MSE | MASTER OF SCIENCE IN ENGINEERING

Speech-To-Text Engine for Forced Alignment

Master Thesis

By

Daniel Tiefenauer
daniel.tiefenauer@students.fhnw.ch

n|w University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

Institute for Data Science (I4DS)

Advisor	Manfred Vogel (manfred.vogel@fhnw.ch)
Expert	Mark Cieliebak (mark.cieliebak@zhaw.ch)
Industry Partner	ReadyLingua
	Ulrike Glavitsch (ulrike.glavitsch@gmail.com)

December 17, 2018

Abstract

tbd.

Contents

1 Introduction

This report documents the progress of the project *Speech-To-Text Engine for Forced Alignment*, my master thesis at University of Applied Sciences (FHNW) (referred to as *IP9*). Some preliminary work has been done in a previous project (referred to as *IP8*). The overall goal, project situation and some background information are described in detail in the project report for IP8 and shall not be repeated here. Only a quick recap of the relevant terms and aspects is given as far as they are relevant for the understanding of this document. A list of abbreviations used is given at the end of the document.

1.1 Scope and overall goal

ReadyLingua (RL) is a Swiss-based company that develops tools and produces content for language learning. Some of this content consists of audio/video data with an accompanying transcript. The overall goal is to enrich the textual data with temporal information, so that for each part of the transcript the corresponding location in the audio/video data can be found. This process is called *Forced Alignment (FA)*. An *InnoSuisse* project was started in 2018 to research how this could be achieved. The *InnoSuisse* project plan contains three different approaches, one of which is pursued in this project.

1.2 Chosen approach and previous work

The approach chosen for this project is based on speech pauses, which can be detected using *Voice Activity Detection (VAD)*. The utterances in between are transcribed using *Automatic Speech Recognition (ASR)*, for which a *Recurrent Neural Network (RNN)* is used. The resulting partial transcripts contain the desired temporal information and can be matched up with the full transcript by means of Sequence Alignment (SA).

All theses parts can be treated as stages of a pipeline:

- **VAD**: the audio is split into non-silent parts (*speech segments*)
- **ASR**: each speech segment is transcribed resulting in a partial (possibly faulty) transcript
- **SA**: each partial transcript is localized within the original transcript

Since the quality of the ASR stage has an imminent impact on the subsequent SA stage, the quality of the alignments depends heavily on the quality of the partial transcripts. This makes the ASR stage the crucial stage of the pipeline. However, ASR is highly prone to external influences like background noise, properties of the speaker (gender, speaking rate, pitch, loudness). Apart from that, language is inherently abiguous (e.g. accents), inconsistent (e.g. linguistic subtleties like homonyms or homophones) and messy (stuttering, unwanted repetitions, mispronunciation).

1.2.1 Previous results and problems

For the VAD stage, an implementation[1] of *WebRTC*[2] was used. This implementation has proved itself capable of detecting utterances with very high accuracy within reasonable time. For the SA stage a combination of the Smith Waterman (SW) algorithm and the Levenshtein distance was used to produce a local alignment for each partial transcript. This combination included tunable parameters like the minimum required similarity between an alignment and its underlying text from the transcript. It and was able to localize potentially erroneous partial transcripts within the full transcript pretty well, provided the similarity between actual and predicted text was high enough. Since each partial

[1] `https://github.com/wiseman/py-webrtcvad`
[2] `https://webrtc.org/`

transcript was aligned in isolation, the SA stage was actually a Local Sequence Alignment (LSA) stage.

For the ASR stage on the other hand, no RNN could be trained that was capable of transcribing the audio segments with a quality high enough for the LSA stage. The main problems were the lack of readily available high-grade training data, very long training times and as a result also very long feedback cycles. Because the ASR stage is at the heart of the pipeline, the self-trained model was replaced by a proprietary solution from Google. For this, API-calls to Google Cloud Speech (GCS)[3] provided the necessary partial transcripts. Using this engine, the pipeline was able to produce very good (although not perfect) transcripts for the individual utterances. Therefore the chosen approach was validated and the pipeline could shown to be generally functional. On the other hand, embedding GCS as the ASR part of the pipeline made the pipeline dependent on a commercial product, whose inner workings remain unknown and who cannot be tuned to the project's needs. Furthermore, although the transcriptions produced by GCS are very accurate, this quality might be an overkill for the purpose of this project. Last but not least the API calls are subject to charges incurring considerably high costs when used on large amounts of data. For these reasons, a partial goal of this project is to research under what circumstances a standalone STT model can be trained which is able to infer transcripts with sufficiently high quality.

The IP8 project proposed the use of *DeepSpeech* for the ASR stage, which uses Connectionist Temporal Classification (CTC) (**ctc˙paper**) as its cost function. Some experiments were made to find out what features can be used to train a RNN for the ASR stage. The features considered were raw power-spectrograms (as stipulated by the *DeepSpeech* paper), Mel-Spectrograms and Mel-Frequency Cepstral Coefficients (MFCC). It was found that training on MFCC features would probably require the least amount of training data because. An RNN using a simplified version of the *DeepSpeech* architecture was trained on data from the *LibriSpeech* project (containing only English samples).

1.3 Goal of this project

In this project, the chosen pipelined approach shall further be refined. Because the VAD and the LSA stage already work pretty well, the focus in this project lies on the ASR stage. Because the pipeline should become language-agnostic and self-contained, a RNN must be trained that can be used in this stage in the pipeline. Such a RNN could be a simplified variant of the *DeepSpeech* model, like the one implemented in the IP8 projects.

The sequence alignment stage in the pipeline is tolerant to a certain amount of errors in the transcripts. This means training the RNN will happen under the following premises:

- The RNN should be as simple as possible and as complex as necessary.
- The RNN only needs to be *good enough* for the task at hand which is FA and not speech recognition.

The reason for the first premise is that more complex neural networks usually require more training data. A network architecture requiring only little training data opens up to minority languages like Swiss German, where training data might be scarce.

The reason for the second premise data efficiency. While a simpler model will probably not be able to produce transcripts with the same accuracy as a complex model, this quality may not be required in the first place.

The goal of this project is therefore to make statements as to under what conditions the ASR stage can be implemented. For this, various combinations of network or data properties are explored as well

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

as varying amounts of training data. Concretely, the following questions shall be addressed:

- **How does the quality of the simplified *DeepSpeech*-RNN change with increasing training data?** By plotting the learning curve we should be able to see whether the RNN is able to learn something useful at all and also get some intuition about how much training data is needed to get reasonably accurate partial transcripts.
- **How does the quality of the partial transcripts change when using synthesized training data?** Neural Network usually require large amounts of training data and often improve with increasing size of the training set. However, labelled training data is usually difficult and/or expensive to acquire. For the purpose of Forced Alignment however, synthesized training data can be easily obtained by adding some distortion to the original signal (reverb, change of pitch, change of tempo, etc.).
- **How does the quality of the partial transcript change when integrating a LM?** STT-engines traditionally use a LM that models the probabilities of characters, words or sentences. A LM can help producing valid transcripts by mapping sequences of characters (that may sound similar to what was actually said) to orthographically correct sentences.
- **How can we assess the quality of the alignments?** This should give us some insight about how the quality of the alignment changes with varying capability of the STT-engine and what quality of transcripts is required.

Answering above questions should help estimating the effort to create a generic pipeline. [4]

1.4 Summary

This chapter gave an introduction into the project, its scope and goal. It also gave a quick overview over the preliminary work done in the IP8 project by outlining problems and impediments experienced there.

[4] Because ASR is highly dependent on the language that should be recognized, a different STT system has to be trained for each language.

2 Training a Neural Network for Speech Recognition

As stated above, the title of this thesis may be a bit misleading because the focus for this project is not on training a state of the art STT engine. This chapter describes the reference model, the simplified model and how they were compared.

2.1 *DeepSpeech*: A reference model

A Neural Network (NN) that had quite an impact on ASR was *DeepSpeech* (**deepspeech**). It reached recognition rates near-par to human performance, despite using a comparably simpler than traditional speech systems. Because the relation between audio signal and text was learned end-to-end (E2E) the network was also pretty robust to distortions like background noise or speaker variation. An open source implementation of *DeepSpeech* is available from Mozilla [5]. This implementation was written in C and Python and uses the *TensorFlow* framework. Although sticking to the architecture proposed by **deepspeech**, it represents a variant of of the model proposed in the paper (**ctc˙paper**), because it uses MFCC as features whereas the original paper proposes raw spectrograms. Since the implementation also uses a LM, the quality of the model is measured as the percentage of misspelled or wrong words (referred to as Word Error Rate (WER)) or as the edit distance (also called Levenshtein distance or Label Error Rate (LER)). A pre-trained model for inference of English transcript can be downloaded, which achieves a WER of just 6.5%, which is close to what a human is able to recognize (**mozillajourney**). *DeepSpeech* serves as a reference model for the simplified model used in this project.

2.2 Related research

The idea of training a STT model on limited data has also been researched by **budget**, although with a different approach. Instead of training a RNN from scratch, they used a Convolutional Neural Network (CNN) trained to recognize English as a base. This network used the *Wav2Letter* (**wav2letter**) architecture whose lower layers were frozen and whose higher layers were re-trained on a (smaller) German corpus, a process is also known as Transfer Learning.

Similar to *DeepSpeech*, the model trained by **budget** uses a LM to decode the model output into character sequences, CTC as its loss function[6] and (mel-scaled) spectrograms as features. The German audio and text data used to train the higher layers were taken from several (very heterogenous) corpora from the Bavarian Archive for Speech Signals (BAS).

budget were lead by the assumption that languages – especially such belonging to the same family – share common features that can be transferred by sharing the pre-trained lower layers that detect them. This assumption was proved true, yielding a model that could be trained faster and with fewer training data while producing results of similar quality up to a certain training time.

Although the experiments conducted by **budget** suggest an interesting starting point to train a model for the ASR stage in this pipeline, this is not the path taken in this project for the following reasons:

- The training data used by **budget** is still much larger than the data available from the IP8 project. Achieving the same amount of training data would require some heavy preprocessing, which according to experience eats up most of the project time. It would also mean that the efforts made in the IP8 project are discarded.
- Because **budget** follow a completely different approach than this project, it would mean starting from scratch and neglecting the insights made in the IP8 project.

[5] `https://github.com/mozilla/DeepSpeech`
[6] although the original *Wav2Letter* model uses an alternative loss function

- Since *DeepSpeech* is used as the reference model, it may make more sense comparing it to a simplified version of itself rather than a completely different model using a CNN architecture, because the impact of the architecture is much more limited.

For those reasons I consider the experiments made by **budget** an interesting alternative. However, to leverage the efforts made in the IP8 project, the goal for this project is still to train a stripped-down version of the *DeepSpeech* model.

2.3 Exploiting the *DeepSpeech* model

The final FA pipeline should provide alignments for any language. One possible approach would be to train a model using the existing Mozilla implementation by providing training-, validation- and test-data for each language. However, this approach does not fullfill the premises initially made:

1. The *DeepSpeech* implementation was explicitly designed for ASR. In such settings a low WER is desirable. But because accurate speech recognition is not the main concern in this project, the architecture of the Mozilla implementation might be overly complicated.
2. The Mozilla implementation requires an (optional) LM, which is tightly integrated with the training process which might not be available for the target languages.

For these reasons, a simplified version of the *DeepSpeech* model was derived from the Mozilla implementation. This version should (hopefully) require less training data to converge and still produce partial transcriptions that can be aligned.

2.4 A simpler model

The model from the IP8 project had some serious performance issues and did also not produce transcripts that were even remotely recognizeable as human language. It was therefore not usable and is further referred to as *previous model*. In the course of this project, it was examined more closely to find out what works best and to help the model converge. A few changes were made to arrive at a new model which was able to learn something meaningful. This model is further referred to as *new model*. The new model started to infer transcripts that – although far from perfect – resembled the ground truth.

2.4.1 Differences between the IP8- and the IP9-model

The following list summarizes the differences between the previous and the new model:

- **Optimizer**: The new model uses Stochastic Gradient Descent (SGD), whereas the previous model used Adam. Adam was used in the previous model because it works very well in various circumstances. It is also the Optimizer used in the Mozilla implementation of Deep Speech (DS). This Optimizer prevents getting stuck at local optima or saddle points by computing adaptive learning rates. It does so by keeping a history of exponentially decaying average of past gradients. However, this optimizer did not seem to work for the simplified model. Despite trying out various values for the parameters, I could not find a working combination. SGD on the other hand worked out-of-the-box with default parameter values from Keras. Because of time constraings, I decided to stick with SGD, at the risk of missing the optimal parameters.
- **number of features**: The previous model used 13 MFCC as features. This number is often found in research papers about acoustic modelling. The Mozilla implementation of *DeepSpeech* however doubled this number to 26. The new model uses the same number of features. Despite the sharp increase, this value is still much smaller than the 160 filter banks used in the original

DeepSpeech model. The amount of training data is therefore still expected to be smaller than in the original model.

2.4.2 Differences between the simplified and the reference model

The new model is a simplified variant of the Mozilla implementation of the *DeepSpeech* model with the following simplifications and changes applied:

- **Different use of LM**: The likelihood *score* of a sequence of words is calculated by an LM. This score is tightly integrated with the training process in the Mozilla implementation, providing adjustable hyperparameters to weigh the score or the number of valid words. The simplified model also uses a LM, but does not use such hyperparameters. In fact, the LM is not included in the training process. Instead, the LM is applied in some sort of post-processing to improve the quality of the inferred transcriptions a posteriori (see next chapter).
- **No convolution in first layer**: Whereas **ctc˙paper** propose a convolution over time in the first layer for performance reasons, this is not done in the simplified model.
- **LSTM instead of SimpleRNN**: Whereas **ctc˙paper** deliberately refrain from using Long Short Term Memory (LSTM) cells for various reasons, the Mozilla implementation has shown that it is possible to implement the *DeepSpeech* model using LSTM cells. Since the simplified model is based on the Mozilla implementation, it also uses LSTM cells.
- **dynamic alphabet**: The Mozilla implementation uses an alphabet with 29 characters[7], which is also what is proposed in the *DeepSpeech* paper. This is due to the fact that apostrophes are frequently found in English word tokens (like *"don't"* or *"isn't"*). The apostrophe is therefore an integral part of English words, but not for other languages. Vice versa, other languages may use a different alphabet (like German, where umlauts are prevalent). Because the number of characters in the alphabet determines the number of nodes in the output layer, the output layer has different shapes for different languages.
- **no context**: The *DeepSpech* paper proposes using combining each feature vector x_t (a frame in the spectrogram) with $C \in \{5, 7, 9\}$ context frames. This context frame was dropped to keep the nuber of features in the input layer small. As a result, the first layer in the model only depends on the 26 features of the feature vector x_t.
- **no convolution in input layer**: The *DeepSpeech* paper proposes a series of optimization to reduce computational cost. Among these optimization is a convolution over time in the input layer with by striding with step size 2. Because the context frame was dropped in this project, the striding was also not applied in order not to lose the information from the intermediate frames.

Figure 1 shows the architecture proposed in the *DeepSpeech* with the changes applied for this project. It looks similar to the one shown in the paper. Note the missing context frame, the use of MFCC features and LSTM cells in the recurrent layer.

2.5 Summary

This chapter described how a simplified variant of the *DeepSpeech* model was derived from the Mozilla implementation. It also described the changes applied to the model from the IP8 project to help the model converge during training.

[7] $a, b, c, ..., z, space, apostrophe, blank$

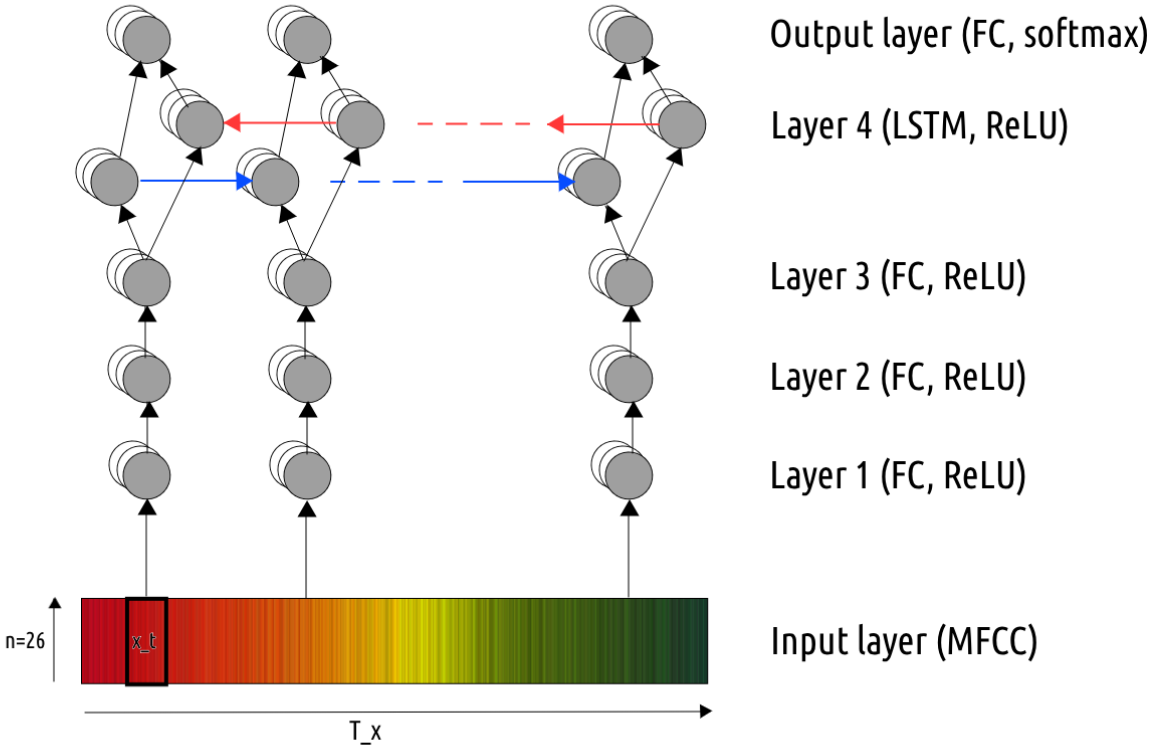

Figure 1 – Architecture of the simplified model. The cell type and the activation function is indicated in brackets for each layer (FC=Fully-Connected, ReLU=Rectified Linear Unit)

3 Integrating a Language Model

This chapter outlines the importance of LMs for ASR. It also describes how a LM was integrated into the simplified DS model as an attempt to improve the quality of the transcription.

3.1 Measuring and improving the performance of a Speech-To-Text engine

Although CTC is the cost that is optimized during training, the usual metrics to evaluate an STT system are WER and LER. These metrics corellate directly with the perceived quality of the system: Transcripts with a low WER and/or LER have a high similarity to the actual transcript and are considered accurate.

The LER (sometimes also referred to as *Levensthein Distance*) is defined as the mean normalized edit distance $ed(a, b)$ between two strings a and b. It operates on character level by counting the number of insertions (I), deletions (D) and substitutions (S) required to produce string a from string b. String a is the reference string, which in this project is the actual transcription of a speech segment (*ground truth* or *label*). String b is a an inferred transcription produced by the simplified model (*prediction*).

The WER builds upon the LER and is therefore very similar. In contrast to LER however, WER operates on word level, i.e. it represents the number of words that need to be inserted, deleted or changed in a inferred transcription in order to arrive at the ground truth.

Both metrics can be normalized by dividing them by the length of the reference string i.e. the number of characters (LER) resp. the number of words (WER). If a single evaluation metric is required, the WER is often the better choice because it is more related to the way humans assess the quality of a STT engine: A transcription that might sound correct when read out loud, but is full of spelling mistakes, is not considered a good transcription.

3.2 Language Models in Speech Recognition

LMs model the probabilities of token sequences. Because a sentence is a sequence of word-tokens, a LM can calculate its likelihood. Traditionally n-gram models (often simply referred to as *n-grams*) have been used for this task. n-grams are overlapping tuples of words whose probability can be approximated by training on massive text corpora. A special token `<unk>` is used for unknown tokens that do not appear in the training corpus. Because of combinatorial explosion and the dynamic nature of human language, the computational power and storage which are needed to train higher-order models increases exponentially with the order n of the model. Most models are trained on an order of $n = 5$ or $n = 6$.

Because the context of n-gram models is determined by their order they are somewhat restricted in that they do not take into account words outside the context to assess the probability of a sentence. Although a lot of research has been made in the field of using NN for language modelling (like for machine translation), n-grams LM are still widely used and often a good choice for many tasks (**slp3**). Because of their simplicity they are often faster to train and require significantly less training data than their neural counterparts.

3.3 A simple spell checker

The Mozilla implementation includes a 5-gram LM, which can be downloaded as a part of the pre-trained model from GitHub [8]. This LM was trained using *KenLM*. The LM is queried during training by decoding the numeric matrices produced by CTC using *Beam Search* or *Best-Path* decoding. It uses a *trie* and precompiled custom implementations of *TensorFlow*-operations written in C to maximize performance.

As mentioned above, the LM is deeply baked in with the training process of the Mozilla implementation, using its own hyperparameters. According to **mozillajourney** this tight integration is the culmination of various attempts to integrate a LM into the inference process. An early attempt used the LM as some sort of spell checker that was able to correct minor orthographic errors. Rather than including the LM-score during training, a spell-checker post-processes the inferences made by CTC *after* training. On one hand this deteriorates quality as has been shown by Mozilla, because no information from the LM is used during training. On the other post-processing the inferences is simpler and reduces complexity. This supports the project premises of a preferably simple model. It can also be implemented with the standard tools provided by Keras and does not need to be precompiled into C. Post-processing the inferences with a spell checker was therefore the approach chosen for the simplified model.

The functionality of the spell checker can be summarized as follows (a more detailed and formal description can be found in the appendix):

1. Given an inference of space-separated word-tokens, process the words from left to right.
2. For each word check if it is contained in the vocabulary of the LM.
   (a) If that is the case, continue with the next word.

[8] `https://github.com/mozilla/DeepSpeech#getting-the-pre-trained-model`

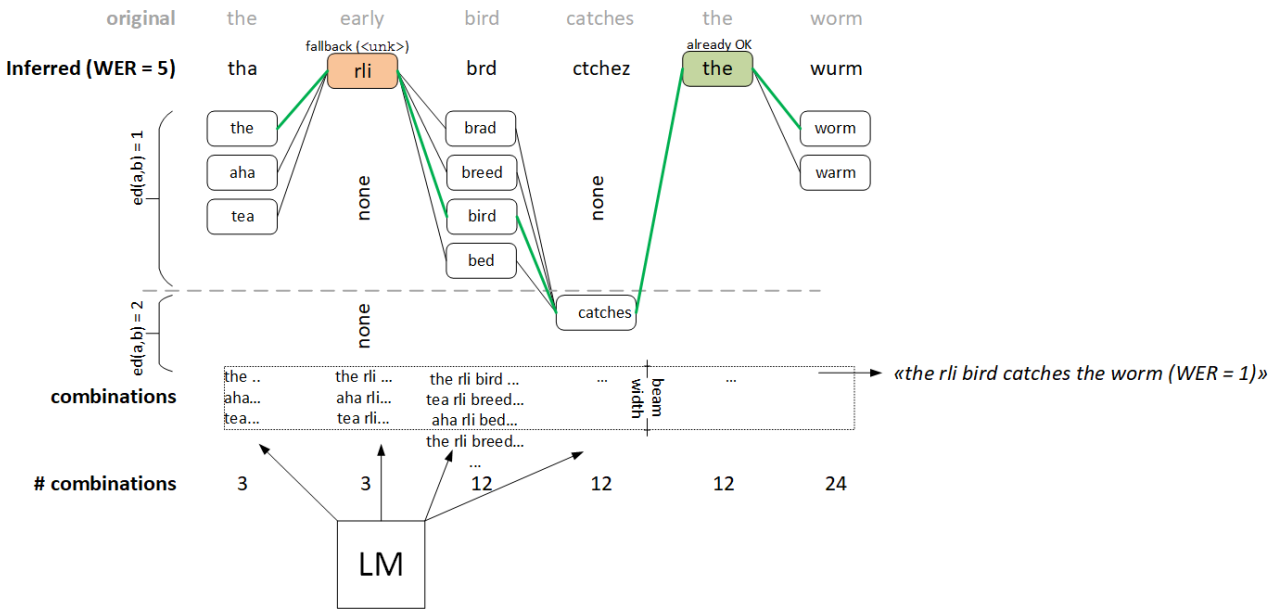

Figure 2 – Example of how the spell checker works

(b) If not, create a list of variations with edit distance 1 and keep only those variations that appear in the vocabulary. Each of these variations is a possible continuation that can be scored by the LM.

3. If none of the variations appear in the vocabulary, create another list of variations with edit distance 2 to the original word. This list can be created recursively from the (complete) list of variations with edit distance 2. Again keep only those variations that appear in the vocabulary.

4. If none of the variations of the word with edit distance 2 are found in the vocabulary, use the original word as fallback. This can happen if the word is just gibberish or if the word is an actual valid word which does not appear in the training corpus for the LM and has therefore never been seen before. Note that in this step the word must not be substituted by the `<unk>` token because it may still be a valid word. Furthermore, replacing the word with the `<unk>` token can have a contrary effect on the alignment, because this token will most likely never appear in a valid transcript.

Above steps are repeated until the whole sentence is processed. For each word this yields a cascade of possible combinations. Each of these combinations can be scored by the LM as the sentence is being processed whereas only the b most likely prefixes are kept at each step (beam search). For this project, a beam width of $b = 1.024$ was used. Figure 2 illustrates how the spell-checker works.

A spell checker in combination with a LM can help inferring orthographically correct words from sequences of characters inferred by CTC and hence decrease the WER. Therefore, by using a LM the quality of transcriptions can improve considerably. Table 1 illustrates this with an example.

3.3.1 Reducing the vocabulary size

The LM from Mozilla was trained on texts from the *LibriSpeech* corpus[9]. Apart from lowercasing, the texts were not normalized or preprocessed. The resulting vocabulary is therefore very big and contains 973.673 unique words. Because no further preprocessing was done, it also contains some exotic words like *"zzzz"* and probably also misspelled words that happen to appear in the corpus. To train the LM,

[9] `http://www.openslr.org/11`

		LER	LER (norm.)	WER	WER (norm.)
ground truth	i put the vice president in charge of mission control	0	0.00	0	0.00
before spell-checking	ii put he bice president in charge mission control	6	0.11	4	0.40
after spell-checking	i put the vice president in charge mission control	3	0.06	1	0.10

Table 1 – Example for how a Spell-Checker (SC) can help improve the quality of an inferred transcription by changing characters and words. Audio and ground truth were taken from the ReadyLingua corpus and the inference was made with the pre-trained DeepSpeech model.

n-grams of order 4 and 5 were pruned with a threshold value of 1, meaning only 4- and 5-grams with a minimum count of 2 and higher are estimated[10]. Because most spelling errors are probably unique within the training corpus, 4- or 5-grams containing an invalid word are unique too and most likely filtered out with pruning.

Above procedure might work well to estimate the likelihood of a sentence. For a simple spell checker however, such a big vocabulary might be counter-productive because it lowers the probability that an obviously wrong word is corrected because for some reason it found its way into the vocabulary. Vice versa a very large vocabulary raises the probability that a random-looking sequence of characters is wrongfully exchanged with a (valid or invalid) word from the vocabulary. To prevent this, the original vocabulary was reduced into three vocabularies containing the 40.000, 80.000 and 160.000 most frequent words from the corpus each. These words make up 98.42%, 99.29% and 99.69% of the corpus.

To create the vocabularies, a list of unique words and their frequency was created from the corpus and sorted by frequency in descending order. Naturally, stop words like *the*, *and* or *of* appear at the top of the list. The first 40.000, 80.000 resp. 160.000 words from this list were stored as the truncated vocabularies, the rest was discarded. Note that truncating the vocabulary only affects the way words are exchanged by the spell checker during post-processing, not how the likelihood of a post-processed sentence is estimated by the LM.

3.4 Further thoughts and considerations

The spell checker in this project uses the vocabulary with 80.000 words. This value was arbitrarily chosen and some unsystematic experiments were made to analyze the correctional capabilities of the spell checker. Because of time constraints and because it was unclear whether the spell checker would help improving the transcriptions in the first place, other vocabulary sizes were not evaluated. Further work may however try to find out an optimal vocabulary size for each language.

3.5 Summary

This chapter described how the 5-gram LM from the Mozilla implementation of *DeepSpeech* was used to implement a rudimentary spell checker. This spell checker uses a vocabulary of the 80.000 most frequent words from the corpus the LM was trained on. It repeatedly swaps invalid words from an inferred transcript with valid words from the vocabulary and calculates the likelihood of various combinations of word sequences.

[10] see `https://github.com/mozilla/DeepSpeech/tree/master/data/lm`

4 Plotting a learning curve

This section describes how training progress was monitored by plotting a learning curve.

4.1 Previous corpora and their problems

The following two corpora were available for training from the IP8 project.

- **LibriSpeech (LS)**: This corpus was created as an artifact of the IP8 project using raw data from OpenSLR. The raw is publicly available and can be downloaded[11]. It consists of a number of audio files which were *partially* transcribed, i.e. there are parts in the audio for which the corresponding transcript is not exactly known (the audio contains *gaps*). The individual samples were obtained by exploiting metadata that was included in the download. The metadata includes a split into a training set (containing approximately 96% of the samples) and a validation resp. test set (each containing approximately 3% of the samples). The split was done manually into disjoint subset, i.e. ensuring each speaker was only included in one set. Additionally, other features like gender or accent were observed to achieve a similar distribution for each set. To leverage the efforts made by *OpenSLR*, this split was not changed.
- **RL**: This corpus was created from raw data provided by *ReadyLingua*. This data is proprietary and contains recordings in several languages which were manually aligned with their transcript. In contrast to the LS corpus, the raw data is fully aligned, i.e. there are no gaps in the audio. However, the metadata does not comprise a split into training-, validation- and test-set. Since the raw data contained recordings and transcripts in more than one language, separate splits were made for each language preserving a ratio of approximately 80/10/10% (relating to the total length of all recordings within each subset). Efforts were made to prevent samples from the same recording being assigned to different subsets. Other features were not observed, meaning the split into train/validation/test-set was done less carefully than in the LS corpus.

The model in the IP8 project was supposed to be trained on the LS corpus, because this corpus is much larger than the RL corpus. In the course of the project it became clear however that training on all samples from this corpus was not feasible within project time because training time would have taken more than two months. It also turned out that the LS corpus was probably less useful than initially assumed because the average sample length was much longer than the samples in the RL corpus. This made training even harder because convergence is much slower when training on long sequences. The RL corpus on the other hand consisted of shorter samples, but the total length of all samples was only a few hours compared to the 1000+ hours in the LS corpus.

4.2 The *CommonVoice* Corpus

Because of the aforementioned problems a new corpus was needed which combined the best of both worlds:

- it should contain a reasonable amount of speech samples to facilitate training an ASR model
- the average sample length should be short enough for the model to learn quickly.

The CommonVoice (CV)[12] corpus is maintained and used actively by the Mozilla Foundation and exhibits both of these properties. This corpus is also used to train the Mozilla implementation of *DeepSpeech*. Datasets for various languages are being prepared and verified, each one containing speech samples of different contributors from all over the world. At the time of this writing, only the English dataset was available, but datasets for other languages will become publicly available at

[11] `http://www.openslr.org/12/`
[12] `https://voice.mozilla.org/en/data`

some time in the future. The English dataset comes pre-divided into training-, validation- and test-set of similar scale like the LS corpus. Each set consists of one audio file per sample and a CSV file containing the transcriptions for each sample.

The simplified model in this project was trained on the training-set of the CV corpus. Although still smaller than than the LS corpus, the total length of all validated samples that can be used for training[13] is much larger than the RL corpus while providing samples of similar length at the same time. Table 2 shows some statistics about the corpora described above.

Corpus	Language	total audio length	train/dev/test	# samples	Ø sample length (s)	Ø transcript length (chars)
LS	English	$24days, 7:13:18$	93.51/3.32/3.16%	166,510	12.60	183.84
RL	English	$5:38:39$	80.39/10.13/9.48%	6,334	3.20	51.81
RL	German	$1:58:30$	81.14/10.26/8.60%	2,397	2.89	45.55
CV	English	$10days, 1:02:53$	96.04/1.99/1.98%	201,252	4.31	48.07

Table 2 – Statistics about corpora that were available for training.

4.3 Plotting the learning curve

The time needed to train an ASR model on all samples of the CV corpus is still too long for the available project time. We can however still get an estimate of the learning progress by plotting a *learning curve*. For this, exponentially increasing amounts of training data (1, 10, 100 and 1,000 minutes of transcribed audio) were used. Training was done making 30 full passes over the training set (*epochs*). The training samples were processed in batches of 16 samples each. They were sorted by the length of their audio signal and then zero-padded[14], yielding samples of the same length in each batch[15].

After each epoch, the progress was monitored by inferring the transcriptions for previously unseen samples from the validation set. The CTC-loss for training and validation was plotted for each amount, yielding separate curves for the training- and the validation-loss. Comparing both curves allows for making statements about at what point the Neural Network starts to overfit.

Complementary to the CTC-loss, the mean values for the LER and WER metric over all samples in the validation set was calculated after each epoch, yielding the curves for the LER resp. WER. Observing these plots can give some insight about how well the network performs on unseen examples.

Both loss and metrics were compared along two dimensions:

- **The decoder dimension**, comparing the two distinct ways to decode a transcript from the probability distributions calculated by the model for each frame in the input signal
- **The LM dimension**, comparing inferences made with and without post-processing the decoded transcript with a spell-checker as described above

Both dimensions are described in more detail below.

4.3.1 Decoder dimension

In a nutshell, CTC aligns the T_y characters from a known transcription (*label* or *ground truth*) with the T_x frames from the input audio signal during training. T_x is typically much larger than T_y and

[13] CSV file: `cv-valid-train.csv`
[14] sorting the samples was done to have sequences of similar length in each batch and thus reduce padding
[15] note that the length of the samples could still vary between batches

must not be shorter. The characters (*tokens*) in the label must come from an alphabet of size V, which for English are the 26 lowercased ASCII characters *a..z*, the space character and the apostrophe (because this character is very common in contracted words like e.g. *"don't"* or *"isn't"*). Additionally, CTC introduces a special token ϵ, called the *blank token*, which can be used to label unknown/silent frames or prevent collapsing (see below). Consequently, the number of characters in the alphabet used by the ASR in this project to recognize English is $|V| = 26 + 1 + 1 + 1 = 29$.

CTC is *alignment-free*, i.e. it does not require an alignment between the characters of a transcription and the frames of an audio signal. The only thing needed is the audio signal X itself plus its ground truth Y. Each token in the ground truth can be aligned with any number of frames in the input signal. Vice versa, repeated sequences of the same characters can be collapsed, whereas the ϵ token acts as a boundary within sequences of a token to prevent collapsing into one, when there should be two (such as in *f-f-o-o-ϵ-o-o-o-o-d-d-d*, which should collapse to *food* and not *fod*).

For each frame input signal CTC calculates a probability distribution over the $|V|$ characters in the alphabet. This yields a $|V| \times T_x$ probability matrix for the input signal. Because $T_x \ggg T_y$, there is usually a vast amount of different valid alignments collapsing to the same ground truth. The probability of each valid alignment can now simply be calculated by traversing the probability matrix from left to right and multiplying the probabilities of each character. Because calculating the probability of each valid alignment individually would be too slow and identical prefixes between valid alignments yield identical probabilities, a dynamic programming approach is usually chosen to calculate the probabilities whereas the intermediate probability for each prefix is saved once computed.

The most probable alignment is calculated by marginalizing (i.e. summing up) over the probabilities of the individual valid alignments. This calculation yields the CTC loss as a sum of products, which is differentiable and can therefore be optimized.

After training, a model using CTC will again output a $|V| \times T_x$ probability matrix for any previously unseen input. This matrix can be used to infer a transcription, a process also known as *decoding*. The CTC paper proposes two different decoding strategies that are applied before collapsing the characters **ctc˙paper**:

- **Best-Path (a.k.a. *greedy*) decoding**: This strategy only ever considers the most likely character at each time step. The transcription before collapsing will be a single path through the the probability matrix, whose probability will be the product of all elements along the path. This approach is easy to implement but does not take into account the fact that a single output can have many alignments, whose individual probability may be lower than the one found with this strategy.
- **Beam-Search decoding**: This strategy approximates the probability of the most probable transcription by following multiple paths simultaneously and only keeping the B most probable paths at each time step. The beam width is a hyperparameter that can be increased to get a more accurate transcription in exchange for higher computational cost.

Beam-Search decoding is expected to perform better. For the sake of completeness, both decoding strategies were compared in this project. This will yield separate learning curves for the decoder dimension. For Beam-Search decoding, the Keras implementation was used, which proposes a default beam width of $B = 100$. This value was not changed.

4.3.2 LM dimension

Using a LM to post-process the inferred transcription with a rudimentary spell checker will not necessarily lead to more accurate transcription, especially if the edit distance between prediction and

ground truth is large. Table 3 contains an example where the use of a spell checker is disadvantageous to the quality of a transcription.

		LER
ground truth	i want to wish you a very happy thanksgiving	
prediction before spell-checking	oento wiceyouepery appy thangksive	0.4318
prediction after spell-checking	onto wiceyouepery app thangksive	0.4545

Table 3 – Example of a transcription whose LER was increased when using a spell checker

In this example, ≪ *oento* ≫ was changed to ≪ *onto* ≫ because this was the most probable word with a maximum edit distance of 2 that was in the vocabulary. Similarly, ≪ *appy* ≫ was changed to ≪ *app* ≫. This lead to a orthographically better sentence, but the LER is higher than without spell-checking.

It is generally expected that post-processing the inference as described above will lead to a lower WER, supposed the LER is already low enough, i.e. the prediction matches the ground truth already pretty well. If the LER value is too high, the spell checker might try too hard to find a word from the vocabulary. This might result in a changed sentence consisting of real words but whose similarity to the ground truth is lower than before the changes. Post-processing might then be counter-productive. Therefore, separate learning curves were plotted for inference with and without post-processing (the *LM dimension*)

4.4 Results and interpretation

Figure 3 shows the learning curve for the CTC-loss. Obviously the training loss decreases steadily for all amounts of training data, converging to values between 30 (training on 1.000 minutes) and 50 (training on 1 minute). Because the network does not generalize well when being trained on only 1 or 10 minutes of audio data, its predictions are somewhat random. This may be an explanation for the jagged curve of the validation plot (dashed lines) for these amounts of training data. When training on 100 minutes, the plot for the validation loss is smoother, but starts to increase between epoch 10 and 15, meaning the network starts to overfit after that point. When training on 1.000 minutes the validation loss does not decrease after epoch 14 anymore and plateaus at a value of about 90, meaning that any training will not contribute to a more generalizable network.

Figure 4 shows how the average values of the LER over all validation samples develops for the different amounts of training data. The plot on the left shows the results when using best-path decoding, the plot on the right for beam search decoding. The plots for all amounts of training data have been integrated in the same plot for the sake of a clearer representation. Both plots support the conclusions made for the CTC loss in that – except when training on 1.000 minutes of audio – the error rates do not decrease after epoch 15 anymore (in fact there is a slight increase). The plots for the LER when training on 1.000 minutes are almost identical for both decoding strategies. Surprisingly, the LER values continue to decline steadily, although only at a very slow rate, finishing with values of 0.54 (best-path decoding) resp. 0.52 (beam search decoding), meaning that the network got a bit more than half of the characters wrong, at the wrong position or entirely failed to predict them. The values when trying to correct the inferred transcriptions with a spell checker are slightly higher for both decoding strategies (0.55 and 0.53) meaning that post-processing did not help. This nourishes the assumption that a spell checker will probably only lower the WER and only do so if the LER is already low enough (see above).

Finally, figure 5 shows the development of the average WER values over all validation samples. Not surprisingly, the plots oscillate around a value of 1, meaning the network did not get any of the words right. Only when training on 1.000 minutes, the network was able to achieve a value below 1, but is still

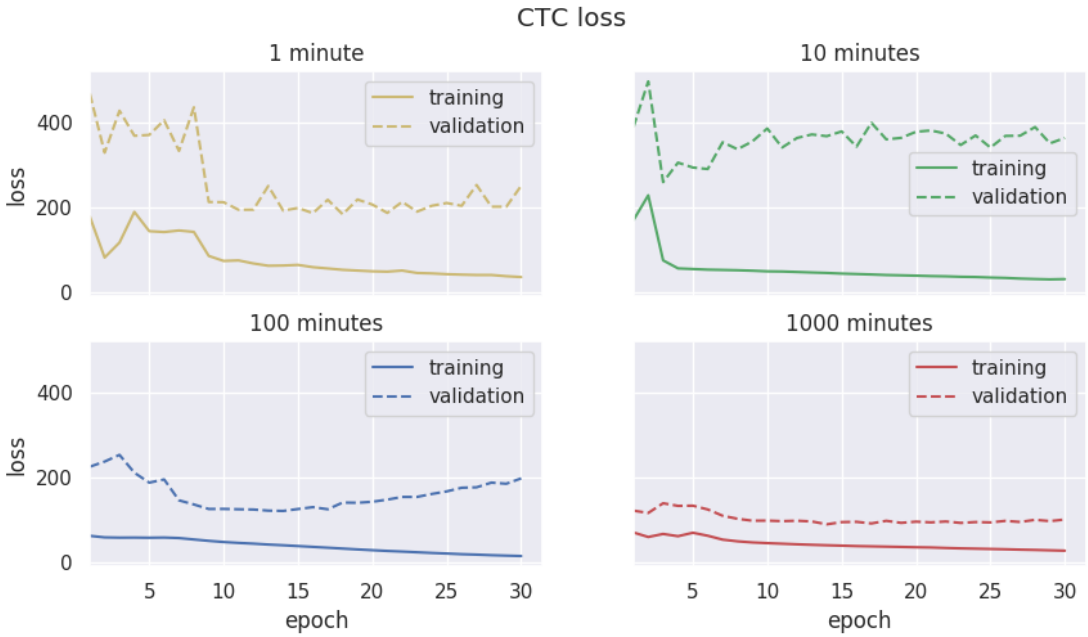

Figure 3 – Learning curve for the CTC-loss while training on 1/10/100/1000 minutes of transcribed audio from the CV corpus using the 5-gram LM provided by the Mozilla implementation of DeepSpeech

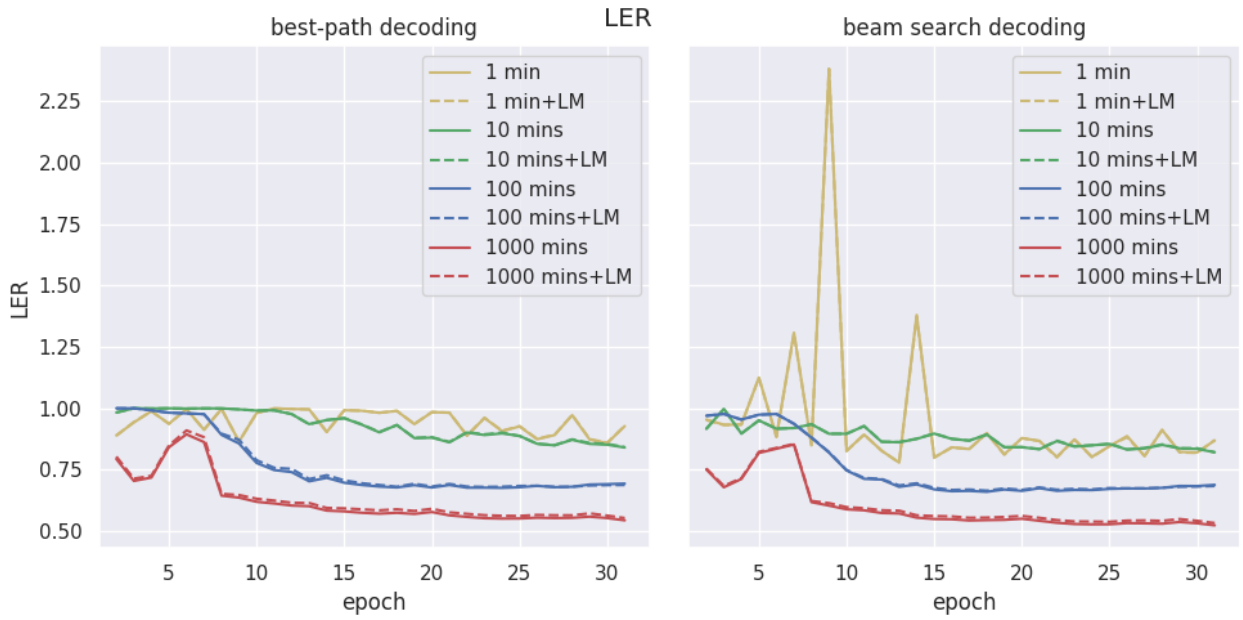

Figure 4 – Learning curve for the LER metric while training on 1/10/100/1000 minutes of transcribed audio from the CV corpus with and without spelling correction with a LM. For the lines where spelling was corrected, the 5-gram LM provided by the Mozilla implementation of DeepSpeech was used.

way higher than the 0.0065 achieved by the Mozilla implementation of *DeepSpeech*. It is noteworthy that the use of a spell checker marginally improves the results here.

In summary it can be said that the lowest LER rates can be achieved when training on 1.000 minutes of audio, not using the spell-checker. Training can be stopped after about 15 epochs however, because the results on validation data does not improve from there on. The average LER of 0.52 suggests that

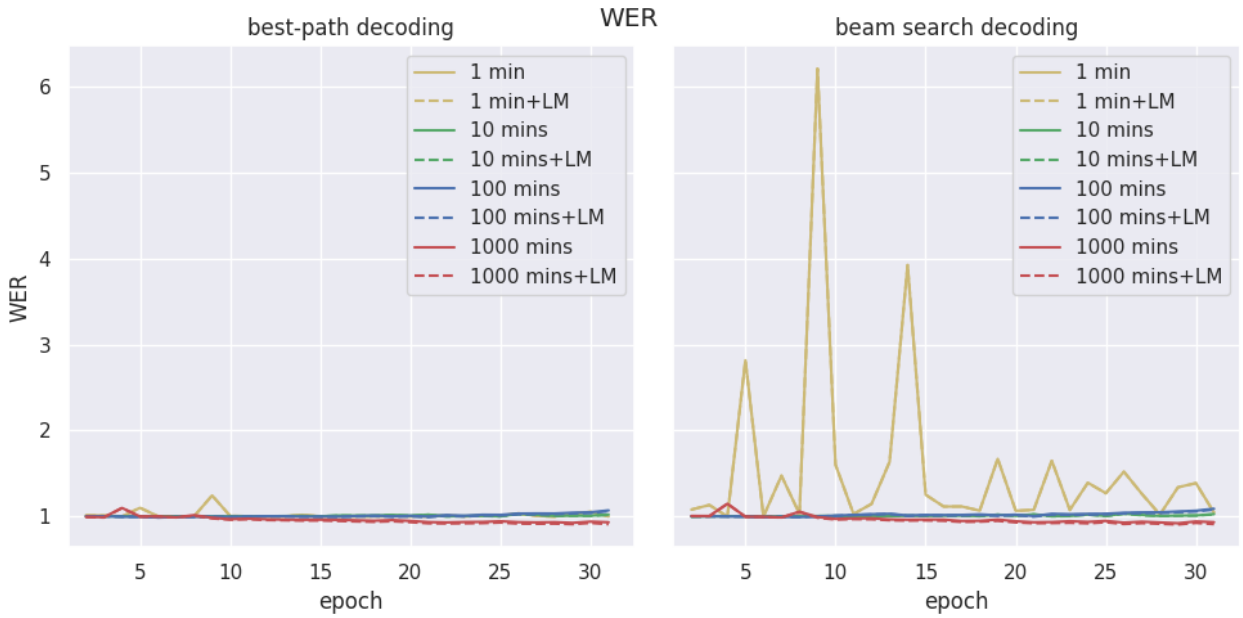

Figure 5 – Learning curve for the WER metric while training on 1/10/100/1000 minutes of transcribed audio from the CV corpus with and without spelling correction with a LM. For the lines where spelling was corrected, the 5-gram LM provided by the Mozilla implementation of DeepSpeech was used.

the network predicts about half of the characters right. This yields transcriptions which – although far from correct – are recognizeable as human language. Sometimes the real transcript might be guessed, especially for shorter sentences. Table 4 shows an example for how the accuracy of the inferred transcript improves with additional training data.

training data	inferred transcript	LER
1 minute	w t e isi	0.76
10 minutes	tiar n th i id	0.71
100 minutes	ave gos goe ei	0.52
1.000 minutes	i've got go fi ha	0.33

Table 4 – Example of how the quality for inferred transcripts improves with additional training data. The LER values were calculated against the ground truth « i've got to go to him »

4.5 Regularization

As an attempt to prevent overfitting (or at least postpone it to later epochs), the network has been regularized by adding dropouts after each layer. The rate of each dropout has been set to 0.1, meaning a random 10% of the unit weights in each layer will be zeroed out. Apart from adding dropouts no further changes were made to the simplified model.

The learning curve for the model with dropouts is similar to the one without dropouts, meaning its validation loss will plateau after about 15 epochs. To compare the simplified model with and without dropout, the average LER on the CV test-set was calculated with both decoding strategies. Table 5 shows the results of the comparison. The observations made for the spell checker also apply to the model with dropouts, meaning that the LER rate is slightly better without spell-checking. The lowest average LER rate (highlighted) was achieved using beam search and no spell checker. Although the

	Ø *LER*	
	not regularized	regularized
best-path decoding		
without spell-checker	0.5359	0.5343
with spell-checker	0.5475	0.5456
beam search decoding		
without spell-checker	0.5146	0.5125
with spell-checker	0.5256	0.5242

Table 5 – Comparison of the simplified model with and without dropout regularization. The average LER was calculated over all samples from the CV test-set. The best value is highlighted.

difference is only marginal, the regularized model will be used for further evaluation.

4.6 Final thoughts and considerations

Above results were achieved with a spell checker using a vocabulary of 80.000 words and the 5-gram LM from Mozilla. This did not help very much, but it might be possible that a different vocabulary size will produce better results. It is also possible that a different Optimizer, different dropout rates or integrating the LM score into the cost function (like Mozilla did) will produce better results. Finally, it might be fruitful to train on smaller batches as it has been observed by **batch˙size˙rnn** that with larger batches the quality of a model degrades.

All these ideas produces many more combinations to try out, but preparing and running them is very time consuming. Because the LER of about 0.5 (1.000 minutes, no spell checker) looks promising, I decided to leave it at this for the moment and see how far I get.

4.6.1 Summary

This chapter gave a quick introduction into how CTC and its different decoding strategies work. It also gave an overview over the available corpora and showed how the training progress developed with varying amounts of training data, different decoding strategies and the use of a spell-checker. Post-processing the inferences with a spell-checker will not always lead to better transcripts. Training was done using samples from the *CommonVoice* corpus. A regularized model was created by adding dropouts to the simplified model. The best results were obtained with the regularized model not using the spell-checker. When training this model on 1.000 minutes of data the average LER on test data is only slightly higher than 0.5, meaning the model will get about 50% of the characters in the transcript right. However it was also observed that both the regularized and unregularized model start to overfit after about 15 epochs.

5 Measuring the performance of the pipeline

Above results reflect the performance of the ASR stage alone. To get some insight about the quality of alignments produced by the whole pipeline, a simple web application was implemented that highlights the aligned parts of the transcript as the audio file is being played. This is very useful for an informal review, because the subjective quality of the alignments can be examined interactively. However, this method is not very systematic and infeasible for larger amounts of test data. To get a clearer sense of how well the pipeline performs, steps were taken to run large numbers of previously unseen samples through the pipeline and measure the quality of the final product (the alignments). This section describes how this was done.

5.1 The quality of alignments

Assessing the quality of alignments is not trivial because there is often no reference alignment to compare to. Even if there is one, assessing the quality of an alignment is somewhat subjective because a different alignment does not necessarily need to be worse or better. Objectively quantifying the quality of a result is difficult for an alignment pipeline because there is a a massive number of theoretically possible alignments for each audio/text combination. We can however derive a few objective criteria that make up a good alignment:

1. The aligned partial transcripts should not overlap each other
2. The alignments should neither start nor end within word boundaries
3. The aligned partial transcripts should cover as much of the original transcript as possible
4. The aligned partial transcripts should be at the correct position, i.e. they should cover the actually spoken text

The first criterion is enforced by changing the type of algorithm used for sequence alignment from a local to a global alignment algorithm. The *Smith-Waterman* algorithm was used in the LSA stage in the IP8 project, which finds a local optimum for each transcript in isolation. The LSA stage in this project uses global sequence alignment (*Needle-Wunsch* algorithm), which finds an optimal alignment for all partial transcrips at once.

The second criterion is ensured by adjusting some of the alignments so that they fall exactly on word boundaries. This is done by moving the alignment boundaries produced by the *Needle-Wunsch* algorithm to the left or right, depending on which one is closer.

The remaining two criteria can be quantified with the following metrics (note the corellation[16]):

criterion	metric	correlation
3	length of text in ground truth that is not aligned vs. total length of the ground truth	negative
4	average Levensthein similarity between the transcript and the text in the ground truth corresponding to its alignment	positive

Table 6 – Metrics to evaluate the quality of alignments

Because the first metric measures how much of the target transcript is covered by the alignments, it is similar to the Recall metric (R) usually used for classification tasks, which measures how much of

[16] positive correlation: higher is better, negative correlation: lower is better

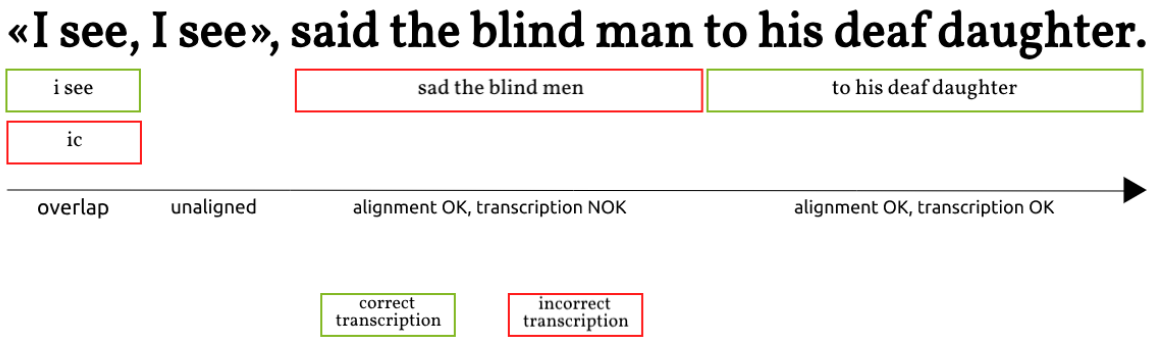

Figure 6 – Example for a partially correct alignment of parts of a sentence

the target class was correctly classified. The second metric measures how well the produced results match up with the underlying parts of the transcript and is therefore similar to Precision (P), which the correctness of the classified results. Both metrics can hence be reduced to the *F-score* (F):

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

Figure 6 shows a constructed example of how the inferred partial transcripts (predictions) are aligned with the known full transcript (ground truth) of an audio/text sample.

5.2 Test and results

The pipeline was evaluated on the test set of the *LibriSpeech* corpus containing 87 audio/text samples. Each sample was run through the pipeline twice using different STT models in the ASR stage:

1. Simplified Keras model: the dropout-regularized model trained on 1.000 minutes was used, because it had the lowest average LER value on the validation data. Training was stopped early after 15 epochs to prevent overfitting.
2. Reference model: the pre-trained model downloaded from the project page of the Mozilla implementation[17] was used

Running the samples through both pipelines should help comparing the results produced by the first pipeline against a hypothetical optimum. Apart from the model used in the ASR stage, all other stages in the pipeline were identical. The average values of P, R and F over all test samples was calculated for each pipeline. Figure 7

[17] `https://github.com/mozilla/DeepSpeech#getting-the-pre-trained-model`

Figure 7 – Average values of P, R and F for a pipeline using the simplified STT model compared to a pipeline using a state of the art model

6 Forced Alignment for other languages

So far, only audio and transcripts in English were considered. A fully automated solution however should be able to align text and audio in any other language. Because of linguistic characteristics like sound patterns and morphology the results might vary a lot between languages when tested under identical circumstances. To get some intuition about the influence of language and whether above conclusions are transferable to other languages, the pipeline was evaluated on the German samples received from *ReadyLingua*.

6.1 Inferring German transcripts

Enabling the pipeline to handle German samples means training a German ASR as its core element. This requires minimal modifications to the network architecture, because German transcripts use a different alphabet. As mentioned before, the apostrophe is far less common in German than in English and was therefore dropped. On the other hand, umlauts are very common in German and were added to the 26 ASCII characters. Since the alphabet represents all possible labels, the output layer in the model needs to be changed to contain 31 units (one for each character in the alphabet, the three umlauts, space plus a blank token) instead of the 29 units used for English.

Training an ASR model for German and plotting a learning curve also requires amounts of training data on a similar scale like the CV corpus used for English. Since at the time of this writing, the CV was still a work in progress, datasets for languages other than English were not available. High-Quality ASR corpora are generally hard to find, especially considering the number of samples needed to train a RNN. There are corpora for ASR in German, but those are either not freely available or their quality is unknown. An extensive list of German corpora for various purposes can be found

at the Bavarian Archive for Speech Signals (BAS)[18], including exotic corpora like the *ALC* corpus[19] containing recordings of intoxicated people. Some of the corpora on this list are free for scientific usage and have been used by **budget** to train their German ASR model with transfer-learning. However, not all of these corpora are targeted at ASR and their quality is often unknown.

6.2 Data augmentation

Integrating new raw data means preprocessing the audio (e.g. resampling) and the text (e.g. normalization, tokenization) to make sure it exhibits the same properties as the other corpora and the data conforms to the target format. This step is usually very time consuming, often taking most of the project time. Because no ASR corpus for German was readily available, training was done on the data received from *ReadyLingua* as a start. The alignment between audio and transcript in this corpus was done manually and is therefore very accurate. Audio and text were already preprocessed in the IP8 project and the metadata was processed and stored as a corpus. The individual training samples could therefore be transformed to the format expected by the Mozilla implementation of *DeepSpeech* (and thus by the simplified model) with comparably little effort. Also, the samples exhibited similar properties (average audio and transcript length) like the CV corpus (refer to table 2). However, the total length of the samples in the training set was only about one and a half hours, which was much less than the 1000+ minutes in the CV corpus and certainly not enough for the 1.000 minutes needed to plot a learning curve like to the one made for English.

An easy way to get more training data is augmenting existing data by synthesizing new data from it. This is particularly easy for audio data, which can be distorted in order to get new samples corresponding to the same transcript. The following distortions were applied in isolation to each sample in the training set:

- **Shifting**: The frames in the input signal were zero-padded with samples corresponding to a random value between 0.5 and 1.5 seconds, shifting the signal to the right, i.e. starting the signal later. This resulted in one additional synthesized sample for each original sample. Shifting to the left was not done to prevent cropping parts of the speech signal.
- **Echo**: The presence of echo can be generated with the Pysndfx library[20] using random values for delay and damping. This resulted in one additional sample.
- **Pitch**: The pitch of the signal was increased or decreased. Increasing and decreasing was done using two different random factors, resulting in two additional samples. This can be seen as a rudimentary way to simulate a female from a male speaker or vice versa.
- **Speed**: Faster or slower speaking rates can be simulated by "stretching" or "compressing" the signal while preserving the pitch. Similar to the change in pitch, two different random factors were use to change the tempo. This resulted in two additional samples.
- **Volume**: The loudness of the speaker was artificially reduced or increased by a random value within the range of $[-15..-5]$ resp. $[5..15]$ db. This resulted in two additional sample.

With above methods eight synthetisized samples can be created for each original sample from the corpus. It turned out however that this was still not enough to plot a learning curve. To augment the data to the 1.000 minutes needed, additional samples were created using random combinations of the distortions. The random parameters differed from the ones used before to prevent overfitting to the distortion. Table 7 shows the corpus statistics before and after data augmentation.

[18] `https://www.phonetik.uni-muenchen.de/Bas/BasKorporaeng.html`
[19] `http://www.bas.uni-muenchen.de/forschung/Bas/BasALCeng.html`
[20] `https://github.com/carlthome/python-audio-effects`

	total audio length	# samples	Ø sample length (seconds)
before augmentation	1 : 36 : 09	1,700	2.89
after augmentation	16 : 40 : 00	18,955	3.16

Table 7 – Comparison of RL corpus before and after data augmentation (training set only)

6.3 Creating a Language Model for German

Since the ASR stage in the pipeline uses a spell-checker querying a LM to post-process the results a 5-gram model similar to the one created by Mozilla needed to be trained first.

6.4 n-Gram Language Models

To understand the following sections better it might be helpful to get a quick recap about LM. LM are probabilistic models that model the likelihood of a given sequence of characters or words. The most widely used type for word-based models LMs are n-gram LM. However, such models can estimate probabilities only for words that appear in the vocabulary of the corpus they were trained on. All other words are Out Of Vocabulary (OOV) words with a probability of 0. The probability of a sentence can be computed from the probabilities of each word (1-grams) with given all its preceding words in the sentence using conditional probability. Getting statistically relevant high numbers for each combination of words requires huge text corpora. However, language is dynamic and new sentences can be created all the time so that no corpus would be big enough. To handle this, n-grams approximate the probability of a combination of words by only considering the history of the last n words (n denoting the order). However, above problem is still valid for n-grams of any order: Because of combinatorial explosion n-grams suffer from sparsity with increasing order.

6.4.1 Perplexity, discount and smoothing

To evaluate an n-gram LM a metric called *perplexity* is usually used, which is the normalized inverse probability on a test set. The perplexity can be interpreted as the grade to which the LM is "confused" by a certain n-gram. A high perplexity therefore corresponds to a low probability. Since the perplexity carries the probability of a certain n-gram in the denominator, the perplexity for OOV-n-grams cannot be calculated (division by zero). To handle this efficiently, a technique called *smoothing* is applied. A very rudimentary form of smoothing is *Laplace Smoothing*, which assigns a minimal count of 1 to every n-gram. All other counts are also increased by adding 1. This prevents counts of zero for n-grams that do not appear in the training corpus. Smoothing therefore shaves off a bit of the probability mass from the known n-grams and moves it to the unknown n-grams. The factor with which the probability of a known n-gram is reduced is called *discount*.

6.4.2 Kneser-Ney Smoothing

Although with Laplace Smoothing a low probability is assigned to previously unseen n-grams (which results in a high perplexity), it performs poorly in application because it discounts frequent n-grams too much (i.e. gives too much probability to unknown n-grams). A better way of smoothing is achieved using *Kneser-Ney Smoothing*. For unseen n-grams, *Kneser-Ney Smoothing* estimates the probability of a particular word w being the continuation of a context based on the number of context it has appeared in the training corpus. For any previously unseen n-gram, a word that appears in only few contexts (e.g. the word *Kong*, which only follows the words *King* or *Hong* in most corpora) will yield a lower probability than a word that has appeared in many contexts, even if the word itself may be very frequent. The intuition behind this is that such a word is assumed less likely to be the novel

continuation for any new n-gram than a word that has already proved to be the continuation of many n-grams.

6.5 Creating a raw text corpus

To train a n-gram model for German, a raw text corpus of German Wikipedia articles was used as corpus. Like the English n-gram from Mozilla KenLM (**kenlm**) was used to estimate the probabilities. The articles were pre-processed to meet the requirements of *KenLM*. It was normalized as follows

- remove Wiki markup
- remove punctuation
- make everything lowercase
- **Unidecoding**: translate accentuated characters (like `è,é,ê`, etc.) and special characters (like the German ß) to their most similar ASCII-equivalent (`e` resp. `ss`). This process helps accounting for ambiguous spelling variants of the same word and misspelled words. It also reduces the number of unique words by reducing different versions to a normalized variant. A special case are umlauts. Although also not part of the ASCII code set, they were kept as-is because they are very common in German.
- **Tokenization**: Because *KenLM* expects the input as sentences (one sentence per line), the raw text was further tokenized into sentences and words using NLTK (**nltk**).
- **Numeric tokens**: Word tokens that are purely numeric (such as year numbers) are replaced with the special token `<num>`. Although such tokens occur frequently in the Wikipedia articles, they are unwanted in the corpus because they represent values and do not carry any semantic meaning. Because there is a infinite number of possible numeric tokens, they were all collapsed to the same normalized token.

The corpus was saved as text file containing one normalized sentence per line. The special tokens `<s>` and `</s>` are used to mark beginnings and endings of sentences as well as the `<unk>` token which is traditionally used to represent OOV words. They are however not part of the corpus because they are added automatically by *KenLM*.

The following lines are an excerpt of a article in the German Wikipedia along with its representation in the corpus.

> Die Größe des Wörterbuchs hängt stark von der Sprache ab. Zum einen haben durchschnittliche deutschsprachige Sprecher mit circa 4000 Wörtern einen deutlich größeren Wortschatz als englischsprachige mit rund 800 Wörtern. Außerdem ergeben sich durch die Flexion in der deutschen Sprache in etwa zehnmal so viele Wortformen, wie in der englischen Sprache, wo nur viermal so viele Wortformen entstehen. (German Wikipedia article about Speech Recognition[21])

```
1 die grösse des wörterbuchs hängt stark von der sprache ab
2 zum einen haben durchschnittliche deutschsprachige sprecher mit circa <num> wörtern
    einen deutlich grösseren wortschatz als englischsprachige mit rund <num> wörtern
3 ausserdem ergeben sich durch die flexion in der deutschen sprache in etwa zehnmal so
    viele wortformen wie in der englischen sprache wo nur viermal so viele wortformen
     entstehen
```

Listing 1 – Representation in corpus

[21] `https://de.wikipedia.org/wiki/Spracherkennung`

Like for the English spell checker, three vocabularies containing the 40.000, 80.000 and 120.000 most frequent words from the corpus was created. The words from these vocabularies make up 87.75%, 90.86% resp. 93.36% of the total number of words in the corpus. It is expected that the optimal number of words in the vocabulary is higher for German than for English. This is due to the fact that different flexions of the same word are very common in German due to grammatical conjugations (different forms for the same verb) and declinations (different cases for the same noun). Therefore German tends to apply a wider range of words and the size of vocabulary had to be increased. Handling the different flexions would require lemmatization and/or stemming the corpus in order to reduce them to a common base form. This has not been done for simplicity and time constraints. It is also doubtful whether this would actually help improving the quality of inferred transcripts, since humans do not speak in lemmata or stems.

6.6 Training the LM

The final corpus contained data from 2,221,101 Wikipedia articles (42,229,452 sentences, 712,167,726 words, 8,341,157 unique words). This corpus was used to train a 5-gram LM using *KenLM*. *KenLM* uses *Kneser-Ney Smoothing* and some optimization techniques called *quantization* and *pointer compression*.

6.6.1 Data structures

n-grams can be represented with a prefix-tree structure (called *Trie*) [22], which allows for pruning. n-grams of order 2 and higher can be pruned by setting a threshold value for each order. n-grams whose frequency is below the threshold will be discarded. *KenLM* does not support unigram pruning.

6.6.2 Quantization

To save memory, the amount of bits used to store the non-negative log-probabilities can be reduced with the parameter q to as little as $q = 2$ bits at the expense of accuracy. This reduction yields $2^q - 1$ possible bins. The value of each bin is calculated by equally distributing the probabilities over these bins and computing the average. Note that the quantization is done separately for each order and unigram probabilities are not quantized.

6.6.3 Pointer Compression

To use memory even more efficiently, the pointers which are used to store n-grams and their probabilities can be compressed. Such pointers are used to represent e.g. word IDs (for q-grams) and are stored as sorted integer-arrays. Additionally, These integers can be compressed using a lossless technique from **raj'lossless** by removing leading bits from the pointers and store them implicitly into a table of offsets. The parameter a controls the maximum number of bits to remove. There is a time-space trade-off meaning that a higher value of a will lead to a smaller memory footprint at the cost of a slower training time.

6.6.4 Building the model

The a 5-gram LM was trained on the German Wikipedia corpus using using the Trie data structure and the same parameters like the model downloaded from *DeepSpeech* ($q = 8$ and $a = 255$). Like the *DeepSpeech* model 4- and 5-grams were pruned by setting a minimum frequency of 1.

[22] note that *KenLM* offers a so called *PROBING* data structure, which is fundamentally a hash table combined with interpolation search, a more sophisticated variant of binary search, which allows for constant space complexity and linear time complexity. This does however not change the fact that n-grams can conceptually be thought as a tree of grams

6.7 Evaluating the LM

Literature suggests two methods to evaluate a LM: Extrinsic and intrinsic evaluation.

6.7.1 Extrinsic and intrinsic evaluation

The best way to evaluate a LMis to embed it in an application and measure how much the application improves (**slp3**). This is called *extrinsic evaluation* and has been done by comparing the learning curves with and without using a LM. However, to measure the performance of a LM independently (*intrinsic evaluation*) one would have to provide a test set containing unseen sentences an assess the scores of the LM on their n-grams. The results can then be compared to a reference LM: Whatever model produces higher probabilities (or lower perplexity) to the n-grams in the test set is deemed to perform better. However, models can only be compared if they use the same vocabulary and always encode characteristics of the training corpus (**slp3**). Since the sentences in a corpus of legal documents use different structures and word distributions than a corpus of children's books, two models trained on these corpora will not be comparable. Evaluating the created German Wikipedia corpus intrinsically would therefore require training a reference model on the same corpus, which can become very time consuming.

6.7.2 Evaluation of KenLM

KenLM has been extensively compared to other LM implementations like the SRI Language Modelling Toolkit (SRILM) both in terms of speed and accuracy. It has been found to be both faster and more memory efficient (**kenlm**) than the fastest alternative. Its low memory profile makes it runnable on a single machine, while other algorithms like *MapReduce* target clusters (**kenlm˙estimation**). The highly optimized performance was a big advantage especially for this project because it enabled testing the model on a local machine. The probabilistic performance of *KenLM* has been evaluated by training a 5-gram model on a 126 billion token corpus (393 million unique words) (**kenlm˙estimation**). This model was embedded in some Machine Translation systems (Czech-English, French-English and Spanish-English) . Evaluation was done by calculating the BLEU score and comparing it to embeddings of other LM. *KenLM* placed first in all submissions.

6.7.3 Evaluation of the German Wikipedia LM

Because of time constraints and because *KenLM* has already been extensively evaluated on English I resigned from evaluating my German LM intrinsically, even though the corpus used for training is not as big as the one used in **kenlm˙estimation**. *KenLM* is to date widely recognized as the best performing LM available, which is also emphasized by the usage of a *KenLM* model in the Mozilla implementation of *DeepSpeech*.

To still get an intuition about how well the model performs, two different experiments were made:

- **Experiment 1**: The probability calculated for valid German sentences was compared against variants of the same sentences with the words in randomized order.
- **Experiment 2**: The LM was used together with its vocabulary to build a simple word predictor.

Both experiments are explained in more depth below.

6.7.4 Evaluation 1: Comparing scores of randomized sentences

The first experiment tests the validity of the probabilities (*scores*) calculated by the LM. For this, an arbitrary choice of 5 valid sentences in German was used. To ensure the sentences could not have

been seen during training, the following 5 sentences were taken from a newspaper printed after the creation of the Wikipedia dump:

```
Seine Pressebeauftragte ist ratlos.
Fünf Minuten später steht er im Eingang des Kulturcafés an der Zürcher Europaallee.
Den Leuten wird bewusst, dass das System des Neoliberalismus nicht länger tragfähig ist.
Doch daneben gibt es die beeindruckende Zahl von 30'000 Bienenarten, die man unter dem Begriff ≪Wildbienen≫ zusammenfasst.
Bereits 1964 plante die US-Airline Pan American touristische Weltraumflüge für das Jahr 2000.
```

Listing 2 – Representation in corpus

All sentences have been normalized the same way sentences were preprocessed for training. For each of them the score was calculated. Then the words were shuffled and the score was calculated again. A good LM should calculate a (much) higher probability for the original sentence, because the shuffled sentence is most likely just gibberish. Table 8 shows the results of the comparison. It is evident that the probabilities for the shuffled sentences are much lower than for the sentences where the words appear in the correct order. The probabilities calculated by the LM are therefore deemed valid.

original sentence (normalized)	score	permuation	score
seine pressebeauftragte ist ratlos	-17.58	ist ratlos pressebeauftragte seine	-21.52
fünf minuten später steht er im eingang des kulturcafes an der zürcher europaallee	-40.23	des er minuten zürcher kulturcafes steht europaallee eingang fünf im später an der	-57.69
den leuten wird bewusst dass das system des neoliberalismus nicht länger tragfähig ist	-35.52	system nicht das ist dass leuten tragfähig des neoliberalismus den bewusst länger wird	-51.27
doch daneben gibt es die beeindruckende zahl von <num> bienenarten die man unter dem begriff wildbienen zusammenfasst	-48.36	dem gibt wildbienen zahl beeindruckende doch man zusammenfasst es daneben bienenarten von die unter die <num> begriff	-75.95
bereits <num> plante die usairline pan american touristische weltraumflüge für das jahr <num>	-58.04	plante touristische für jahr pan american das bereits usairline <num> <num> weltraumflüge die	-64.02

Table 8 – Comparison of log10-probabilities calculated for news sentences and a permutation of their words

6.7.5 Experiment 2: Word predictor

The second experiment tests whether the trained LM is able to continue a sentence given its beginning. For this each word from the vocabulary is appended and the score of the resulting stumps is calculated. The most likely continuation can be estimated by sorting the resulting list in descending order (the probabilities are $\log_1$ 0-based, i.e. negative) and taking the first element. This behavior can be applied iteratively to construct a sentence from a stump. For this experiment a sentence was started with the stump ≪ *Ein 2007 erschienenes* ≫. Afterwards a word from the five most probable continuations

was appended. The extended stump was then again fed into the LM. This process was repeated until some kind of sentence ending was encountered. Each extended stump was preprocessed the same way the sentences were preprocessed for training (lowercasing, replacing numbers with `<num>`, etc.). Figure 8 shows the path taken through the predictions. Note that the predictions for the second and third word of the stump after typing the first word are shown in grey for illustrative purposes, although they were not considered for continuation.

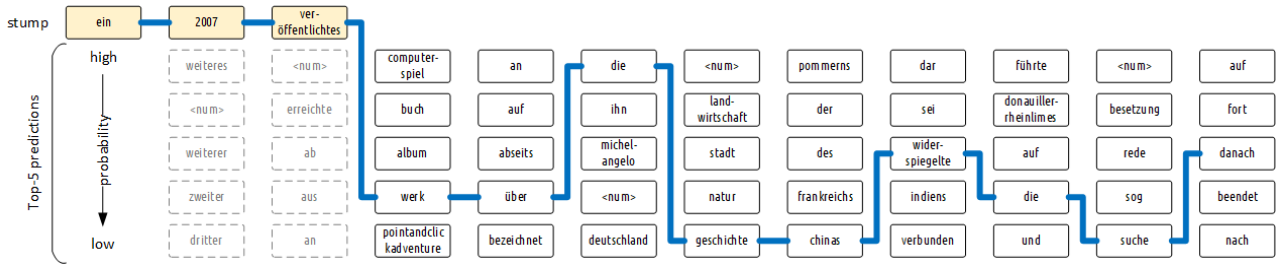

Figure 8 – Word predictions of the trained 5-gram model for continuations of the stump « Ein 2007 erschienenes ... ». The blue path represents a grammatically valid German sentence.

Although prediction was slow we can observe that the words suggested by the LM are generally grammatically correct continuations and often make sense, although the probability for some of the predicted words (like *Michelangelo*) is sometimes unexplicably high. Nevertheless it was possible to create a valid German sentence from the stump using only the suggested words. The LM even seems to have captured some notion about grammatical concepts like German cases (e.g. that « *die Geschichte Chinas* » is more likely than « *die Geschichte China* »). On the other hand we can observe that the meaningfulness of the suggestions decreases with the progress because some long-distance relationships between words are lost for small values of n.

6.8 STT model performance

The observations made when training the simplified Keras model on German audio data are similar to the ones made when training on English data in that the spell-checker will not help and the CTC validation loss will decrease until epoch 15 and then plateau or increase slightly.

When evaluating the LER metric on the test set, the best performance was achieved with a regularized model that was trained on 1.000 minutes of audio, including synthesized samples. The average LER value was then 0.4918, which is even better than the 0.5125 achieved when training on English samples from the LS corpus. This result has to be taken with a pinch of salt though, because the test data in the RL corpus is not as extensive as the one from the LS corpus and it has also not been extracted with the same diligence.

The effect of regularisation and/or use of synthesized training data can be visualized. Figure 9 shows both measures in isolation and in combination. From the plot on the left it becomes evident that transcripts inferred by a regularized model will generally have a lower LER than without regularisation. The plot in the middle shows how the use of synthesized training data has a smoothing effect on the curve of the LER. Finally, the plot on the right shows the progress of the average LER values when combining both measures, i.e. training a regularized model with training data including synthesized samples. The effect is bot a smoother curve and mostly lower LER rates, although there is an awkward spike between epoch 15 and 20.

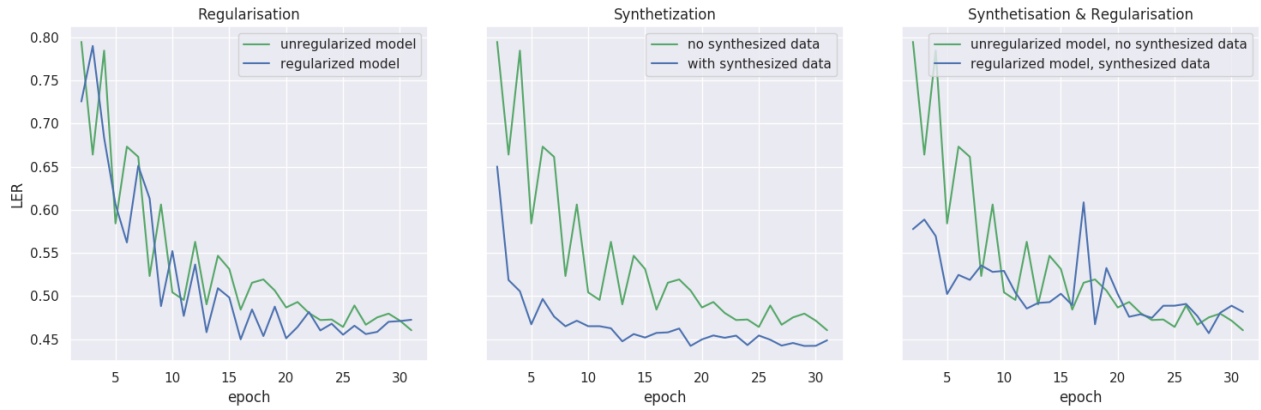

Figure 9 – Impact of regularization and/or synthetisation on the progress of average LER values. Regularization alone (left) will lead to lower LER rates. Synthesized training data (middle) will lead to a smoother curve and lower LER rates. Both measures combined will also combine their effects, although the trend is less obvious.

6.9 Pipeline performance

Because the regularized model trained with synthesized data has both the lowest LER rates and a smoother curve, this model is used in the ASR stage of the pipeline when aligning German samples. The pipeline can then be evaluated like it was evaluated for English samples. However, while for English samples the Mozilla implementation of *DeepSpeech* could be used as a reference model, such a reference model is not available for German[23] because the *CommonVoice* German dataset is not mature enough. To still evaluate how well the pipeline does for German audio/transcript samples, the segmentation information available in the test data is used instead of splitting the audio signal into voiced parts. The speech segments are put through the other stages (ASR and Global Sequence Alignment (GSA)) like before. By doing so, the VAD stage is canceled out and the results reflect the impact of the last two stages. The final alignments are then evaluated as before, i.e. by calculating *P*recision, *R*ecall and *F*-score, but without comparing the values to a reference model. Because there was no reference model, the inferred transcripts were not concatenated and compared to the original transcript.

6.10 Summary

This gave an overview on how n-gram LM work and how a 5-gram model for German similar to the one downloaded from Mozilla was trained on Wikipedia corpus and validated empirically. It also showed how a pipeline was built that aligns German audio/text samples. A STT model was trained using augmented data from *ReadyLingua*. Like with the English samples, the training progress was visualized with a learning curve. The best performance was achieved using an unregularized model that has been trained on 1.000 minutes of original and synthetisized data. Likewise the pipeline performance was evaluated, yielding similar results like the English pipeline. However, the pipeline was evaluated on only 6 samples (12 minutes 29 seconds). This is not enough and does not reflect a realistic distribution of speakers, genders, accents, etc. The pipeline for German needs to be evaluated further to make statements about its general capability.

[23] except proprietary STT engines like *Google Cloud Speech*, which are only available online

7 Conclusion

7.1 Outlook and further work

The pipeline works very well with normalized audio and transcripts. However, this does probably not represent exactly how the pipeline will be used by *ReadyLingua* in production. Depending on its use it may be required to align the partial transcript with an unnormalized full transcript (containing uppercase letters, punctuation, etc.). This evaluation on unnormalized transcripts was not done because for the *LibriSpeech* corpus only the normalized transcripts were available. Efforts have been made to find the text passage in the original book corresponding to the concatenated sequence of transcripts, but this was only done by normalizing the book text too.

Furthermore, it may be interesting how the pipeline behaves with transcripts containing errors, unspoken or missing texts as well as audio that contains distortion like noise, music or multiple speakers. For that, corresponding recordings and transcripts have to be collected first. It might also be possible to generate such samples from the existing data through augmentation.

Both of these topics should provide enough work for a follow-up project. Finally there are some tools encountered during the project that were not tried out because there was no time. One example is the *Hunspell Checker*[24] that could be used instead of the self-implemented spell-checker[25].

[24] `http://hunspell.github.io`

[25] There is a Python module available at `https://github.com/blatinier/pyhunspell`. Dictionaries (needed by Hunspell) for various languages can be downloaded at `https://github.com/wooorm/dictionaries`

List of Figures

List of Tables

Acronyms used in this document

ASR Automatic Speech Recognition

BAS Bavarian Archive for Speech Signals

CNN Convolutional Neural Network

CTC Connectionist Temporal Classification

CV CommonVoice

DS Deep Speech

E2E end-to-end

FA Forced Alignment

FHNW University of Applied Sciences

GCS Google Cloud Speech

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

GSA Global Sequence Alignment

LER Label Error Rate

LM Language Model

LS LibriSpeech

LSTM Long Short Term Memory

LSA Local Sequence Alignment

MFCC Mel-Frequency Cepstral Coefficients

NN Neural Network

RL ReadyLingua

RNN Recurrent Neural Network

SA Sequence Alignment

SGD Stochastic Gradient Descent

STT Speech-To-Text

OOV Out Of Vocabulary

SRILM the SRI Language Modelling Toolkit

SW Smith Waterman

VAD Voice Activity Detection

WER Word Error Rate

The simple spell checker in detail

- split the sentence into words
- for each word w_i in the sentence check the spelling by generating the set C_i of possible corrections by looking it up in V, the vocabulary of the LM, as follows:
  - if $w_i \in V$ its spelling is already correct and w_i is kept as the only possible correction, i.e.

$$C_i = C_j^0 = \{w_i\}$$

  - if $w_i \notin V$ generate C_i^1 as the set of all possible words w_i^1 with $ed(w_i, w_i^1) = 1$. This is the combined set of all possible words with one character inserted, deleted or replaced. Keep the words from this combined set that appear in V, i.e.

$$C_i = C_i^1 = \left\{w_i^1 \mid (w_i, w_i^1) = 1 \wedge w_i^1 \in V\right\}$$

  - if $C_i^1 = \emptyset$ generate C_i^2 as the set of all possible words w_i^2 with $ed(w_i, w_i^2) = 2$. C_i^2 can be recursively calculated from C_i^1. Again only keep the words that appear in V, i.e.

$$C_i = C_i^2 = \left\{w_i^2 \mid ed(w_i, w_i^2) = 2 \wedge w_i^2 \in V)\right\}$$

  - if $C_i^2 = \emptyset$ keep w_i as the only word, accepting that it might be either misspelled, a wrong word, gibberish or simply has never been seen by the LM, i.e.

$$C_i = C_i{>}\,2 = \{w_i\}$$

- for each possible spelling in C_i build the set P of all possible 2-grams with the possible spellings in the next word as the cartesian product of all words, i.e.

$$P = \{(w_j, w_{j+1} | w_j \in C_j \wedge w_{j+1} \in C_{j+1}\}, \qquad C_j \in \{C_i^0, C_i^1, C_i^2, C_i^{>2}\}$$

- score each 2-gram calculating the log-based probability using a pre-trained 2-gram-LM

8 Author's declaration

Hiermit erkläre ich, dass ich die vorliegende schriftliche Arbeit selbstständig und nur unter Zuhilfenahme der in den Verzeichnissen oder in den Anmerkungen genannten Quellen angefertigt habe. Ich versichere zudem, diese Arbeit nicht bereits anderweitig als Leistungsnachweis verwendet zu haben. Eine Überprüfung der Arbeit auf Plagiate unter Einsatz entsprechender Software darf vorgenommen werden.

Würenlingen, December 3, 2018

Daniel Tiefenauer