the *„I am“* word pair have a higher probability than *„I are“* in the English language.

Acoustic and language models both create a core of each ASR system. Together they search for a word sequence $W = w_1, w_2, ..., w_n$ with a maximum posterior probability given the input acoustic observations $X = x_1, x_2, ..., x_n$. The acoustic and language model combination can be expressed in a form of the Bayesian rule shown in Equation 2.1, where the probability $P(X|W)$ is modeled by an acoustic model (finding the highest probability of the acoustic observation given the word sequence), while a language model estimates and models the second part of the rule, $P(W)$. Assuming $P(X)$ is the same for all possible transcriptions, we do not have to consider that at all and only the numerator part of the rule remains to be realised by an ASR system.

$$W^* = \arg\max_w P(W|X) = \arg\max_w \frac{P(X|W)P(W)}{P(X)} \tag{2.1}$$

Predictions of both acoustic and language models together with other support components like lexicon and context perception are taken as an input into a decoding network. All components of an automatic speech recognition system and the whole process from capturing the acoustic signal to the decoding network producing text is shown in Figure 2.1.

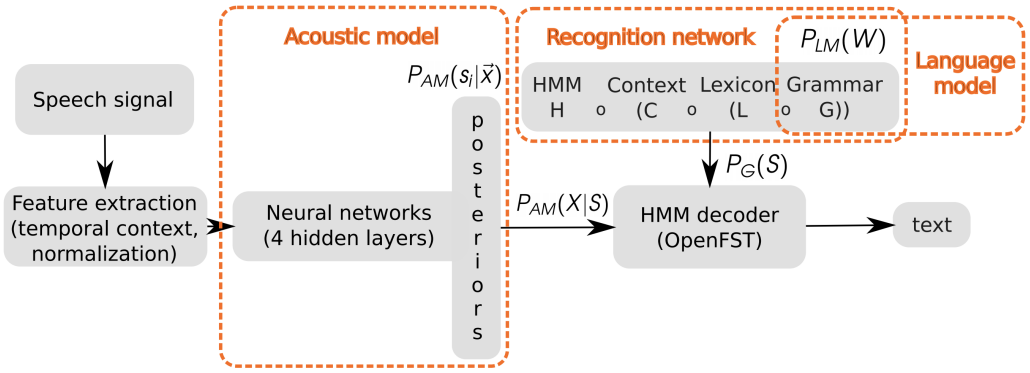

Figure 2.1: Components of an automatic speech recognition system transcribing a speech signal to text (figure obtained from [38]).

Word error rate

Word error rate (WER) is the most common technique for measuring accuracy of an ASR system. A predicted word sequence is at first aligned with the given ground truth transcription. The WER value is then computed as shown in Equation 2.2, where S is number of substitutions, D is number of deletions, I is number of insertions and C is number of the correct words.

$$WER = \frac{S + D + I}{S + D + C} \tag{2.2}$$

2.1 Feature extraction

Not all of the information in the high dimensional data recorded by a recording device are important for speech recognition. Some information can even make the recognition more difficult and worsen the results. The sounds generated by humans are shaped by their vocal tracts including their teeth, tongue etc. The way the vocal tract is used determines which phonemes are spoken out loud. Besides carrying the phoneme information, the audio carries other data, e.g. the speaker gender, mood, background noise etc. All of these are undesired in the speech recognition process because the information is irrelevant during the recognition. That is why the raw audio signal itself is usually not used as the input to ASR systems and is instead processed before being passed further.

A necessary step before passing a speech signal into an ASR system is feature extraction. It is a process of creating suitable data representation which is both compact and carries the necessary information for later recognising. There are multiple popular feature types like Mel Frequency Cepstral Coefficients [24], Linear Prediction Coefficients [11] or Linear Prediction Cepstral Coefficients. In this work Mel Frequency Cepstral Coefficients (MFCC) were used as they are one of the most commonly used feature types and are proven over the years and the research community to be robust and suitable for speech recognition.

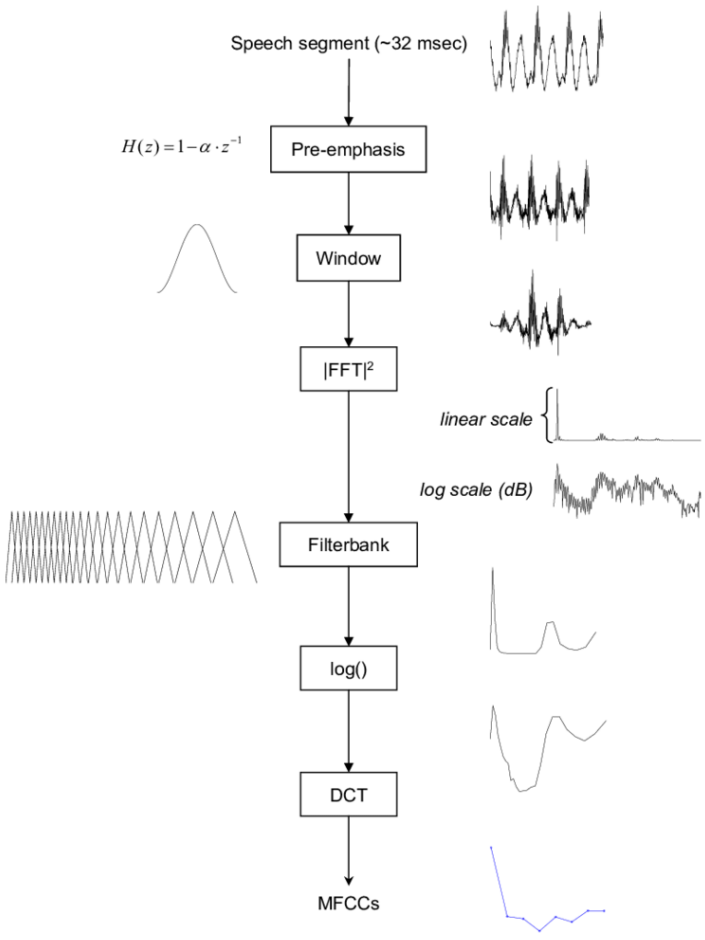

Figure 2.2: Mel Frequency Cepstral Coefficients features extraction process (figure obtained from [18]).

MFCC extraction is performed in several steps which can be seen in Figure 2.2. The signal is at first cut into short frames. They are short enough to be statistically non-changing and long enough to carry enough information. Their length is typically between 20-40ms. Similarly to human ears reacting differently to different frequencies, power spectrum is computed from each frame using the discrete Fourier transform. After that Mel filterbank is applied on the spectrum which is inspired by better human sensitivity to lower frequencies. As humans don't hear loudness on a linear scale, logarithms of filterbank energies are taken as input to discrete cosine transform (DCT) in the last step, where DCT assures decorrelation of the overlapping filterbank energies.

Speaker identity vectors for speaker adaptation

For an acoustic model to be more robust to channel and speaker variability, it can be adapted to a target speaker by supplying speaker identity vectors (i-vectors) as input features to the model in parallel with the regular acoustic features [33]. An i-vector of a given speaker is concatenated to every frame belonging to that speaker before being passed to a recognition system. The i-vectors are constant for a particular speaker and change across different speakers. As additional features they proved to provide significant gains in the WER scores, see the original paper for more detailed info.

$$M = m + Tw \tag{2.3}$$

Equation 2.3 represents a speaker and channel dependent supervector M (stacked vectors of GMMs (more about GMMs in Section 2.3) representing a high dimensional space), where m are means of a universal background model (UBM - a GMM modelling many speakers), w is an i-vector and T is a total variability matrix which transforms the i-vector w from its subspace into the high dimensional space of the UBM with means m. To extract an i-vector from an input utterance, an i-vector extractor has to be trained. The i-vector extractor is defined by m, T and Σ (a diagonal covariance matrix of the UBM). The parameters are estimated using an expectation maximization (EM) algorithm. The EM algorithm iteratively estimates the parameters of the model (m, T, Σ) in two repetitious steps:

- Expectation step - for each utterance u, calculate the parameters of the posterior distribution of $w(u)$ using the current estimates of m, T, Σ.
- Maximization step - update T and Σ by solving a set of linear equations in which the $w(u)$'s play the role of explanatory variables

2.2 Hidden Markov models

Hidden Markov model (HMM) is a statistical model computing joint probability of a set of hidden states given a set of observed states. Once we know the joint probability of a sequence of hidden states, we determine the most probable sequence of observations. The purpose of HMMs in speech recognition is to find a phoneme sequence (hidden states) with maximum posterior probability given the incoming acoustic signals (observed states) which is the $P(X|W)$ part of Equation 2.1. HMM is a generative model with similar characteristics to finite state automatons. It consists of probabilistic transitions a_{ij} from a state i to a state j. The topology of ASR HMMs is left to right so that the model can accept incoming

speech sequences (frames). For each frame the model moves from one state to another according to a specific transition probability.

An HMM is fully defined by a state transition probability matrix A (probability of moving from one state to another), an observation emission probability matrix B (probability of observing a particular output given being in a particular state) and an initial state probabilities λ (probability that the sequence starts in any of the possible states). Thus an HMM as a whole is defined as $\pi(A, B, \lambda)$. Given the list of observations X the goal is to set the HMM parameters using training data so that the model is able to answer the following questions:

- What are the probabilities of observing X, i.e. $P(X|\lambda)$
- What is the most likely sequence of hidden states which yielded the observed sequence, i.e. $\arg\max_x P(W|X)$

2.3 Gaussian mixture models

To estimate how well a feature vector fits a certain HMM state representing a specific phoneme, various machine learning methods can be applied. Before the neural networks, Gaussian Mixture Models (GMM) were a common modelling tool. Although the neural networks replaced them as a primary function of HMM states representation, GMMs are still found useful as a preprocessing step before the neural net training. As mentioned in [29], GMMs help to force-align phonemes in the individual speech utterances before sending them into a neural network based model for training.

Gaussian mixture is a function consisting of several Gaussians, each identified as $k \in \{1, ..., K\}$ where K is a number of clusters in the data. Each Gaussian k is defined by its mean value μ, covariance matrix Σ and a mixing probability π specifying weight of each gaussian. The general Gaussian density function is shown in Equation 2.4 where $\boldsymbol{x}$ represents data points, D is the number of data dimensions and $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as specified above.

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}}{\sqrt{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|}} \tag{2.4}$$

To estimate the parameters of each Gaussian so that it represents the target data clusters, in our case phonemes, the expectation maximization algorithm is typically applied, introduced in [7]. Figure 2.3 demonstrates a simple example of three 1-dimensional Gaussians (the full lines) trying to approximate three data clusters (the dotted lines).

2.4 Feedforward neural networks

Current state-of-the-art approaches in HMM states representation use deep neural networks (DNN) [9]. An artificial neural network (NN) is a set of connected nodes called neurons which were loosely inspired by biological neurons in a human brain. Similarly to the biological neurons, the artificial ones send signals to the neighbouring neurons through their connections. Each connection is defined by a weight specifying the amount of the input signal being transferred to the connected neuron.

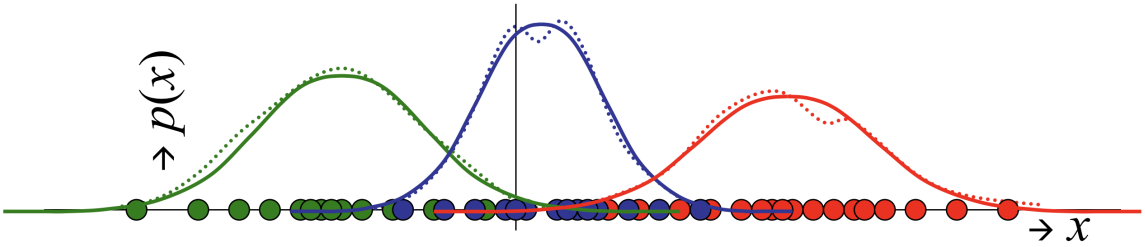

Figure 2.3: Example of three 1-dimensional Gaussians (the full lines) trying to approximate three data clusters (the dotted lines) (figure obtained from [18]).

The neuron function is to transform its multidimensional weighted input $\boldsymbol{x}^T\boldsymbol{w}$ to its single output by applying a non-linear function (e.g. sigmoid, Rectified Linear unit (ReLu) [25] or p-norms [43]). In feedforward neural networks the output of a neuron is input to neurons in the following neuron layer. A feedforward NN consists of an input layer, hidden layer(s) and an output layer. If an NN has multiple hidden layers it is called a deep neural network [16]. Deep neural networks in comparison to the shallow ones are able to learn representations of data with multiple levels of abstraction, which is very useful in complex tasks solving like image or speech recognition.

The output of a neural network is typically normalised to represent a valid probability distribution. The most commonly used normalization layer is a softmax function defined by Equation 2.5, taking a vector of K real numbers as input, and normalizing it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers [3].

A feedforward neural network with one hidden layer can be then defined by Equation 2.6, where $\boldsymbol{W}$ are weight matrices, $\boldsymbol{b}$ are bias vectors, $\boldsymbol{x}$ is an input feature vector and h is an activation function. A detail of a single neuron and an NN with two hidden layers are depicted in Figure 2.4. More information about neural networks structure and neural networks in general can be found in [2].

$$\text{softmax}(\boldsymbol{z}) = \frac{\exp z_i}{\sum_{j=1}^{K} \exp z_j} \tag{2.5}$$

$$y = \text{softmax}(\boldsymbol{W}_{(2)} h(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) \tag{2.6}$$

Gradient descent and backpropagation training

Values in $\boldsymbol{W}$ and $\boldsymbol{b}$ from Equation 2.6 are objects of an iterative training process. The goal of the training process is to estimate the parameters on a training dataset so that the NN is able to predict correct output values on unseen data.

The gradient descent algorithm serves to estimate the NN parameters [31]. To find a local minimum of a function, a step in a direction negative to a gradient of the function in a given point is taken. While training an NN the gradient descent method is applied in a form of the backpropagation algorithm [32]. The backpropagation algorithm is performed in two steps. The first step is a forward pass in which the input vector is propagated to the end of a neural network and a loss value of an objective function is computed. The error value is

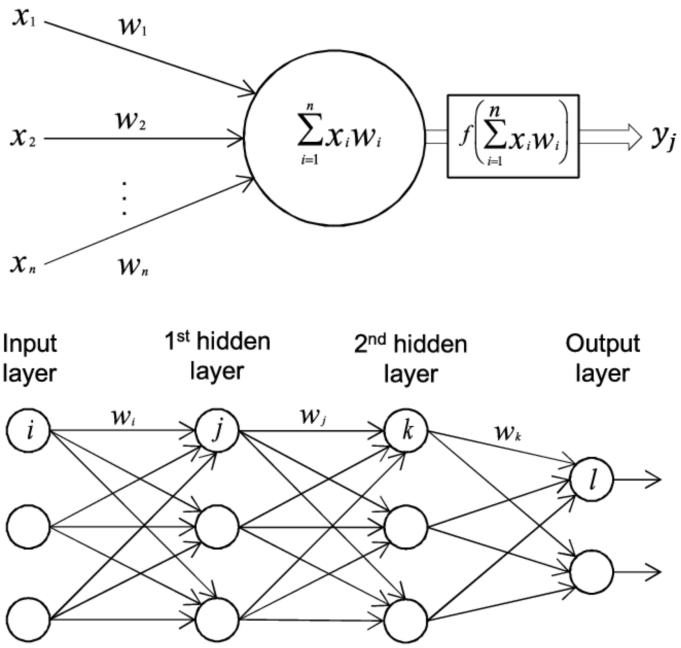

Figure 2.4: A detail of an artificial neuron function and a simple feedforward neural network with one input layer, two hidden layers and one output layer (figure obtained from [21]).

then propagated back using a chain rule of derivatives. Weights of the neural network are then updated according to Equation 2.7, where $\boldsymbol{w}^t$ are all neural net parameters in a step t, including weight matrices and biases, η is a learning rate and ∇E_n is a gradient of the objective function.

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \sum_{n=1}^{M} \nabla E_n(\boldsymbol{w}^t) \tag{2.7}$$

The accuracy of a DNN during the training is approximated by an objective function. A basic objective function commonly used for a 1-of-K classification problem is a multi-class cross-entropy (CE). However, later studies proved that replacing the cross-entropy objective function with sequence-discriminative criteria (e.g. maximum mutual information (MMI)) improves the overall results in the ASR task as it is a sequence classification problem [37]. Further on, usage of a so called lattice-free version of the MMI objective function (LF-MMI) decreases the computational costs by omitting the need of pre-computation of the lattices for all possible word sequences (the reference word sequence lattice remains) and avoiding the initial training with a CE model to create a precise weights initialisation [29].

Convolutional neural networks

A convolutional neural network (CNN) is a neural network with one or more convolutional layers. CNNs were firstly introduced in image processing [17] but due to their ability to break down complex spatial information abstractions, they were transferred to other areas like natural language processing and automatic speech recognition [1]. A convolution is sliding a filter over an input while creating a new output.

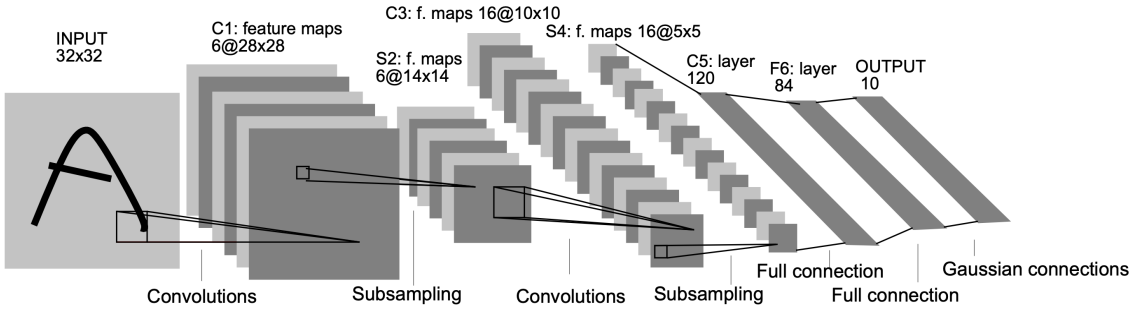

Figure 2.5: A convolutional neural network as presented in the original paper [17] can recognise spatial relationships by performing convolutions over its input (e.g. over an image).

Each convolutional layer contains series of filters known as convolutional kernels. A filter is a matrix of values that are used to multiply a subset of the input values. The sum of the multiples is as a single value transferred to the input of another layer. The values of the filters are objects of the CNN backpropagation training as described in Section 2.4, so that each convolutional layer and its kernels learn to find specific important information in the input data. A simple convolutional neural net is depicted in Figure 2.5.

Time-delay neural networks

Dynamic nature of speech causes many difficulties to automatic speech recognition systems. Specific speech features may be an important clue in recognising certain aspects of speech but they often appear not at the exactly same position and time in a phoneme sequence. Thus ASR systems have to be able to represent temporal relationships between acoustic events, while at the same time providing for invariance under translation in time. The time-delay neural network (TDNN) architecture successfully solves this problem [39]. To recognise a single phoneme, a window of both past and future feature frames is taken as input into a multi-layer neural network.

Additionally, as shown in Figure 2.6, later studies showed that not all connections between all frames are necessary, which significantly improves performance and lowers the amount of computations needed while training the network [26]. A factored form of TDNN (TDNNf) which is structurally the same as a TDNN whose layers have been compressed via SVD and is trained from a random start with one of the two factors of each matrix constrained to be semi-orthogonal also proved to lower the WER score of a system integrating it [27].

2.5 Language modelling

The aim of a language model (LM) is to estimate word sequences apriori probabilities in a given language (e.g. English). An LM models the $P(W)$ part of the Bayes rule from Equation 2.1. In practise, given a sequence of words $w_1, w_2, ..., w_t$, an LM computes a probability distribution of the next word w_{t+1}, while choosing a word from a pre-defined set of words in a lexicon.

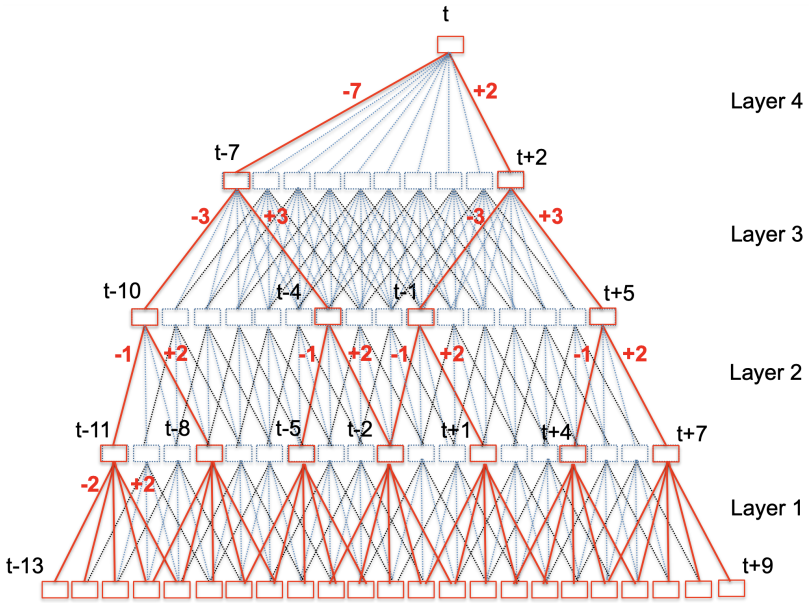

Figure 2.6: Pruned time-delay neural network as presented in [26] computes with subsampling (red) instead of using all connections (blue+red).

n-gram language models

Multiple approaches in language modelling exist (statistical LMs, neural network based LMs). A commonly used statistical language model is n-gram language model. Its main goal is to identify occurrence counts of sets of n, $n-1$, ..., 1 words in a large text corpus. The n-gram model is then used to predict n-th word given an $n-1$ long word sequence, see Equation 2.8.

$$P(w_{t+1}|w_t, w_{t-1}, ..., w_{t-n+2}) = \frac{count(w_{t+1}, w_t, ..., w_{t-n+2})}{count(w_t, ..., w_{t-n+2})} \tag{2.8}$$

If a specified n-gram isn't found in the training text corpus, a lower n is used until iteratively n equals one. This approach is called backoff. However, if a word wasn't seen in a training text corpus, the associated n-gram gets a zero probability. That's why techniques like Kneser-Ney smoothing [14] were introduced to improve the original n-gram algorithm. The Kneser-Ney smoothing adds a small non-zero probabilities to all words from the lexicon even if not present in the training text corpus which helps to overcome the mentioned problem.

Another n-gram LM disadvantage is the necessity of storing all corpus n-grams in the memory. Also, n-grams with similar meaning may be assigned different probabilities depending only on their usage frequency in the training text corpus, while their similarity is not taken into account at all. This results in a need of huge training text corpuses containing all possible word combinations. Additionally, a word context should play an important role during the next word prediction, which is strictly limited while using n-grams as higher n brings new complications.

Recurrent neural network language models

Neural network based language models, specifically recurrent neural network (RNN) LMs, try to solve these problems by not counting on exact word orders as n-grams do. Instead, the RNNs create their own internal representations of the input word sequences by taking theoretically infinite word sequences as input which is done by recurrent neural blocks shown in Figure 2.7 [19]. Also, the words are not represented by their alphabetical form. Instead, every word is assigned a high dimensional vector embedding (e.g. word2vec [20]) which the RNN takes as input. The main idea of word embeddings is assignment of close vectors to words with a similar meaning. The vector embeddings can be easily used in mathematical equations without loosing information about their true meaning.

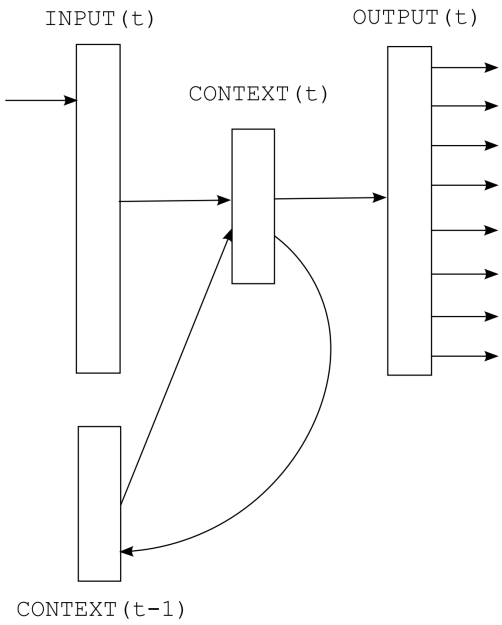

Figure 2.7: A simple recurrent neural network with one recurrent block taking its output as input in the next time step (figure obtained from [19]).

A recurrent block is a simple neuron which, besides other weighted values, takes its own output from a previous time step as input. This allows the RNN block to be influenced by an infinite number of the previously generated words. However, the word *infinite* is really true only theoretically. In reality, during the training of the RNN architectures different problems occur preventing the RNN blocks to be influenced by all of the past outputs. Long short-term memory (LSTM) blocks try to solve the problems of vanilla RNNs by introducing more complicated structures to preserve as much information from the past states as possible [10].

Interpolation of different domain language models

Having multiple different-domain text corpuses mixed together only to increase the corpus size may result in worse prediction abilities of an LM trained on such data. Instead, similarly to humans, assigning different probabilities to the same words depending on a domain and situation is prosperous to ASR systems. Building separate domain-specific language models and applying an interpolation technique to combine them into a single

language model optimised for the target use case showed better performance [30]. The simplest interpolation technique is a linear interpolation [12] expressed by Equation 2.9. The probability of a word w given its history h, while strongly assuming that $p(h|i)$ is independent of a domain i, where λ_i is a learned parameter - a weight of the domain i.

$$p(w|h) = \sum_i \lambda_i p_i(w|h) \tag{2.9}$$

Different improvements of the original algorithm were developed. One of them is a normalised log-linear interpolation of backoff language models introduced in [13]. A log-linear interpolated model uses the weights λ_i as powers. To form a probability distribution, the product is normalised with a normalization factor Z, where x are all words in the combined vocabulary, see equations 2.11 and 2.10.

$$p(w|h) = \frac{\prod_i p_i(w|h)^{\lambda_i}}{Z(h)} \tag{2.10}$$

$$Z(h) = \sum_x \prod_i p_i(x|h)^{\lambda_i} \tag{2.11}$$

Inter-word silence probabilities

As shown in [5], modelling not only probabilities of words following each other but also the inter-word silence pronunciations may improve the final predictions of an ASR system. As an example, zero-silence is less likely to follow the word White in "Gandalf the White said" than in "The White House said". The mentioned work proves that although the final improvement isn't huge, it's consistently beneficial on multiple different datasets.

2.6 Decoding with finite state transducers

As a final step in an automatic speech recognition pipeline, all the above mentioned components are combined in a decoding network represented by a finite state transducer (FST). A finite state transducer is a finite state automaton which besides accepting the input also produces an output. An FST consists of a finite number of states which are linked by transitions labeled with an input/output pair. The FST starts out in a designated start state and jumps to different states depending on the input, while producing output according to its transition table. Besides an input/output pair each transition is assigned a weight (e.g. probability).

Considering hybrid ASR system, a final FST is a composition of four transducers [22]. Transducer H from Equation 2.12 represents an HMM topology as described in Chapter 2.2. C as a context dependent transducer accepts context dependent phonemes, i.e. phoneme pronunciation depends on neighbouring phonemes (e.g. triphone is a context dependent phoneme). The pronunciation lexicon L assigns specific words to phoneme sequences according to a pre-defined list of word/pronunciation pairs. Finally, grammar G, i.e. language model, estimates probabilities of recognised word sequences.

$$HCLG = H \circ C \circ L \circ G \tag{2.12}$$

Word lattice as output of decoding

Possible word sequences are an output of the decoding network. Several hypotheses compactly stored in a word lattice format are assigned to a set of input frames. A word lattice is a weighted automaton representation of alternative word sequences together with probabilities of the individual words. A simple example of a word lattice is depicted in Figure 2.8.

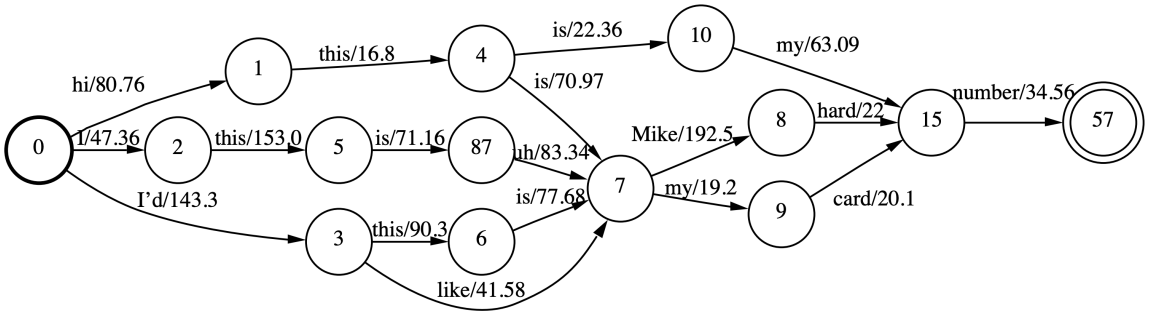

Figure 2.8: A word lattice representing alternative hypotheses of word sequences together with the weights of the individual transitions (figure obtained from [21]).

Recurrent neural network lattice rescoring

Although RNN language models (RNNLM) show better results in comparison to the n-gram models, because of the RNNLM theoretically infinite history input lengths it is technically impossible to compile the model into a static decoding graph. Thus RNNLMs are usually not directly used in the decoding. Instead, lattice rescoring is a common approach to take the advantage of recurrent neural language models in the decoding process [41]. After a word lattice is generated from the 1st-pass decoding, its probabilities are then rescored with an RNNLM.

2.7 Kaldi - speech recognition toolkit

Kaldi[1], introduced in [28], is a free open-source toolkit for speech recognition researchers. It consists of many tools and utilities for different parts of an ASR pipeline from data processing and feature extraction, through GMM and DNN based HMM training and speaker adaptation techniques, to finite state transducers, decoding graphs and lattice rescoring algorithms implementations. Kaldi is a C++ library providing binaries implementing the specific speech operations. Besides the speech processing components themselves, Kaldi also demonstrates how to integrate the binaries into a usable ASR system. It is via a set of python and shell scripts showing how to use the Kaldi binaries in meaningful ways. Kaldi contains many of such recipes obtaining state-of-the-art results for different widely known datasets. The recipes are usually a good starting point when working on your own speech recognition problem.

The most interesting parts of Kaldi are the *src* and *egs* directories. *src* contains the core C++ functionality of Kaldi while *egs* contains all of the recipes built for different ASR datasets. A typical recipe consists of a *run.sh* script which calls all the necessary programs

[1] https://kaldi-asr.org/

from the *steps* (mostly scripts calling the C++ binaries) and *utils* (mostly data preparation scripts) directories. Also, a data directory needs to be in a Kaldi defined format. A typical data directory is a set of csv files mapping utterance texts, ids, audio segments, speakers and features between each other. More detailed information can be found in the Kaldi documentation[2].

The most similar toolkit to Kaldi is probably the HTK[3] toolkit. The HTK toolkit also builds on top of HMM based ASR systems and implements tools necessary to create a speech recognition system. However, I chose to use Kaldi for several reasons. Kaldi's supportive community is a huge advantage of Kaldi. The Kaldi project is much more alive than the HTK project and even a new version of Kaldi is currently being in progress which is important, especially nowadays when the research moves forward so fast. The huge number of prearranged recipes and tools in Kaldi are also very helpful and could be difficult to reproduce in HTK in some cases.

Forced alignment tools based on Kaldi

To align manually created transcriptions to its audio, a simple ASR with a language model biased to the reference transcription itself precisely can decode the audio. The decoded transcription contains timestamps which are the result of the force alignment.

A sequence of two tools built on top of the Kaldi toolkit performed the forced alignment in this work. First, the Gentle[4] tool trains a biased LM on the input transcriptions. Then, using a Kaldi acoustic model, the input audio is decoded. After decoding is finished, the Gentle output alignments are recursively realigned using a Canetis[5] tool. Forced alignment implemented in Canetis based on [23] works as follows: After the alignment is applied on the whole input text, the text is split into successfully aligned parts and unaligned parts. The parts of the transcript alignment that are most confidently correct are anchors. The anchors divide the utterance into unaligned and aligned parts. Iterative applying of the alignment on the individual unaligned parts results in a very precise overall alignment. The alignment process in depicted in Figure 2.9.

[2] https://kaldi-asr.org/doc/
[3] http://htk.eng.cam.ac.uk/
[4] https://lowerquality.com/gentle/
[5] https://github.com/nsheth12/canetis

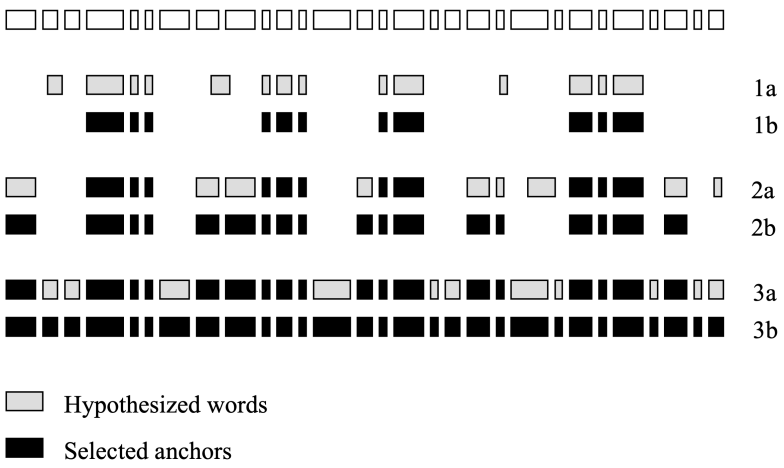

Figure 2.9: Recursive utterance alignment process - an alignment technique is iteratively applied on incorrectly aligned parts until an ending condition (figure obtained from [23]).

Chapter 3

Dataset creation

As the proposed ASR system was meant to outperform the existing general domain ASR systems on our target domain, the data used to train it had to perfectly represent the target domain and the accuracy of the data annotation had to meet high expectations. This chapter describes in detail our data scenarios and the process of the data annotation.

3.1 Data scenarios description

Data obtained from our customers may be divided into the following categories:

- Depositions. Relatively clear audio with three to five persons in a quiet room, usually sitting around a table with a microphone device on top of it. The conversation is formal and partly managed by a moderator (a court reporter). There is less crosstalks (speakers speaking over each other). If a part of speech is inaudible the speakers are asked to speak more clearly.
- Witness statements. This scenario covers various kinds of situations thus I find it best to divide them into three subcategories:
  - Police station statements. This setup is very similar to the depositions setup. The difference is often a more loosely managed debate as the persons present are often criminals arrested by police. The microphone device may be of a lower quality and with a worse placement (usually somewhere in the top of the room).
  - Outdoor witness statements. This is the most difficult setup as it can take place in any outdoor or indoor environment. Often very noisy audio with street or wind background noise. The microphone is usually placed on a body of a police officer.
  - Driving under influence (DUI). A specific outdoor scenario where a police officer inspects a driver for alcohol influence or intoxication. A similar scenario to the previous one except the conversation content is highly predictable.
- Phone calls. As the scenario name suggests, the subjects of this scenario are phone calls between two persons. It may be a phone call from a prison, a 911 call or any other kind of a phone call which is used in a particular legal case.
- Court room testimony. A challenging scenario with many speakers often far from a microphone. However, there are not many of these recording types in our dataset.

The scenarios described are a difficult task for a single ASR system and the best solution would be to build a separate ASR system for each of them. Although this is true, the most frequent recording types received from our customers are the depositions and good quality witness statements. For now, we wouldn't be able to collect enough data for each of these use cases separately. This problem will be partly handled with data augmentation described later in the text.

Another bright side of our use case is that many speakers (the customers themselves or persons from the customers' work surroundings) are often repeatedly present in the incoming audio. Although this will help our production systems, this work doesn't use this information and the train and test sets contain unique speakers as mentioned later in the text.

3.2 Data annotation process

The goal in this project was to annotate 300 hours of audio conversations. Because of product reasons our data annotation pipeline is divided into two steps:

1. Audio transcription.
2. Transcription cleaning and audio alignment.

Audio transcription

Immediately after obtaining a recording from a customer, the transcription team makes a verbatim transcription of the conversations in the obtained audio recordings. A verbatim transcription means all speech parts, every significant noise (e.g. knocking on door) and every spoken noise or filler words like „Uhm“ and „Um-mm“ are transcribed. The transcriptions often contain meta-information e.g. „Nodes her head affirmatively“ (in case of video files). These transcriptions are meant primarily for a customers usage and not to be a part of a dataset. This is corrected in the second annotation step. In the first transcription step the recordings are also segmented into parts spoken by different speakers. The created segments are assigned start and end timestamps using an automatic forced alignment tool.

Transcription cleaning and audio alignment

As ASR models are typically trained on short speech utterances ideally 10-30 seconds long, too long one speaker segments are not suitable for the ASR training. Thus the transcriptions from the previous transcription step need to be cut into shorter utterances every of which has to carry a timestamp information specifying where the utterance starts and ends in the recording audio.

Although the automatic alignments showed to be good enough for the customer facing application, their accuracy is not perfect as many utterance alignments failed with several seconds inaccuracies (mainly because of crosstalks, unintelligible speech and long silence periods). That's why I had to solve the problem by manually adjusting the automatically generated timestamps. A part of this annotation step was also adjustment and normalization of the transcription texts as they sometimes contained undesired meta-information. This was also a good review of the transcription quality from the first annotation step.

Failed annotation approaches

As data annotation is a highly time and money consuming process, it is important to choose the right approach so that the resources are not wasted. The annotation procedure should match the three main criteria which are time, money and accuracy. A several hundred hours big dataset annotation may take months of work and thousands of dollars if managed wrongly. Thus it is very important to choose an appropriate annotation software and a group of the right responsible people to finish the task successfully.

My first attempt of this data annotation project involved third party annotation applications (SpokenData.com[1], TranscriberAG[2]) and a group of students which is a common way of small annotation projects (e.g. [36]). Students would fix the automatically generated timestamps using an annotation tool. After finding out both applications weren't totally suitable for our use case due to missing functionalities, inefficiencies or implementation problems I decided to pause the in-house annotation project.

The second approach involved a cloud-sourcing annotation company. After talking to several candidates I chose FigureEight[3] as they're proven by working with big customers like Facebook, ebay or Oracle on various annotation tasks. They were opened to design a special tool for the timestamps adjustment task. We started a trial period during which they tried to prove to be able to achieve the required results. Because of the cloud-sourcing approach many cheap workers could work on the task which would save plenty of time. However, the cloud-sourcing approach also turned out to be a disadvantage of this approach as many workers cannot be managed properly which results in bad annotation accuracy. Even after several trial iterations the company wasn't able to deliver the desired annotation accuracy, i.e. the utterances often overlapped, words were cut out from the audio or the utterance audio contained speech not present in the utterance transcription. Because of both high cost and inaccurate annotations, which I think were caused by inefficiency of their annotation tool and too many unmanaged workers, I decided to stop the collaboration.

Final annotation approach

With the newly gained knowledge of annotation processes I decided to return to the in-house annotation project. I designed a new annotation application with all of the needed functionalities, see Figure 3.1. The application offered easy-to-use user interface with properly chosen keyboard shortcuts for the most common actions. A built-in review system helped the new annotators to learn the task quickly. The application, given my design, was implemented by our company front-end engineers so I won't go into detail as it is not my work. The tested annotation speed and efficiency showed to be the best from the so far tested software. The annotation in the proposed tool takes between two to four times the original audio length depending on audio quality.

As a workforce a group of 15 students was chosen with agreement of 15 hours of work per week. During the first weeks of the annotation project I reviewed the work of the annotators. Using a built-in application system of utterance commentaries I taught the workers what results and accuracies were expected of them. After several weeks I chose the best annotator to be a reviewer instead of me.

This approach also turned out to have a problem. It was the workers time - the students couldn't be made to spend the desired amount of time on this job due to their school

[1] https://www.spokendata.com/
[2] http://transag.sourceforge.net/
[3] www.figure-eight.com/

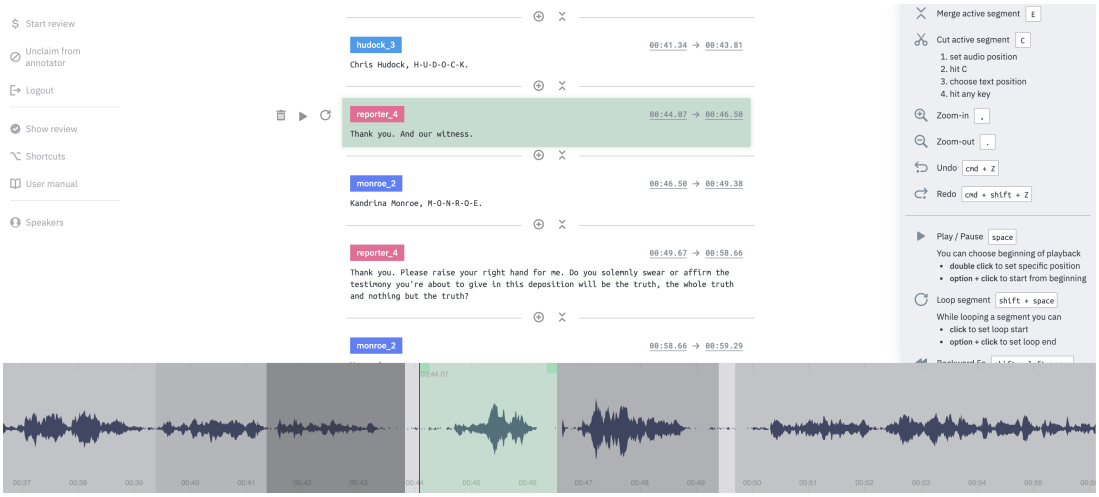

Figure 3.1: A proprietary highly efficient audio annotation tool designed by myself and implemented by our team of engineers to meet our specific dataset preparation needs.

responsibilities. Having more students in the team would make the project unbearable for me from the managing point of view thus the workforce was enriched by another annotation company - CloudFactory[4].

The difference between cloud-sourcing companies like FigureEight and the CloudFactory annotation company lies in the size and management of their teams. While the cloud-sourcing companies tend to have big teams with just a small focus on their management, CloudFactory provides small dedicated and trained teams which behave like a part of the client's own team.

While letting our old annotators and reviewers in the pipeline and joining the force with the CloudFactory full-time dedicated workers, the plan was to deliver 300 hours of annotated data in two and a half month. However, the project finished with 110 annotated audio hours. The payment system of the CloudFactory workers (dollars for a human hour) caused the workers to be too slow. The most efficient annotation approach turned out to be usage of a proprietary annotation tool and a team of in-house annotators payed by a finished audio hour and controlled by skilled reviewers. This made the annotators both fast and accurate.

3.3 Freely available legal domain dataset

Besides our own transcription data, an additional freely available dataset was used. The Supreme Court Oral Arguments[5] dataset consists of roughly 7,000 hours of transcribed audio of different court hearings recorded during the last several decades in the US Supreme Court. Except different room impulse responses of the big court rooms, the scenario is supposed to be very similar to deposition and partly witness statement setups mentioned in 3.1 which is formal conversation speech in a legal domain.

Usage of data augmentation corrected the room impulse response differences to better match our target domain. Data from [15] were used for augmenting the Supreme Court

[4] https://www.cloudfactory.com/
[5] https://www.supremecourt.gov/oral_arguments

dataset. The augmentation dataset was composed of real room impulse responses (RIRs), simulated RIRs and various manually recorded noises.

Due to the lattice-free MMI objective function sensitivity to incorrect transcripts [29] [6], the Supreme Court acoustic data were cleaned from the utterances with supposedly incorrect transcriptions. The cleaning was performed using a *Kaldi* script[6]. The script removed utterances whose, after decoding with a biased LM, lattice oracle path was still far from the reference transcript. This way, 12% of the acoustic data was considered to be incorrect and thus removed from the dataset. The removed utterances often contained crosstalks or the audio was cut in the middle of the utterance text.

3.4 Text corpuses for language model training

To train the language models (both n-gram and RNNLM), different text corpuses were experimented with. Firstly, the company transcriptions were used as one text corpus. Secondly, the whole Supreme Court hearings dataset transcriptions created the second text corpus (4M sentences). The rest consisted of a general language text corpus (a 42M sentence subset of Google Billion Word Benchmark [4]), a set of legal deposition transcriptions collected on the Internet (3M of conversational data) and a set of publicly available court decisions from The Caselaw Access Project ("CAP")[7] (20M sentences of non-conversational data), a part of which is free to download, specifically court hearings from the states Arkansas, Illinois, New Mexico a North Carolina.

[6] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/cleanup/clean_and_segment_data_nnet3.sh

[7] https://case.law/about

Chapter 4

Automatic speech recognition system architecture proposal

This chapter describes all components assembled together to create the proposed automatic speech recognition system. The whole architecture pipeline (Figure 4.1) is divided into four main steps, each of them described in the chapter sections in detail:

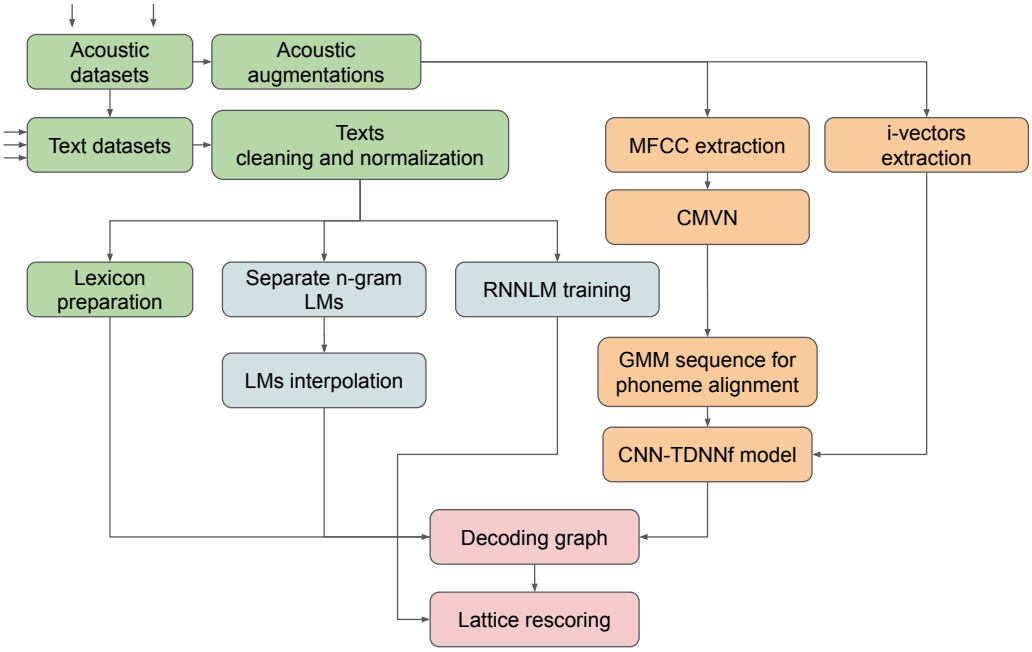

Figure 4.1: The overall architecture of the proposed ASR system from data processing to individual models training is dividable into four main steps: data processing, language models training, acoustic model training and decoding.

- Data processing - the text transcriptions from all sources are normalised to be in a common format suitable for speech recognition training. All of the data are split into unique train and test sets. The train set is augmented using room impulse responses and speed perturbation.

- Language modelling - all of the text data are fed into separate n-gram language models training which are thereafter interpolated to compose a single language model. A recurrent neural network language model for lattice rescoring builds on the top of the interpolated n-gram model when applied in the lattice rescoring process.
- Acoustic modelling - gaussian mixture and neural network based HMM acoustic models are trained on the augmented audio data.
- Decoding - the trained language and acoustic models together compose a decoding network being able to transcribe the audio conversations into a text format. After a lattice is generated from the acoustic model decoding, it is rescored with an RNNLM to improve the prediction quality.

4.1 Data processing

Both the text transcriptions and audio data have to meet certain requirements before being fed into the training. The lexicon should include every valid English word from the transcription texts and as few out of vocabulary words (OOV) as possible should exist in the dataset. Also, transcriptions should be normalized, meaning processing of punctuation, numbers and letter cases. See an example of a text before (a) and after (b) text normalization.

```
   (a) I'm Bond, James Bond.  It's nice to meet you!  My ID is 007.
1964, what a year...
   (b) I'm Bond James Bond it's nice to meet you my I D is zero zero
seven nineteen sixty four what a year
```

The dataset audio has to be prepared too. All audio files are converted to a common audio format with the desired specifications. The audio is then augmented to increase the dataset size and to enhance robustness of the final model. The whole data processing pipeline is depicted in Figure 4.2.

Text normalization

As only the company transcriptions were manually normalized and transformed to the fully verbatim form, automatic normalization had to take place for all the other texts available. The annotated datasets often contained more texts than just the plain letter sequences which are typically the output of an HMM based speech recognition system. The transcription text should consist only of words and symbols present in the lexicon. In this work that means only alphabetical characters and the apostrophe character to represent the marking of the omission of one or more letters (as in the contraction of do not to don't) or of possessive case of nouns (as in the eagle's feathers). Thus the rich text produced by the annotators containing punctuation, theoretically infinite number of digit character combinations and inconsistent letter cases had to undergo automatic normalization process.

The normalization script was prepared by iterative rules updates and testing on a subset of the text data which is demonstrated by Figure 4.3. First, a simple test set composed of texts before and after normalization containing several examples was prepared. Then the normalization script normalized a subset of the whole text corpus. The output of the normalization round was examined manually. If incorrectly normalized texts were found,

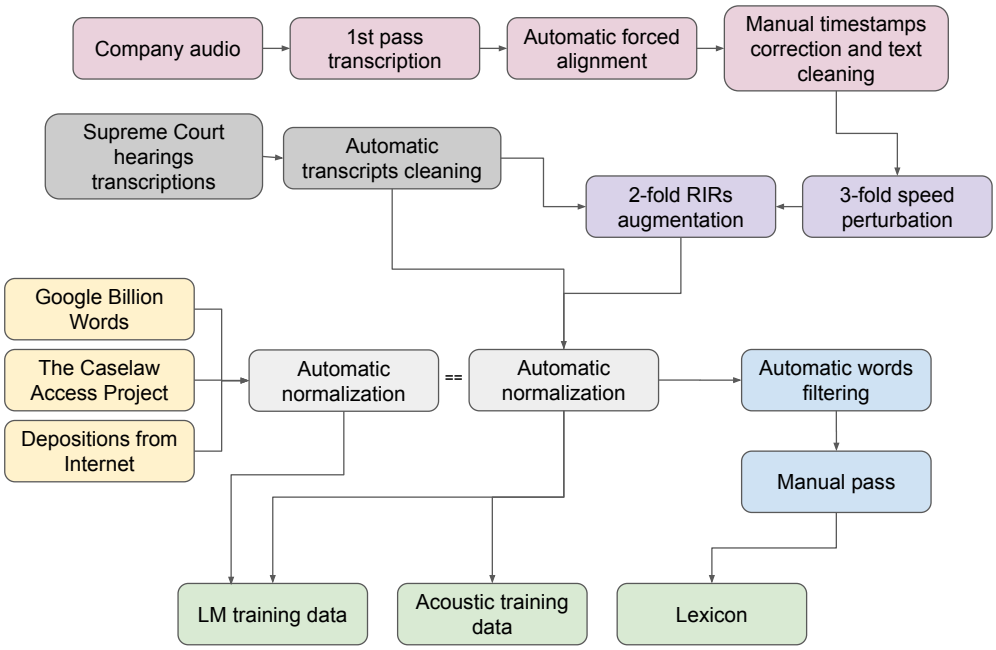

Figure 4.2: Data processing pipeline preparing texts for language models training, transcriptions and audio data for the acoustic model training and a set of recognisable words for the lexicon.

new rules fixing them were implemented in the normalization script, the problematic text sequences were added to the test set and the updated normalization script was tested on the test set to find out whether the new rules didn't broke any of the old test examples. This procedure of normalizing a new subset of text corpus, manual examination of the output, updating the rules and the test set according to the errors found and running the updated normalization on the test set was repeated several times until the manual checks stopped discovering incorrectly processed text sequences. The final normalization script was run on all the remaining texts.

The normalization script consisted of these main procedures:

- **Letter case sensitivity** - the letter cases are in some ASR systems not taken into account. As the Parrot product efficient usage requires correct letter case sensitivity, I tried to keep this information from the original data. However, I decided not to keep all of the original letter cases. Only the proper nouns remained capitalized. The rest of the upper case words (e.g. sentence beginnings) had to be replaced to the lowercase form. Counting on a heuristic saying which upper case letter should be switched to its lower case version, I adjusted the input text letter cases using the following process. The words not at the sentence beginnings starting with an upper letter kept their form as they were considered to be transcribed correctly by a human annotator. Words at the sentence beginnings underwent an additional control. A trained named entity recognition model from a *Spacy* [1] python package estimated a role of the word at a sentence beginning. If the word represented a certain entity, the

[1] https://spacy.io/

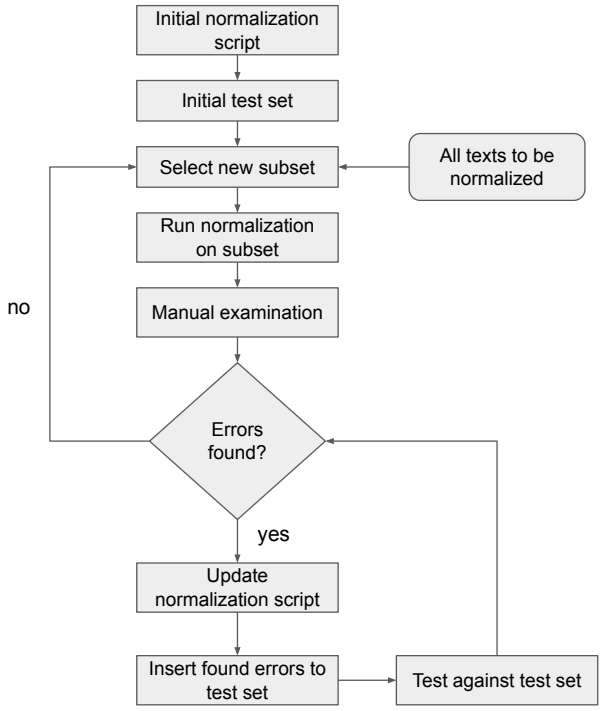

Figure 4.3: The normalization script was prepared by iterative rules updates and testing on a subset of the text data. Each iteration of the subset normalization was manually examined and if errors were introduced, the normalization rules were updated accordingly. To not break the existing rules by creating the new ones, a test set was maintained. After enough iterations (when I considered the normalization script to be robust enough), the normalization was run on all remaining data.

first letter kept its upper form. Otherwise it was transformed to a lower case. There were several other specific use cases in the transcriptions. For example abbreviations with fullstops after letters couldn't be considered to be ends of sentences (e.g. Mr. Underhill or J. F. Kennedy) and had to be processed differently.

- **Digits to words conversion** - adding a theoretically infinite number of numbers in their digit form into a lexicon is obviously impossible. Instead of inserting all of them, every number expressed by digits was converted to its appropriate alphabetical form (e.g. 5,010 to *five thousand ten*). As an example, to express all numbers between 0 and 5,000 with this enhancement, only a bunch of words is needed instead of all 5,000 words representing all numbers from this range. A python package *num2words*[2] performed the digits to words conversion in the basic use cases. Several exceptions had to be handled such as .5 processed as *point five* as these were not processed by the mentioned package.

- **Special characters processing** - special characters like dollars and percents were replaced with their alphabetical equivalents in an appropriate way (*$100,000* as *one hundred thousand dollars*, *Marks & Spencer* as *Marks and Spencer*, *'70s* as *seventies*, *john@somewhere.com* as *john at somewhere dot com* etc.).

[2] https://github.com/savoirfairelinux/num2words

- **Abbreviations spelling** - regular expression rules took care of detecting abbreviations in the text (e.g. FBI, CIA). A space was put between the individual letters of the abbreviations so that the model would be able to learn the representation of the single spelled letters better. Learning this helped the model with both name spelling, which appears in almost all client recordings, and transcribing unknown abbreviations not present in the lexicon as the alphabet to be spelled is a finite set of letters unlike all possible letter combinations forming specific abbreviations.

- **Manual normalization** - in some cases the automatic normalization wasn't able to process the text into the correct form in which the text was actually pronounced. Such cases had to be at least partially processed manually to. Years pronunciation was an often example. If a year 1999 was pronounced as nineteen ninety-nine, the human annotator had to correct the text to 19 99 (a space inserted) so that the automatic normalization could process the number in the correct way that matches the real pronunciation. This was similar with phone numbers etc. The manual normalization step was performed only on the company transcriptions. In the Supreme Court transcriptions the script automatically normalized all numbers between 1900 and 2000 as *nineteen something* as this was the most usual pronunciation in the dataset and it would done less harm as normalizing these numbers to *one thousand nine hundred something*.

- **Additional enhancements** - as minor improvements, some specific use cases were also handled automatically. E.g. words like *hm-mm* and *uh-uh* were allowed to keep the dash symbol, while in other use cases like words spelling (e.g. *H-A-L-L-O*) the dash symbols were replaced with a space.

Pronunciation lexicon creation

A custom case sensitive lexicon, a list of recognisable words with their pronunciations, was prepared for the purpose of this project. A multistage process filtered all words in the company and Supreme Court transcriptions. To let only existing English words remain in the lexicon and to exclude typos from the lexicon a word cleaning step took place. After counting occurrences of all words in the training text corpus, only those with an occurrence count higher than one were kept for the lexicon. Words seen only once throughout the whole text corpus were checked by a more advanced filter. The Google translate service[3] played a role in the advanced filter. If the translation service was able to translate a word found in the corpus to Spanish, i.e. the output contained a different text than the English text provided as input, the word was considered to be a valid English word. Otherwise, the word was searched in a names database of common names from various countries[4] and a brands database of more than 7,000 known brands[5]. If not found, it was definitely excluded from the lexicon.

In addition to automatic words cleaning, I examined the whole lexicon manually and discovered the obviously incorrect words (mostly words with misused single quotes and misspelled words, e.g. dont, i'm, reuslt etc.) which resulted in about 400 removed words. The removed words which could be certainly replaced by the corrected versions of the words (e.g. don't instead of dont), were used to clean the transcription texts even more. Using

[3] https://translate.google.com/
[4] https://github.com/smashew/NameDatabases
[5] https://github.com/MatthiasWinkelmann/english-words-names-brands-places

a dictionary with incorrect and correct versions of the removed words, the incorrect words were found in the text corpus and their corrected version from the dictionary replaced the incorrect word in the corpus.

The final lexicon resulted in 114,890 words. The lexicon words were assigned pronunciations using a grapheme to phoneme model[6] pre-trained on the CMU dictionary[7]. If the package didn't find a word in the CMU dictionary it performed a prediction with the pre-trained model. However, although the CMU dictionary contains multiple pronunciations for some words, this package provided only a single pronunciation for a word. Thus the code of the package had to be adjusted to return all possible pronunciations for a specific word from the CMU dictionary.

Audio processing

Several audio preparation steps took place in order to prepare the audio data. Firstly, different audio formats had to be converted to the *wav* format. Thereafter applying the acoustic data augmentation to increase the dataset size and making it more suitable for our target domain. Finally, additional minor enhancements were experimented with to find out whether they improve the model accuracy.

- **Audio format** - the original data from customers come in different formats (mp3, ogg, mp4 and others). Thus as a first step all recordings are converted to wav files. The further specifications of the wav files are 16kHz frequency and 16 bit mono channel. For converting the files AWS Elastic Transcoder[8] was used as a part of our production pipeline (out of the ASR system itself).

- **Filter out audio with inaccurate transcriptions** - some utterance transcriptions may have an inaccurate transcriptions, especially the utterances from the Supreme Court dataset as their annotation wasn't managed in-house. Thus to prevent the model from learning incorrect phoneme representations, such utterances need to be filtered out. The cleaning was applied twice. First, before a proper DNN model was trained, a GMM model trained on not cleaned data performed the cleaning using an appropriate Kaldi script[9]. After a newly trained DNN model was available, the cleaning was performed for the second time on the same subset of the Supreme Court data as before, using a Kaldi script[10]. In both cases similar amount of transcriptions was removed (10% and 12% respectively) which proved the robustness of the cleaning algorithm and its independence on the acoustic model. The removed utterances often contained crosstalks or the audio was cut out before the text end of the utterance.

- **Augmentation** - to increase our own dataset size and for the Supreme Court data to better match the target domain, both datasets underwent an augmentation procedure during which room impulse responses (RIRs) from the augmentation dataset described in Subsection 3.3 were used to augment the data. As our own dataset size was not large at the moment of the training, speed perturbation was used to create different speed

[6] https://github.com/Kyubyong/g2p

[7] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[8] https://aws.amazon.com/elastictranscoder/

[9] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/cleanup/clean_and_segment_data.sh

[10] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/cleanup/clean_and_segment_data_nnet3.sh

versions of the original data. Kaldi scripts[11,12] performed the RIRs augmentation and the speed perturbation steps. The actual parameters specifying by how much was the dataset size increased were based on the original paper and also differ between different experiments thus the exact values are stated later in the experiments description section.

- **Silence frames concatenation** - the first states in a Kaldi HMM, and the last states respectively, represent silence (or non-speech in general). That means such an HMM expects every utterance to begin and end with frames not containing speech. Otherwise the first or end states would be forced to accept speech phonemes and the silence would not be modelled correctly which would end up by mixing silence and speech phonemes frames during the recognition. Although the annotators were instructed to add a silence period before and after every annotated speech utterance, in cases when the speakers spoke fast without a silence period between individual speech sequences, some utterances did not contain the beginning and end silence (around 50 percent of utterances in the Supreme Court dataset and 20 percent utterances in our dataset according to a Kaldi script `analyze_phone_length_stats.py`). To fix this, a set of different silence frame sequences (~30ms audio files) was manually collected from our dataset. The collected silence frame sequences were thereafter randomly assigned and concatenated with every speech utterance. This was meant to prevent the HMMs from using speech phonemes to model silence.

Dataset splitting into unique speaker subsets

To correctly measure accuracy of the system and to objectively compare the training results between individual experiments, the dataset had to be split into disjunct sets for training and evaluation - a train set and a test set. For the results to be as unbiased as possible, the speaker sets present in the individual data splits should be disjunct.

The Supreme Court dataset was split using a Kaldi script, `utils/subset_data_dir.sh`, which divided the data directory randomly by selecting certain number of utterances. This was possible as the speakers in the dataset were assigned unique IDs.

The dataset from our customers doesn't contain unique speaker IDs. A uniqueness approximation method had to come in place. The only information about the speakers we have is their name (e.g. John Smith), or a variant of it (e.g. Detective Smith, Mr. Smith). This information was set by the customers themselves or by our annotators during the transcription process. Before splitting the dataset, the speaker names were converted to more robust identifiers. The requirements of the newly generated identifiers were:

1. Every speaker ID has to be unique, so that any two utterances with the same speaker ID really have the same speaker.

2. Using the speaker IDs, it has to be possible to split the dataset into two sets such that no speaker from one set is present in the other.

To ensure the first requirement, every speaker name was concatenated with a unique recording name. Thus if there would be two different speakers with a same name in different

[11] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/data/reverberate_data_dir.py

[12] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/utils/data/perturb_data_dir_speed_3way.sh

recordings, they would remain unique due to the unique recording identifier. Uniqueness of speakers with same names in one recording was secured by adding an index to a recording speakers list to every speaker.

The second requirement was secured by the following approach: Vast majority of the speaker names ended with a surname of a person (John Smith, Detective Smith etc.). The information whether John Smith from one recording is or is not the Detective Smith from another recording wasn't available and from the product point of view, we couldn't ask the customers to provide better identification. Thus the unique speaker dataset split was done according to speaker surnames. The speakers with the same surname were clustered to a common speaker surname set. If the test set contained a set of speakers named Smith, the train set didn't contain any speaker named Smith.

Additionally, the ideal dataset split would consider also recording uniqueness between the individual subsets. This would mean that if a recording X contained speakers A and B and the speaker A would be assigned to the test set, also the speaker B would have to be assigned to the test set. Moreover, if another recording Y contained a speaker A, the speaker would also have to be assigned to the test set together with all the other speakers from the recording Y and so on.

In our dataset, in which the same speakers often repeat among the recordings, this would mean a huge chaining of speakers and a reasonably sized test set couldn't be created. Thus only the speaker uniqueness condition was fulfilled completely. The recording uniqueness was achieved only partially: Every speaker was assigned a value representing how many utterances would be chained to this speaker by fulfilling the recording uniqueness. When creating the test set, firstly the speakers with the least chained utterances were selected to be a part of this subset. This allowed at least a few recordings to be unique in the test set.

4.2 Language modelling

Separate language models were trained for decoding and lattice rescoring. Several versions of both the n-gram language models for the decoding graph and the RNNLMs for the lattice rescoring were tested and examined using different combinations of text corpuses described in Section 3.4

Training n-gram language models with KenLM

To train the n-gram language models an open-source tool KenLm[13] was used. KenLM binaries estimate language models from text using modified Kneser-Ney smoothing algorithm. This toolkit also provides other functionalities to manipulate with n-gram LMs such as different LM format conversions (text/arpa), log-linear LMs interpolation and model pruning, which is a technique to decrease the size of an LM by not saving the n-grams that appear less than a certain threshold in the training corpus.

A similar widely used toolkit for statistical models building is SRILM[14]. It also estimates n-gram LMs from text, provides pruning and interpolation techniques but integrates different algorithms to achieve the results. KenLM claims to be faster, more accurate and has lower memory requirements than SRILM as it performs all the computations on disk. I used KenLM in this work laso because of its license which is more friendly with commercial usage.

[13] https://kheafield.com/code/kenlm/
[14] http://www.speech.sri.com/projects/srilm/

KenLM binaries expect files with individual text utterances on separate lines as input. The *lmplz* binary takes the desired n-gram order, pruning settings and the text file on its input and prepares the resulting LM in the *arpa* format. Having multiple n-gram LMs prepared with the *lmplz* binary, the *interpolate* binary computes the final interpolated LM and assigns log-linear weights to the individual n-gram LMs taken on the input. The interpolation finetunes the LMs' weights given a tuning text corpus set different from the test set. The output of the interpolation is an LM in the *arpa* format, log-linear weights of the input LMs and the final perplexity on the tuning set. The tuning set on which the interpolation was performed was a part of the train set not used during the separate LMs training. The results and experiments with different pruning setups and data combinations are presented in the following chapter.

Recurrent neural network language model for lattice rescoring

The recurrent neural network language model (RNNLM) was built on the letter based features modelling presented in [42]. The training based on the Kaldi Switchboard recipe[15] implementation of the above mentioned algorithm was run on several combinations of the prepared text corpuses. The specifics of different versions of RNNLMs trained are stated later in the following chapter.

4.3 Acoustic modelling

Before the training of the final DNN acoustic model, several preparation steps had to take place. First a GMM model had to be trained to provide phoneme alignments of all speech utterances as the input for the DNN, i.e. the DNN model had to be aware of the position of individual phonemes in the utterance audio. After high dimensional features were generated from the audio and i-vector speaker adaptations were extracted, the DNN training itself could begin. Most of the training pipelines and the DNN architecture was based on a Kaldi recipe for the Switchboard database[16]. Unless otherwise specified, all of the following algorithms were performed using Kaldi scripts and utilities.

Feature extraction

Both low (13) and high (40) dimensional MFCC features were extracted[17]. Low dimensional features were taken as an input for the GMM training while the high dimensional features were used by the DNN architecture. After extracting the features, cepstral mean and variance statistics (CMVN) per speaker were computed[18]. CMVN minimises the distortion by noise contamination for robust feature extraction by linearly transforming the cepstral coefficients to have the same segmental statistics.

Additionally to the MFCC features, the TDNN architecture takes 100 dimensional i-vectors as input. The i-vector extractor was prepared with a script based on the kaldi

[15] https://github.com/kaldi-asr/kaldi/blob/master/egs/swbd/s5c/local/rnnlm/tuning/run_tdnn_lstm_swbd.sh

[16] https://github.com/kaldi-asr/kaldi/tree/master/egs/swbd

[17] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/make_mfcc.sh

[18] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/compute_cmvn_stats.sh

Switchboard recipe meant for this purpose[19]. The extractor was thereafter used to extract i-vectors from the training and testing subsets.

Gaussian mixture models for phoneme alignment

For a TDNN architecture it is important to know where the individual phonemes of a speech utterance start and end in the utterance audio. Inspired by the Switchboard recipe[20], a sequence of GMM models was trained for this purpose, each model aligning its input with the alignments generated with the previously trained model.

Starting with the simplest monophone training[21] on a subset of the 10k shortest utterances, an initial HMM model was prepared. A monophone model is an acoustic model that does not include any contextual information about the preceding or following phone and is used as a building block for triphone models, which do make use of contextual information. The small subset of the shortest utterances ensured to minimize the error while aligning the initially not aligned data. Before selecting the shortest utterances, duplicate utterances were removed resulting with at most 80 duplicates left (too many utterances with text e.g. *„Okay.“* are not desired in the training set).

The monophone model aligned a larger dataset subset, specifically 30k utterances. Delta-delta features based model[22] took the alignments and was trained twice, while the first version aligned the utterances for the second version of the same model architecture, except its parameters representing the model size (number of leaves, number of gaussians) were set to higher values for the second model.

Following levels of the sequentially trained GMM based models performed Linear Discriminant Analysis and Maximum Likelihood Linear Transform[23] (1 model), and Speaker Adaptive Training [24] (2 models). Additionally, a Speaker Adaptive Training model with silence pronunciation modelling[25] was trained on top of aligments generated from the previous models.

Time-delay neural network model

The neural architecture used in this work is a Kaldi nnet3 setup with the new chain improvements[26]. The chain models bring lower computation resource requirements and introduce usage of a sequence-level objective function throughout the whole training. The convolutional time-delay neural network presented in the above mentioned setup consists of different layer types. The network starts with 6 convolutional layers with batch normalizations and a ReLu activation function. The CNN layers are followed by 9 TDNN layers. After decoding the word lattices are rescored with the previously trained recurrent neural network language model. Experiments with different architecture sizes are presented in the following chapter.

[19] https://github.com/kaldi-asr/kaldi/blob/master/egs/swbd/s5c/local/nnet3/run_ivector_common.sh
[20] https://github.com/kaldi-asr/kaldi/blob/master/egs/swbd/s5c/run.sh
[21] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_mono.sh
[22] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_deltas.sh
[23] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_lda_mllt.sh
[24] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/steps/train_sat.sh
[25] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/utils/dict_dir_add_pronprobs.sh
[26] https://kaldi-asr.org/doc/chain.html

Chapter 5

Experiments and results

This chapter presents experiments with different combinations of the training datasets and their augmentations. It also demonstrates the positive and negative contributions of the individual features presented and described in the previous chapters. Pruning and model size relationship is examined. Differently sized acoustic model architectures and their WER scores are shown.

5.1 Training on 18 hours of the company data mixed with the Supreme Court dataset

In this set of experiments, a 29.5 hour dataset of the perfectly annotated company data was used. The dataset was split into 11.5 hour test set and an 18 hour train set. The company data were combined with a subset of the Supreme Court dataset. Both datasets underwent different augmentation procedures. Different combinations of text corpuses for both n-gram LM and RNNLM were examined to show how manipulation with domain specific data may help to improve both perplexity and WER scores.

4-gram language models training on different data combinations

While training the n-gram language models I tried to prove the assumption from Section 2.5 stating that building separate domain-specific language models and applying an interpolation technique to combine them into a single language model optimised for the target use case shows better performance than having only a single LM trained on a mixed-domain dataset. I also show how better text normalization can improve the perplexity of the final interpolated model.

First I trained four 4-gram LMs on different text corpuses. The corpuses used were Google Billion Words corpus, all legal speech transcriptions found on the Internet combined with the court decision statements from The Caselaw Access Project, Supreme Court transcriptions and our company transcriptions. The resulting perplexities and log-linear weights of the final LM are shown in Table 5.1. In this setup, only a simple text normalization technique was applied. That meant basic numbers normalization (e.g. 1988 - *one thousand nine hundred and eighty eight* instead of *nineteen eighty eight*), letter case processing incorrectly handled some cases e.g. Mr. Underhill as a sentence end/beginning and only several special characters were processed in this normalization version such as dollars or at symbols inside email addresses.

Table 5.1: 4-*gram language models tested on a subset of 9k sentences of the company transcriptions not seen during the training. Log-linear weights were assigned by the interpolation procedure using the KenLM tool. The Size stands for number of sentences in the text corpus.*

Dataset	Size	Weight	Perplexity
Google Billion Words	42M	0.19	510
All Legal Domain Corpuses	23M	0.13	427
Supreme Court	4M	0.26	407
Parrot Transcriptions	17k	0.59	146
Interpolated LM	-	-	**126**

Table 5.2 compares the same text corpuses combination as in the previous setup, except a more robust text normalization was performed on them. That meant more rare use cases were handled, such as abbreviations spelling (FBI - F B I), numbers between 1900 and 2000 were normalized with the more common pronunciation (1988 - *nineteen eighty-eight*), which was not ideal for all cases, but this numbers range was more often spelled in this way than the other. More use cases in letter case processing were independently handled. The normalization procedure also detected and removed lots of structural data mainly in the Google Billion Words and Court Decisions corpuses (e.g. long IDs, tables and non-alphanumeric texts) which resulted in less text used for the models training. This was done by counting alphabetical and non-alphabetical characters in given texts. If the ratio was above a certain threshold, the text was not used for an LM training. For the results to be comparable with the ones from Table 5.1, the older version of normalization processed the test set.

Table 5.2: 4-*gram language models tested on a subset of 9k sentences of company transcriptions not seen during the training. Log-linear weights were assigned by the interpolation procedure. The same text corpuses combination as in Table 5.1 was used except more robust normalization was applied on them. The robust normalization removed a big part of the Google Billion Words and Court decisions corpuses.*

Dataset	Size	Weight	Perplexity
Google Billion Words	29M	0.18	339
All Legal Domain Corpuses	18M	0.51	214
Supreme Court	4M	-0.10	311
Parrot Transcriptions	17k	0.52	139
Interpolated LM	-	-	**124**

At this point Google Billion Words corpus represented 56% of all text data available. Such amount of data makes the size of the final model quite large. As this text corpus was far from our target domain, in the following two experiments I removed the Google Billion Words corpus, see Table 5.3. This slightly worsened the resulting perplexity, but given the amount of the removed text from the training, it is only a minor downgrade. This proved that a huge amount of training text doesn't really improves the final model quality if the data are far from the target domain.

Lastly, all of the text corpuses except Google Billion Words were split into five sets based on the origin of the data. This followed the main idea of interpolation mentioned

Table 5.3: 4-*gram language models tested on a subset of 9k sentences of company transcriptions not seen during the training. Log-linear weights were assigned by the interpolation procedure. In comparison to the experiment in Table 5.2, the Google Billion Words text corpus was not included in the interpolation.*

Dataset	Size	Weight	Perplexity
All Legal Domain Corpuses	18M	0.58	214
Supreme Court	4M	-0.02	311
Parrot Transcriptions	17k	0.55	139
Interpolated LM	-	-	**125**

earlier (more specialized LMs perform better than a single general one) and proved to be the best solution. The results are shown in Table 5.4.

Table 5.4: 4-*gram language models tested on a subset of 9k sentences of company transcriptions not seen during the training. Log-linear weights were assigned by the interpolation procedure. The same text corpuses as in the previous setups were used except they were splitted into several subsets depending on the origin of the data.*

Dataset	Size	Weight	Perplexity
Court Decisions	15.5M	0.23	340
Legal Transcriptions	2.5M	0.58	176
Supreme Court	1.5M	-0.08	337
Supreme Court from before 2000	2.5M	-0.11	340
Parrot Transcriptions	17k	0.48	139
Interpolated LM	-	-	**123**

Recurrent neural network language models training

Two RNNLMs for lattice rescoring were trained in this part. The first one used only two text corpuses for its training - Supreme Court transcriptions and our company transcriptions of the 18 hours of audio data while applying the original version of the simpler text normalization. The second RNNLM was trained on the same text corpuses as presented in Table 5.4 as this n-gram setup showed to have the best perplexity. The differences in the achieved WER scores can be seen later in the tables 5.5 and 5.6 for the first RNNLM version and the improved one respectively.

Acoustic models training

While training the following acoustic models, only several base ASR components were used. That means previously described n-gram language models, i-vectors and a CNN-TDNNf acoustic architecture. No online CMVN normalization, silence probabilities modelling or silence frames concatenation were applied in this set of experiments.

The small 18 hours dataset had to be extended by the Supreme Court data. 500 hours of the original Supreme Court data were augmented by the room impulse responses resulting

in 1,000 hours of acoustic data. The 18 hours of the company training data were extended by a 3-fold room impulse responses (RIRs) augmentation and 3-fold speed perturbation. The amount of augmentation was the same as in the original paper [15] where both 3-fold speed perturbation and 2(3)-fold RIRs augmentation was performed on the original data. This way the original 18 hours were extended to 220 hours of audio.

For all of the following decodings, the best WER score was achieved by setting the beam size 15, lattice pruning beam 12 and rescoring lattice beam 12. Higher values resulted in around 0.01% improvement but drastically increased the computing time thus the lower beam values seem to be more reasonable especially when planning on integrating the system into a production environment.

Table 5.5 shows how the model performed when trained only on the Supreme Court data in comparison to training on both the Supreme Court and company data. The augmented 18 hours helped to improve the final WER score by more than 10.6 points. Additionally, rescoring the lattices with an RNNLM further boosted the results by 3.4 points. The 4-gram language model used in these setups was the one described in Table 5.1, which means the basic text normalization and not the ideal text corpuses split during the interpolation were applied. The presented RNNLM for lattice rescoring was also trained on the basically normalized texts from the Supreme Court dataset and the company transcriptions. Similarly, the acoustic training stood on the same text normalization as the language models. The results in the table are compared to the transcriptions by the general Google speech recognition system.

Table 5.5: *Convolutional time-delay (cnn-tdnnf) models trained on the cleaned and augmented Supreme Court (s.c.) dataset (500h augmented to 990h) and 18 hours of our company data. The 18 hours of the company data were extended by augmentations to 220h. test*1 *is an 11.5 hour test set of the company data, test*2 *is a 60 hour subset of the Supreme Court dataset, both not seen during the training.*

	Training info:		WER [%]	
Acoustic model	**Language model**	**Dataset**	test1	test2
google-asr	unknown	unknown	35.0	-
cnn-tdnnf	4-gram	500h s.c. + 440h aug.	38.3	13.8
cnn-tdnnf	4-gram + lat. rescoring	500h s.c. + 440h aug.	34.9	15.1
cnn-tdnnf	4-gram	18h parrot + 200h aug. 500h s.c. + 440h aug.	27.63	13.96
cnn-tdnnf	4-gram + lat. rescoring	18h parrot + 200h aug. + 500h s.c. + 440h aug.	**24.21**	-

Models in Table 5.6 took advantage of the improved text normalization and an 4-gram language model interpolated on more appropriately split text corpuses as shown in Table 5.4. The RNNLM shown in this table was trained on the same texts as the 4-gram LM and the acoustic training used the improved text normalization too. As the data were normalized differently, the table shows WER scores on test sets normalized with both the basic (*test*1) and the improved (*test*3) text normalization.

Table 5.6: *Improved 4-gram LM from Table 5.4 (v2) and a new RNNLM both trained on the text corpuses normalized with the improved version of the text normalization compared to the basic normalization (v1). The same AMs as in Table 5.5 used. test1 is the company 11.5h test set same as the test1 in Table 5.5. test3 is the same set except normalized with the improved text normalization.*

	Training info:		WER [%]	
Acoustic model	**Language model**	**Dataset**	test1	test3
google-asr	unknown	unknown	35.0	35.1
cnn-tdnnf	4-gram (v1)	18h parrot + 200h aug. 500h s.c. + 440h aug.	27.63	-
cnn-tdnnf	4-gram (v1) + lat. rescoring	18h parrot + 200h aug. + 500h s.c. + 440h aug.	24.21	-
cnn-tdnnf	4-gram (v2)	18h parrot + 200h aug. 500h s.c. + 440h aug.	24.65	24.42
cnn-tdnnf	4-gram (v2) + lat. rescoring	18h parrot + 200h aug. + 500h s.c. + 440h aug.	22.59	**22.32**

The experiments showed that only a small amount of the target data can rapidly improve the accuracy of the ASR system. Lattice rescoring seems to bring a constant contribution in all setups. Both more robust text normalization and appropriate text corpuses split for the LMs interpolation helped to achieve a better WER score.

5.2 Training on 80 hours of the company data mixed with the Supreme Court dataset

After annotation of a larger amount of audio hours was finished, other experiments were performed to show how the additional data help to improve the WER score. The previously used data together with the newly annotated hours resulted in 90h of audio features. As the speaker set of the newly obtained audio intersected with the speaker set of the previous test and train sets, to sustain speaker uniqueness between the test and train sets and to not waste the audio data needlessly by adding it all to the already existing test set, a new split of the dataset was necessary. Thus the new WER scores are not directly comparable with the previous setups. However, using the Google speech recognition as a baseline system, the results can be still compared at least relatively although different test sets were used.

The new train set of the company data consisted of 80 hours of audio, the test set remained to have 10 hours of audio. The data in the following experiments were also mixed with the Supreme Court dataset and augmented with both room impulse responses and speed perturbation. New language models were prepared containing the additional company transcriptions.

The newly trained 4-gram language model used the same text corpuses split as in Table 5.4, except the company transcriptions contained more data. The perplexity of the final interpolated model is shown in Table 5.7. Due to the new test and train sets split, the test set was different from the one in Table 5.4 thus the results are not directly comparable between the tables.

Table 5.7: *4-gram language models tested on a subset of 7k sentences of company transcriptions not seen during the training. Log-linear weights were assigned by the interpolation procedure. Transcriptions of 80 hours of the company data were used in the Parrot Transcriptions corpus in comparison with the 4-gram in Table 5.4 being trained on transcriptions of 18 hours of the company data. Different test sets were used thus the perplexities are not comparable.*

Dataset	**Size**	**Weight**	**Perplexity**
Court Decisions	15.5M	0.19	326
Legal Transcriptions	2.5M	0.33	171
Supreme Court	1.5M	-0.11	334
Supreme Court from before 2000	2.5M	-0.09	347
Parrot Transcriptions	43k	0.80	92
Interpolated LM	-	-	**94**

Next, using the enlarged company dataset, I executed the same experiments as previously with the smaller dataset. The augmentations of the Supreme Court data were the same as in the previous experiments, which means 2-fold RIRs augmentation resulting in 1000 hours of features. For the company data 3-fold speed perturbation combined with 2-fold RIRs augmentation was applied resulting in 480 hours of features.

As mentioned above, the improvements cannot be directly compared to the models in Table 5.6 as they were evaluated on a different test set. Instead, we can estimate the improvement of the models by measuring the differences between our models and the Google speech recognition system on different test sets.

When training with the smaller dataset, the difference between our model and the Google speech recognition system was 10.68 points without and 12.78 points with lattice rescoring, see Table 5.6. In the new experiments shown in Table 5.8, the same acoustic and language models setup brought improvement of 14.8 without rescoring in comparison to the Google speech recognition system when trained on the dataset containing 80 hours of the company data instead of 18 hours.

Table 5.8 also demonstrates how different amounts of the Supreme Court hours increase or decrease the WER score. Visible differences were found when testing various Supreme Court augmentations approaches. Specifically, five approaches in usage of the Supreme Court data were tested, all of which combined with a constant amount of 80 hours of the company data augmented to 480 hours:

1. 480 company hours only.
2. 480 company hours with 500 clean (not augmented) Supreme Court hours.
3. 480 company hours with 500 Supreme Court hours all of which contained inserted room impulse responses.
4. 480 company hours with 500 clean Supreme Court hours augmented to 1000 hours half of which was the original not augmented data and the second half contained room impulse responses.
5. 480 company hours with 1000 clean Supreme Court hours without augmentations.

Table 5.8: *Various combinations of the company data (80 hours augmented to 480 hours using 3-fold speed perturbation and 2-fold room impulse responses augmentation) and the Supreme Court data both augmented and not augmented (or both). test4 is a 10 hour test set from the unseen company data.*

Training info:			WER [%]
Acoustic model	**Language model**	**Dataset**	test4
google-asr	unknown	unknown	37.4
cnn-tdnnf-a	4-gram	80h parrot + 400h aug.	23.88
cnn-tdnnf-b	4-gram	80h parrot + 400h aug. 500h clean s.c.	22.89
cnn-tdnnf-c	4-gram	80h parrot + 400h aug. 500h aug. s.c.	**22.57**
cnn-tdnnf-d	4-gram	80h parrot + 400h aug. 1000h clean s.c.	23.25
cnn-tdnnf-e	4-gram	80h parrot + 400h aug. 500h clean s.c. + 500h aug.	23.20

The results in Table 5.8 clearly show that the Supreme Court data addition helps to improve the final WER score (e.g. compare cnn-tdnnf-a and cnn-tdnnf-b models). However if too much data from this dataset is used during training, the model loses performance on the company target domain test set (models *cnn-tdnnf-d* and *cnn-tdnnf-e* from Table 5.8). Also, inserting room impulse responses to the Supreme Court data tends to result in a lower WER score than when training on the same subset except not being reverberated, see models *cnn-tdnnf-b* versus *cnn-tdnnf-c* and *cnn-tdnnf-d* compared to *cnn-tdnnf-e* in Table 5.8.

5.3 Experiments with additional components

This section describes experiments with all of the additional components mentioned earlier in the chapters 2 and 4 and their negative or positive contributions. Namely, silence frames concatenation to the training utterances, silence pronunciation modelling and online cepstral mean and variance normalization. Then I show how different pruning options affect the size on the resulting model and its accuracy. Also, acoustic and language models with different number of layers are compared in the end of the section.

Silence frames concatenation to training utterances

In the setup, results of which are demonstrated in Table 5.9, all utterances in the company data were concatenated with randomly selected 30ms audio chunks (from a set of manually prepared 100 chunks) containing silence or non-speech, which was meant to prevent the HMM states from wrong silence modelling at the starts and ends of utterances. The rest of the training setup remained the same as in Table 5.8. When comparing the WER scores obtained by this model, they are worse in all cases by about 1 percent point.

Although the results on the *test*4 test set may suggest that this feature worsens the overall ASR performance, I examined the individual decoded utterance texts manually

Table 5.9: *Model trained on utterances with 30ms silence audio chunks concatenated to their starts and ends. The results can be compared to Table 5.8 where the same models where used except for the silence audio chunks concatenation. The WER scores were measured on two test sets, test4 is the same test set as in Table 5.8, test4$_{silframes}$ is the same test set except randomly selected silence chunks were concatenated to the individual utterances stars and ends.*

Training info:			WER [%]	
Acoustic m.	**Language m.**	**Dataset**	test4$_{silframes}$	test4
google-asr	unknown	unknown	-	37.4
cnn-tdnnf	4-gram	80h parrot + 400h aug.	24.39	24.46
cnn-tdnnf	4-gram	80h parrot + 400h aug. 500h clean s.c.	**23.19**	**23.55**
cnn-tdnnf	4-gram	80h parrot + 400h aug. 1000h clean s.c.	23.75	24.12
cnn-tdnnf	4-gram	80h parrot + 400h aug. 500h clean s.c. + 500h aug.	23.25	24.18

and find out the opposite may be true. The *test*4 contains many short utterances. As the utterances of *test*4 were not concatenated with the silence audio frames although the train set utterances were, it made it more difficult for the model trained on utterances with silence frames concatenated to decode the test utterances without them, especially when they are very short thus their starts and ends play a bigger role. The model may have been too confused at utterance beginnings. When it started to recover it was already at the end of an utterances where it got confused again as no silence was present there. However, this problem was less observable in long utterances where the model had more data to recover from the missing silence at an utterance beginning. Table 5.10 compares WER scores achieved on long utterances by models trained with and without silence frames concatenation.

Table 5.10: *Measurements showing by how much percent the model trained on utterances concatenated with silence frames outperformed the model trained on utterances without the silence frames concatenated (negative values suggest the model performed worse). Min. utterance length was the minimum number of characters in the reference utterances to be selected for the comparison.*

Min. utterance length	**Utterance count**	**WER difference**
80	57	**0.55**
70	117	0.05
60	174	0.13
50	272	0.04
40	427	-0.11
30	663	-0.22
None	8009$_{(all)}$	-0.66

As the number of long utterances in the test set was relatively small, the comparison in Table 5.10 doesn't have to be necessarily correct. Another comparison was performed on a unique Supreme Court subset disjunct with the training set. The subset contained 6000 utterances all of which were longer than 100 characters. Both models, trained with and without silence frames concatenation, achieved the same WER score 18.64%. This comparison also doesn't answer the question whether the silence frames concatenation really helps but it suggests that it doesn't worsen the results on long utterances. This means that although the feature doesn't improve the final WER score on our test set, it can be integrated into the production system where only long utterances exist.

Silence probabilities pronunciation modelling

According to the original paper introducing the silence pronunciation probabilities modelling, this feature brings small but stable improvements across different datasets. I tested the feature on two setups. In these experiments, the silence pronunciation probabilities modelling was included in training of the last model of the GMM models sequence and in training of the consequent TDNNf architecture. A Kaldi script[1] generated the pronunciation probabilities. The results and achieved WER scores are presented in Table 5.11.

The feature showed no improvement on neither short nor long utterances; the difference in WER scores on long utterances subset was even worse than on the whole test set. Manual examination of the automatic transcriptions didn't end up with any reasonable explanation. A common problem of the model with silence pronunciation probabilities modeling was spelling of single letters. While the model without the pronunciation modelling usually didn't have problems with spelling of e.g. names, the model with the pronunciation modelling usually tried to substitute the spelled letters with similar words, e.g. *'and they are J are high to ya'* instead of *'M A R G A R I T'*.

Table 5.11: *Silence pronunciation probabilities modelling used with models trained on dataset with and without concatenated silence frames compared to the model not modelling the silence pronunciation.*

Training info:			WER [%]
Acoustic model	**Language model**	**Dataset**	test4
cnn-tdnnf	4-gram	80h parrot + 400h aug. 500h clean s.c.	**22.89**
cnn-tdnnf	4-gram$_{silprob}$	80h parrot + 400h aug. 500h clean. s.c.	23.76
cnn-tdnnf$_{silframes}$	4-gram$_{silprob}$	80h parrot + 400h aug. 500h clean. s.c.	24.87

Online cepstral mean and variance normalization

While training acoustic models, Kaldi provides an option to train with online cepstral mean and variance normalization. Table 5.12 shows improvement brought by this feature. Both models in the table are the same except for the online cmvn normalization turned on and off.

[1] https://github.com/kaldi-asr/kaldi/blob/master/egs/wsj/s5/utils/dict_dir_add_pronprobs.sh

The model used was the best model from Table 5.8, meaning cnn-tdnnf acoustic model with a 4-gram LM trained on 80 company hours augmented to 480 hours and 500 Supreme Court hours all of which combined with RIRs. Improvement of the online cmvn normalization turned out to be 0.14 points.

Table 5.12: *Models trained with and without online cepstral mean and variance normalization.*

Model	WER [%] test4
cnn-tdnnf$_{without_online_cmvn}$	22.57
cnn-tdnnf$_{with_online_cmvn}$	22.43

RNNLM lattice rescoring

A new RNNLM for lattice rescoring was trained with transcriptions of all 80 hours of company data used together with other text corpuses present in Table 5.7. Table 5.13 shows how the new RNNLM performs in comparison to the one trained on texts containing less company data. Only 0.01 improvement was achieved by this data addition which suggests that such text amount is probably not enough to move the weights of the model. The table also shows how a bigger architecture performed. It showed that adding one more recurrent layer (3 instead of 2 LSTM layers) does not change the resulting score at all.

Table 5.13: *Comparison of RNNLM rescoring using an LM trained on different amounts of target domain data and in combination with the same amount of texts from other text corpuses, specifically texts described in tables 5.4 and 5.7. Also a model with one more LSTM layer presented with no improvement in the WER score.*

Training info:			WER [%]
Acoustic model	**Language model**	**Dataset**	test4
cnn-tdnnf	no rescoring	80h parrot + 400h aug. 500h aug. s.c.	22.57
cnn-tdnnf	RNNLM$_{old}$	80h parrot + 400h aug. 500h aug. s.c.	20.76
cnn-tdnnf	RNNLM$_{new}$	80h parrot + 400h aug. 500h aug. s.c.	20.75
cnn-tdnnf	RNNLM$_{new_larger}$	80h parrot + 400h aug. 500h aug. s.c.	20.75

Pruning language models to decrease the decoding graph size

Training an n-gram LM on large portions of text data can make the resulting model size huge as all of the n-gram combinations have to be stored with the model. Although big LMs can improve the ASR quality, their usage in a production environment can be impractical as such model consumes lots of memory, its loading takes more time and the decoding itself is longer. The bigger the model the higher the requirements for all its resources. To be more

specific, the models from Table 5.8 used a 4-gram arpa format LM with size 5.8G, which resulted in an HCLG decoding graph of 20G, meaning the model takes initially 20GB of memory before the decoding even starts. Thus I experimented with different sized n-gram LMs where the size was regulated by pruning the models.

Pruning an n-gram LM means that if a certain $(n-i)$-gram for non negative i's appeared less times than a given threshold in a corpus, the $(n-i)$-gram won't be saved in the model. Setting the threshold to values low enough doesn't have to necessarily worsen the model prediction capacities as the $(n-i)$-grams that appeared only once or twice in the whole corpus are often not so important or are nothing but noise.

Results in Table 5.14 demonstrate how different pruning thresholds affected the final size of the LM in the arpa format and the achieved accuracy. Different pruning settings for different text corpuses were applied. The text corpuses that were assigned lower weights during the interpolation procedure were pruned more then the other LMs. The table uses KenLM pruning format - ($a^{(1)}$ $b^{(2)}$ $c^{(3)}$ $d^{(4)}$) are pruning thresholds for unigrams, bi-grams, 3-grams and 4-grams. KenLM doesn't support pruning of unigrams thus the threshold for them was set to zero in all setups. I decided not to prune the LM trained on the company transcriptions at all in any of the setups as this corpus was the smallest and most important one. The last row of the table shows an LM trained only on the company transcriptions without any pruning. Although the LM size was very small, it visibly worsened the final WER score.

Table 5.14: *Demonstration of how pruning of an 4-gram LM changes its size (arpa format) and its prediction capabilities. Different pruning settings were used for different text corpuses which were then interpolated. The LMs with lower interpolation weights were pruned more than the others. The pruning thresholds in this table are in this order: ($a^{(1)}$ $b^{(2)}$ $c^{(3)}$ $d^{(4)}$) for unigrams, bi-grams, 3-grams and 4-grams, each corpus on separate line. The corpuses are displayed in this order: court decisions, general transcriptions, Supreme Court transcriptions, Supreme Court transcriptions before 2000, company transcriptions. The last row is a 4-gram trained only on the company transcriptions with no pruning.*

Pruning thresholds	**Perplexity**	**LM size**	**WER** [%]
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1	94.5	5.8G	22.57
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0	93.9	3.4G	22.60
0 1 1 3 0 0 1 3 0 1 2 3 0 1 2 3 0 0 0 0	94.3	1.8G	22.59
0 3 3 3 0 1 1 3 0 5 5 5 0 5 5 5 0 0 0 0	95.1	1.2G	22.70
0 0 0 0	92	32M	24.26

Different sizes of acoustic architectures

In this part I examined how different numbers of layers in the acoustic model change the resulting WER score. Table 5.15 shows setups with different numbers of CNN and TDNN layers in the model architecture. The first row is the architecture described so far in the previous setups. It is clear that by increasing the number of layers the WER score is decreasing. More experiments with even bigger architectures will have to be performed in the future.

Table 5.15: *Comparison of acoustic architectures with different numbers of CNN and TDNN layers. The data used were from the best previous setup - company 80h augmented to 480h and 500h of Supreme Court with room impulse responses.*

#CNN	**#TDNN**	WER [%] test4
6	9	22.43
7	9	22.35
7	10	22.28
8	11	**21.97**

Final speech recognition setups with best WER scores

Finally, the best model from all of the above experiments was chosen to represent the overall achievement of this work. Table 5.16 shows this model compared with the google speech recognition. The best performance was achieved by training the acoustic model with 8 CNN layers and 11 TDNNf layers on 80 hours of company audio data augmented to 480 hours by 3-fold speed perturbation and 2-fold room impulse responses. The data were mixed with 500 hours of Supreme Court transcriptions. The Supreme Data were mixed with room impulse responses (1-fold augmentation). Online CMVN was used and the lattices were rescored with a RNNLM with 2 LSTM layers.

Table 5.16: *The final setup with the best WER score compared to Google speech recognition. The setup is an architecture of 8 CNN layers, 11 TDNN layers, with online CMVN, trained on 80 company hours augmented to 480 with 3-fold speed perturbation and 2-fold room impulse response augmentation in combination with 500 hours from the Supreme Court dataset (with 1-fold room impulse responses augmentation). The language model is a 4-gram LM with pruning option set to 0 0 0 1 for all corpuses, meaning only 4-grams occurring exactly once in the training corpus are removed. The lattices are rescored with an RNNLM. Both the 4-gram and the RNNLM were trained on data from Table 5.7 (~22M sentences).*

Acoustic model	Training info: **Language model**	**Dataset**	WER [%] test4
google-asr	unknown	unknown	37.4
cnn-tdnnf	4-gram + lat. rescoring	80h parrot + 400h aug. 500h aug. s.c.	**20.17**

Chapter 6

Conclusion

This work demonstrated how to build a hybrid automatic speech recognition system using a Kaldi speech recognition toolkit. Using relatively small amount of the target domain data together with a freely available Supreme Court hearings dataset, a large 17.2% improvement over the general ASR system from Google was achieved. This was possible due to precise manual data annotation and automatic data cleaning together with usage of state-of-the-art speech recognition techniques provided by Kaldi.

Several experiments were performed while training multiple domain specific 4-gram language models to be interpolated to a single language model. The work showed how the target domain data help with improving the WER scores. Experiments were run showing what is the optimal amount of different domain data to be add to a training to improve the accuracy of the model the most. Room impulse responses and speed perturbation were applied as augmentation in different combinations. Recurrent neural network language models trained on different amounts of data brought a stable improvement in all setups.

Experimental results were presented using silence frames concatenation to the training utterances. This feature didn't bring an improvement on the used test sets, however closer examination of the decoded texts in a subset of long utterances indicated that the feature may be of help in real world scenarios where only long utterances are usually decoded. This feature should be further tested and experimented with.

Despite statements in the original cited paper presenting inter-word silence probabilities modelling claiming that this feature decreases WER scores consistently across datasets, using it in my work increased the WER score in two experiments.

Different n-gram LM pruning setups were presented to show how does the model size depends on the pruning frequency. By obtaining a slightly worse performance (0.1%) a much smaller (x4) decoding graph could be build which is more suitable for production usage. Additionally, multiple experiments with the size of the acoustic model and RNNLM architectures showed that more layers tends to decrease the WER score for the acoustic model while remaining the same for the RNNLM. More experiments have to be run in the future with even bigger architectures.

Finally, the best ASR setup given the available data was presented, combining the positively contributing features described or introduced in this work. In the future this ASR system may be trained on more target domain data to further improve its quality. Also, the neural architectures used to train this model can be adjusted and the best hyper-parameters should be found to achieve even better results by e.g. adding more or different layers to both RNNLM and acoustic models, different learning rate setup and/or different number of training epochs (early stopping). The silence frames concatenation feature should be also

examined more in detail to find out if and how much does it help in real world scenarios. In the future I'm also planning on experimenting with different orders of the n-gram language model.

Bibliography

[1] Abdel Hamid, O., Mohamed, A.-R., Jiang, H., Deng, L., Penn, G. et al. Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* IEEE Press. october 2014, vol. 22, no. 10, p. 1533–1545. Available at: https://doi.org/10.1109/TASLP.2014.2339736. ISSN 2329-9290.

[2] Bishop, C. M. *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995. ISBN 0198538642.

[3] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T. et al. *One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling*. 2013.

[5] Chen, G., Xu, H., Wu, M., Povey, D. and Khudanpur, S. Pronunciation and silence probability modeling for ASR. In: *INTERSPEECH*. 2015.

[6] Dan Povey, Vijay Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, Sanjeev Khudanpur. *Purely sequence-trained neural networks for ASR based on lattice-free MMI* [url: http://people.csail.mit.edu/mitra/meetings/2016-Sep27-Vijay.pdf]. September 2016.

[7] Dempster, A. P., Laird, N. M. and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*. 1977, vol. 39, no. 1, p. 1–38.

[8] Fellbaum, C. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[9] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*. Nov 2012, vol. 29, no. 6, p. 82–97. ISSN 1558-0792.

[10] Hochreiter, S. and Schmidhuber, J. Long Short-term Memory. *Neural computation*. december 1997, vol. 9, p. 1735–80.

[11] Itakura, F. Line spectrum representation of linear predictor coefficients of speech signals. *The Journal of the Acoustical Society of America*. 1975, vol. 57, S1, p. S35–S35. Available at: https://doi.org/10.1121/1.1995189.

[12] Jelinek, F. and Mercer, R. L. Interpolated estimation of Markov source parameters from sparse data. In: Gelsema, E. S. and Kanal, L. N.,

ed. *Proceedings, Workshop on Pattern Recognition in Practice*. Amsterdam: North Holland, 1980, p. 381–397.

[13] Klakow, D. Log-linear interpolation of language models. In:. January 1998.

[14] Kneser, R. and Ney, H. Improved backing-off for M-gram language modeling. *1995 International Conference on Acoustics, Speech, and Signal Processing*. 1995, vol. 1, p. 181–184 vol.1.

[15] Ko, T., Peddinti, V., Povey, D., Seltzer, M. L. and Khudanpur, S. A study on data augmentation of reverberant speech for robust speech recognition. In: IEEE. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, p. 5220–5224.

[16] LeCun, Y., Bengio, Y. and Hinton, G. Deep Learning. *Nature*. 2015, vol. 521, no. 7553, p. 436–444. Available at: https://doi.org/10.1038/nature14539.

[17] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. Taipei, Taiwan. 1998, vol. 86, no. 11, p. 2278–2324.

[18] Messina, R. Acoustical Modeling for Speech Recognition: Long Units and Multi-Modeling. june 2015.

[19] Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J. and Khudanpur, S. Recurrent neural network based language model. In: *Eleventh annual conference of the international speech communication association*. 2010.

[20] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*. 2013, p. 3111–3119.

[21] Mohri, M. *Speech Recognition, Lecture 12: Lattice Algorithms.* [online]. Available at: https://cs.nyu.edu/~mohri/asr12/lecture_12.pdf.

[22] Mohri, M., Pereira, F. and Riley, M. Speech recognition with weighted finite-state transducers. In: *Springer Handbook of Speech Processing*. Springer, 2008, p. 559–584.

[23] Moreno, P. J., Joerg, C. F., Thong, J.-M. V. and Glickman, O. A recursive algorithm for the forced alignment of very long audio segments. In: *ICSLP*. ISCA, 1998. Available at: http://dblp.uni-trier.de/db/conf/interspeech/icslp1998.html#MorenoJTG98.

[24] Muda, L., Begam, M. and Elamvazuthi, I. Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques. *CoRR*. 2010, abs/1003.4083. Available at: http://arxiv.org/abs/1003.4083.

[25] Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, p. 807–814.

[26] Peddinti, V., Povey, D. and Khudanpur, S. A time delay neural network architecture for efficient modeling of long temporal contexts. In: *INTERSPEECH*. 2015.

[27] Povey, D., Cheng, G., Wang, Y., Li, K., Xu, H. et al. Semi-Orthogonal Low-Rank Matrix Factorization for Deep Neural Networks. In: *Interspeech*. 2018, p. 3743–3747.

[28] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O. et al. The Kaldi speech recognition toolkit. *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. january 2011.

[29] Povey, D., Peddinti, V., Galvez, D., Ghahremani, P., Manohar, V. et al. Purely Sequence-Trained Neural Networks for ASR Based on Lattice-Free MMI. In:. September 2016, p. 2751–2755.

[30] Pusateri, E., Van Gysel, C., Botros, R., Badaskar, S., Hannemann, M. et al. Connecting and Comparing Language Model Interpolation Techniques. *ArXiv preprint arXiv:1908.09738*. 2019.

[31] Ruder, S. An overview of gradient descent optimization algorithms. *CoRR*. 2016, abs/1609.04747. Available at: http://arxiv.org/abs/1609.04747.

[32] Rumelhart, D. E., Hinton, G. E. and McClelland, J. L. A General Framework for Parallel Distributed Processing. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, p. 45–76. ISBN 026268053X.

[33] Saon, G., Soltau, H., Nahamoo, D. and Picheny, M. Speaker adaptation of neural network acoustic models using i-vectors. In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. 2013, p. 55–59.

[34] Stamp, M. *A revealing introduction to hidden Markov models*. 2004.

[35] Steve Young. A review of large-vocabulary continuous-speech. *IEEE Signal Processing Magazine*. Sep. 1996, vol. 13, no. 5, p. 45–. ISSN 1558-0792.

[36] Strassel, S., Graff, D., Martey, N. and Cieri, C. Quality Control in Large Annotation Projects Involving Multiple Judges: The Case of the TDT Corpora. In: *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*. Athens, Greece: European Language Resources Association (ELRA), May 2000. Available at: http://www.lrec-conf.org/proceedings/lrec2000/pdf/212.pdf.

[37] Veselý, K., Ghoshal, A., Burget, L. and Povey, D. Sequence-discriminative Training of Deep Neural Networks. In: *Proceedings of Interspeech 2013*. International Speech Communication Association, 2013, p. 2345–2349. Available at: https://www.fit.vut.cz/research/publication/10422. ISBN 978-1-62993-443-3.

[38] Veselý, K. *The project Kaldi – Open source speech recognition*. [online]. 2018. Available at: https://www.fit.vutbr.cz/study/courses/ZRE/public/pred/14_Kaldi/2018-kaldislidesforzre.pdf.

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

[40] Warden, P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints*. april 2018. Available at: https://arxiv.org/abs/1804.03209.

[41] Xu, H., Chen, T., Gao, D., Wang, Y., Li, K. et al. A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition. In: IEEE. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5929–5933.

[42] Xu, H., Li, K., Wang, Y., Wang, J., Kang, S. et al. Neural network language modeling with letter-based features and importance sampling. In: IEEE. *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. 2018, p. 6109–6113.

[43] Zhang, X., Trmal, J., Povey, D. and Khudanpur, S. Improving deep neural network acoustic models using generalized maxout networks. In: IEEE. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, p. 215–219.