IP9

Daniel Tiefenauer

September 27, 2018

Abstract

tbd.

Contents

1 Introduction **1**
1.1 Scope and overall goal 1
1.2 Chosen approach and previous work 1
1.2.1 Previous results 1
1.3 Goal of this project 2

2 Training an Recurrent Neural Network (RNN) for Automatic Speech Recognition (ASR) **3**
2.1 Exploiting the *DeepSpeech* model 3
2.2 A simpler *DeepSpeech* model 3

3 Including a Language Model (LM) **4**

4 Plotting a learning curve **5**

5 Creating a Language Model for German **8**
5.1 n-Gram Language Models 8
5.2 Creating a raw text corpus and training the model 8
5.3 Evaluating the model 9

6 Ehrlichkeitserklärung **14**

1 Introduction

This report documents the progress of the project *Forced Alignment with a Recurrent Neural Network*. The project serves as a master thesis at University of Applied Sciences (FHNW) (IP9). Some preliminary work has been done in a previous project (IP8). The overall goal, project situation and background information are described in detail in the project report for IP8 and are not repeated here. Instead, a quick recap is given for the sake of completeness of this documentation.

1.1 Scope and overall goal

ReadyLingua is a Switzerland 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 this data with temporal information, so that for each part of the transcript the corresponding point in the audio/video data can be found. This process is called *acfa*. An *InnoSuisse* project was started in 2018 to research how this could be achieved. The *InnoSuisse* project foresees three different approaches, one of which is followed in this project.

1.2 Chosen approach and previous work

The approach chosen in this project is based on speech pauses, which can be detected using *Voice Activity Detection (VAD)*. The utterances in between are transcribed using *ASR*, for which a *RNN* is used. The resulting partial transcripts contain the desired temporal information and can be matched up with the full transcript with a process called *Local Sequence Alignment (LSA)*.

In the IP8 project, VAD, ASR and LSA were treated as part of a pipeline which split a given audio file into individual utterances, transcribe them and localize them in the original transcript. Since the quality of the ASR stage has an imminent impact on the subsequent LSA stage, the quality of the alignments is heavily dependent on the quality of the partial transcripts. 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

For the VAD stage, an implementation of WebRTC was used which was shown to be capable of detecting utterances with very high accuracy within reasonable time. For the LSA stage a combination of the Smith-Waterman algorithm and the Levenshtein distance was used. This combination included tunable parameters and proved to be able to be able to localize partial transcript within the full transcript pretty well, provided the similarity between actual and predicted text was high enough.

Because the final pipeline should be language-agnostic, the IP8 project proposed the use of *DeepSpeech* for the ASR stage, which uses Connectionist Temporal Classification (CTC) (Graves, Fernández, and Gomez 2006) as its cost function. It included some experiments on what features could be used to train a RNN for the ASR stage. Possible features 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). However, developing a fully-fletched ASR system is extremely time-consuming and could not be done within the project time. For that reason a state-of-the-art *Speech-To-Text (STT)* engine (*Google Cloud Speech*) was embedded in the pipeline as the ASR stage. 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.

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 is on the ASR stage. A RNN is trained that can be used as for this stage in the pipeline. Because the superordinated goal of this project is Forced Alignment (FA) and not Speech Recognition, this RNN only needs to be *good enough* for the downstream LSA stage. By exploring various combinations of properties of the network or the data as well as varying amounts of training data, the conditions should be researched under which such a network could be trained. Concretely, the following aspects shall be examined more closely:

- **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 scarce and 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).
- **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 transcripts (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.

The answers to above questions should help in estimating the effort to create a generic solution (required minimum amount of training data, architecture, etc.). [1]

[1] 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 an RNN for ASR

As stated above, this project does not aim at training a state of the art STT engine. Because the Smith Waterman (SW) algorithm used for local alignment is tolerant to a certain amount of errors in the transcripts, the RNN need only be *good enough* for the task at hand (FA). If such a network can be trained under the given circumstances it could be used in the ASR stage of the pipeline. The pipeline would then become become self-contained and would not ne dependent on a commercial solution that cannot eb tuned and whose inner workings are unknown. Furthermore such a RNN would open up to recognizing languages for which there is not a third-party solution yet, such as Swiss German.

2.1 Exploiting the *DeepSpeech* model

A Neural Network (NN) that had quite an impact on ASR was *DeepSpeech* (Hannun et al. 2014) which 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 a DeepSpeech model is available from Mozilla [2]. Since this implementation uses a LM, the quality of the model is measured as the percentage of misspelled or wrong words (called 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 (Morais 2017).

A model could be trained by providing training-, validation- and test-data for an arbitrary language (e.g. from the *ReadyLingua* corpus). However, this is not the preferred procedure for this project for various reasons:

1. The *DeepSpeech* implementation was designed for ASR. In such settings a low WER is desirable. But this is not the main focus for this project. As a result, the architecture of the Mozilla implementation might be overly complicated for this project, although it might make sense for pure ASR tasks.
2. The problem with above point is that more complex models usually require more training data. However, as for any neural network, the limiting factor for training a RNN is often the lack of enough high quality training data. This becomes especially important when recordings in a minority language should be aligned.
3. The 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, the RNN architecture of the *DeepSpeech* model was used as a basis for a simplified version, which should (hopefully) require less training data in order to converge and still produce partial transcriptions that can be locally aligned.

2.2 A simpler *DeepSpeech* model

An implementation of the RNN used for STT in the previous IP8 project was done in Python using Keras[3]. The following simplifications and alterations were made:

- No LM
- No convolution in first layer
- LSTM instead of SimpleRNN

[2]`https://github.com/mozilla/DeepSpeech`
[3]`https://keras.io`

Figure tbd. shows the architecture proposed in the *DeepSpeech* compared to the simplified version used in this project / with the changes made for this project (eines von beidem verwenden).

tbd: Hier Bild Architektur einfügen

However, despite training on the *LibriSpeech* corpus, this network did not seem converge. Furthermore, performance was a big issue, although the RNN used a simpler architecture and no computational power was needed to query a LM. Training on aligned speech segments from the *LibriSpeech* corpus was not possible within project time because it would have taken approximately two months when using a single acGPU. However, this duration is at least consistent with the experience made by the Machine Learning team at Mozilla Research, which used a cluster of 16 GPUs that required about a week (Morais 2017) to train a variant [4] of the RNN originally proposed in (Graves, Fernández, and Gomez 2006).

In this project some experimentation was done to make the model converge:

- increased number of MFCC features from 13 to 26, because this is the value used in the *DeepSpeech* model
- tried out different optimizers. Surprisingly, Stochastic Gradient Descent (SGD) seemed to work better than Adam
- switched back to a Simple RNN instead of Long Short Term Memory (LSTM)
- include a language model

3 Including a LM

Although CTC is the cost that is optimized during training, the main metrics to evaluate an ASR system are usually WER and LER. The LER is defined as the mean normalized edit distance ($ed(a, b)$ i.e. the number of insertions, deletions and changes required to produce string b from string a) between a an inferred transcription (*prediction*) and the actual transcription (*ground truth* or *label*). It operates on character level and is sometimes also referred to as *Levensthein Distance*. The WER builds upon the LER but operates on word level, i.e. it represents the number of words in a inferred transcription, that need to be inserted, deleted or changed in order to arrive at the ground truth.

If a single evaluation metric is required, the WER is often the better choice because it is more related to the way humans would assess the quality of a transcription: A transcription which might sound correct when read out loud but is full of spelling mistakes is not a good transcription. A LM can help inferring orthographically correct words from sequences of characters detected by CTC and hence decrease the WER. Therefore, by using a LM the quality of transcriptions improves perceivably, as the following example shows:

transcript	value	LER
actual transcript	`and i put the vice president in charge of mission control`	1.00
inference without LM	`ii put he bice president in charge of mission control`	0.11
inference with LM	`i put the vice president in charge of mission control`	0.08

Table 1: examples of infered transcripts with pre-trained DeepSpeech model with and without LM (sample $20161203potusweeklyaddress$ from the ReadyLingua corpus

A LM models probabilities of sequences of words. Therefore a LM can be used to assign a probability to a sentence. A simple LM that does that is the n-gram LM. n-grams are overlapping tuples of words whose probability can be approximated by training on massive text corpora. Although a lot of research has been

[4]the variant used MFCC as features whereas the original paper proposed raw spectrograms

made in the field of using NN for language modelling (like for machine translation), n-grams LM are still widely used and often the right tool for many tasks (Jurafsky and Martin 2019), because they are faster to train and require significantly less training data.

The Mozilla implementation includes an n-Gram LM using *KenLM*. The LM is queried while decoding the numeric matrices produced by CTC using *Beam Search* or *Best-Path* decoding. It uses a *trie* and precompiled custom implementations in C of *TensorFlow*-operations to maximize performance and dedicated weights for the influence the number of valid words and the LM itself on the inferred transcription. It is therefore deeply baked in with the decoding process.

A simpler variant that is used in this project is to infer the transcriptions first with *Beam Search* or *Best-Path* decoding using the standard tools provided by Keras. The inferred transcriptions are then post-processed by running it through some sort of spell-checking, which is done as follows:

- 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

4 Plotting a learning curve

The total length of all transcribed speech segments in the *LibriSpeech* corpus is roughly 47 days (1141 hours). Training on all these samples was not feasible within project time . However, we can get an estimation of the expected learning progress by plotting a *learning curve*. This was done using transcribed audio segments from the *LibriSpeech* corpus. Training was done on exponentially increasing amounts of training data (1, 10, 100 and 1000 minutes of transcribed audio). To assess the impact of using a LM, the learning curve was plotted twice: Figure 1 shows the learning curve for WER and LER without using a LM. Figure 2 shows the learning curve for training with a 4-gram LM.

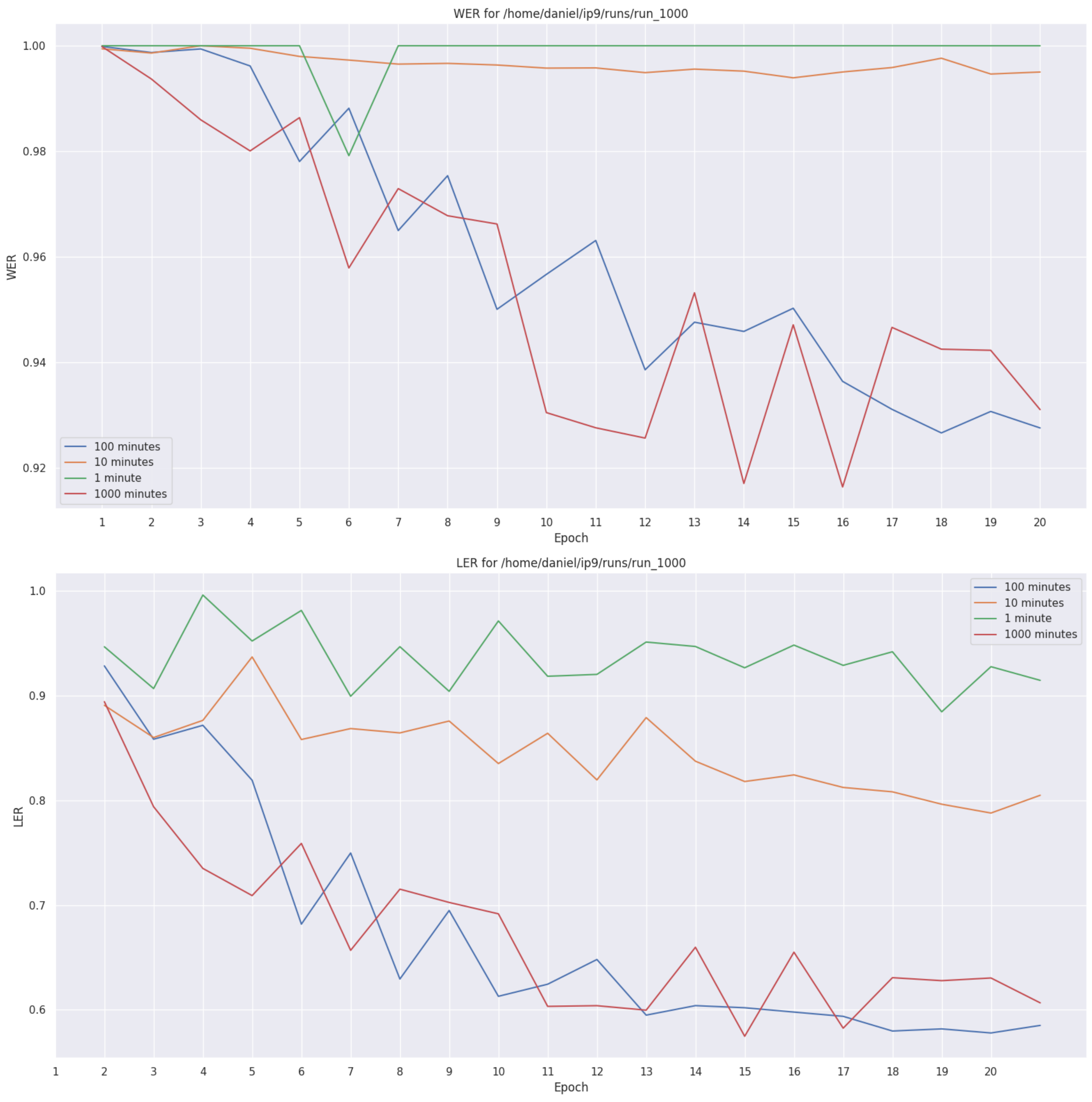
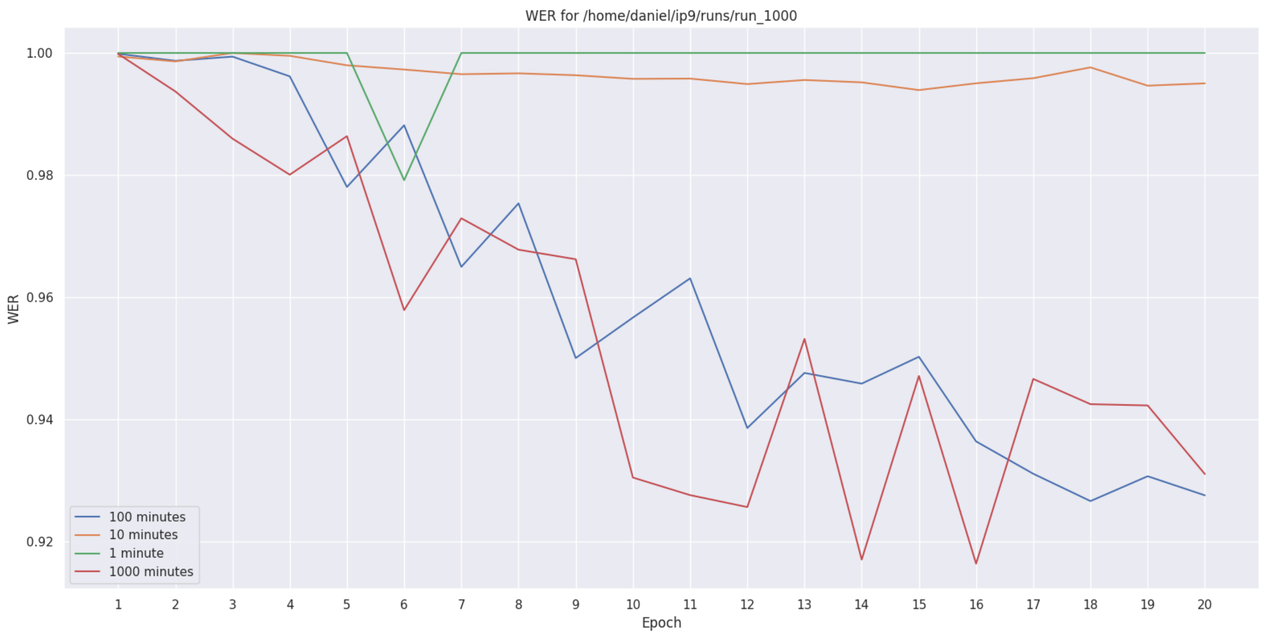


Figure 1: Learning curve for training on 1/10/100/1000 minutes of transcribed audio from ReadyLingua using a 2-gram LM and Beam-Search decoding



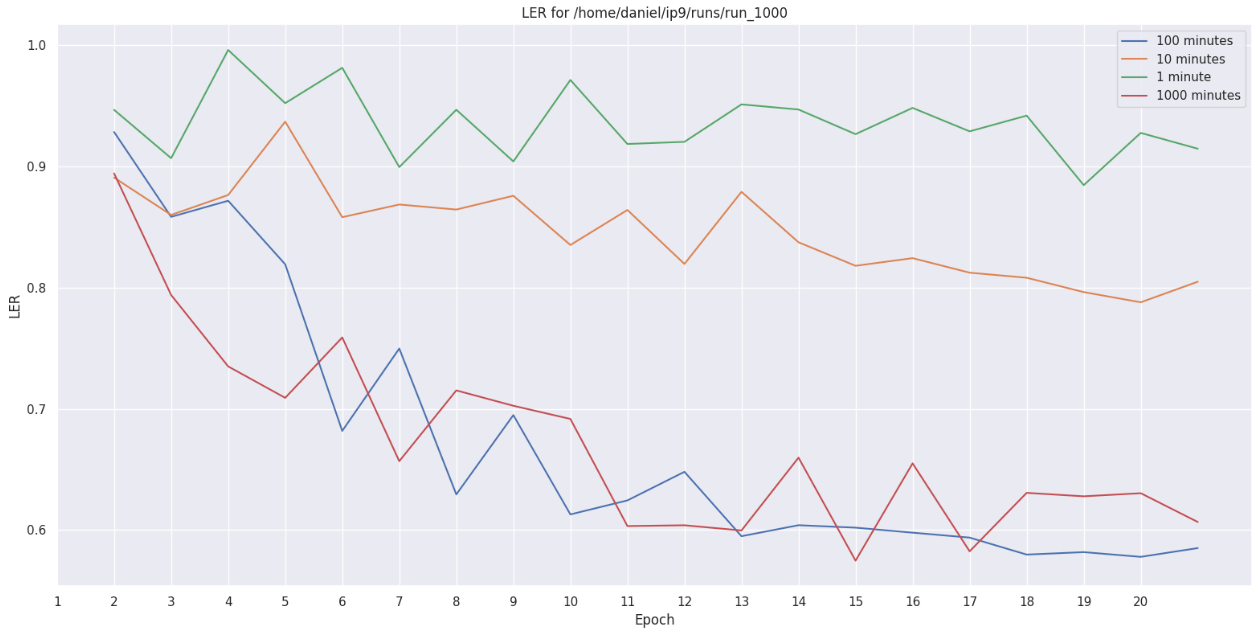


Figure 2: Learning curve for training on 1/10/100/1000 minutes of transcribed audio from ReadyLingua using a 2-gram LM and Beam-Search decoding

5 Creating a Language Model for German

The learning curve above was plotted using the results from training on English samples. To compare get an intuition about whether the conclusions made are transferable to training on samples in other languages, the training was done again on German samples. In order to do this, a language model for German had to be created first.

5.1 n-Gram Language Models

To understand the following sections better it might be helpful to get a quick recap about LM. The most widely used type of 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. Single words are 1-grams, but the same applies for n-grams of any order. But because of combinatorial explosion the n-gram, such LM suffer from sparsity with increasing order n. To handle OOV issues 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, which is important when calculating the perlexity of a model (because the count appears in the divisor and we cannot divide by zero) Although with Laplace Smoothing a very low probability is assigned to previously unseen n-grams (which results in a high perplexity), it performs poorly in application. A better way of smoothing is achieved using *Kneser-Ney Smoothing*.

5.2 Creating a raw text corpus and training the model

A LM was trained on a raw text corpus of German Wikipedia articles using KenLM (Heafield 2011). The articles were pre-processed to meet the requirements of *KenLM*. It was normalized by removing Wiki markup, punctuation and making everything lowercase. Accentuated characters (like `è,é,ê`, etc.) and special characters like the German *ß* were translated to their most similar ASCII-equivalent (`e` resp. `ss`) to account for ambiguous spelling and reduce the number of words. Umlauts (although not part of the ASCII codeset) were kept as-is because they are used very frequently in German.

Because *KenLM* expects the input as sentences (one sentence per line), the raw text was further tokenized into sentences and words using NLTK (Loper and Bird 2002). Word tokens that contain only numeric characters (such as year numbers) are changed to `<unk>`, a special token which is traditionally used to denote an OOV word. Although numeric tokens occur frequently in the Wikipedia articles, they are unwanted in the corpus because they do not carry any semantic meaning and because there is an infinite number of possible numbers.

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[5])

```
die grösse des wörterbuchs hängt stark von der sprache ab
zum einen haben durchschnittliche deutschsprachige sprecher mit circa <unk> wörtern
    einen deutlich grösseren wortschatz als englischsprachige mit rund <unk> wörtern
```

[5] https://de.wikipedia.org/wiki/Spracherkennung

```
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

The final corpus contained data from 2,221,101 Wikipedia articles (42,229,452 sentences, 712,167,726 words, 8,341,157 unique words). A 4-gram *KenLM* model was trained on this corpus. KenLM uses *Kneser-Ney Smoothing*. n-grams can be represented with a tree structure [6], which allows for pruning. The n-grams used to train the LM have been pruned by setting the minimal threshold for n-Grams of any order ($n \in 1..4$) to 40. This is the value that Google used (reference from Jurafsky). Pruning unigrams helped getting rid of obvious spelling mistakes and very rare tokens that only appear in very special contexts (like the tokens *aaaaa* or *zzyzzyxx*) (EDIT: Pruning unigrams is not supported by KenLM, but the vocabulary can be limited). Such words are mostly not no real German words and should therefore not be trained on. Pruning higher-order n-grams was done to increase performance (both in space and time).

5.3 Evaluating the model

The best way to evaluate a LMis to embed it in an application and measure how much the application improves (Jurafsky and Martin 2019). 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, because models can only be compared if they use the same vocabulary (Jurafsky and Martin 2019), this would require training the reference model would need to be trained on the same corpus, which can become very time consuming.

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 (Heafield 2011) than the fastest alternative. Its low memory profile makes it runnable on a single machine, while other algorithms like *MapReduce* target clusters (Heafield et al. 2013). This was a big advantage especially for this project. The probabilistic performance of *KenLM* has been evaluated by training a 5-gram model on a 126 billion token corpus (393 unique words) (Heafield et al. 2013). 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.

Because of time constraints and because *KenLM* has already been extensively evaluated on English I resign from evaluating my German LM intrinsically, although the corpus used for training is not as big as the one used in Heafield et al. 2013. To this day *KenLM* is widely recognized as the best performing LM out there, which is emphasized by the usage of a **KenLM!** (**KenLM!**) model in the Mozilla implementation of *DeepSpeech*.

To still get an intuition about how well the model performs, the model's score on some test sentences were calculated. To make sure the sentences could not have been seen during training, the following set of 5 sentences of the current newspaper (date after creation of the Wikipedia dump) was used:

- Seine Pressebeauftragte ist ratlos.
- Fünf Minuten später steht er im Eingang des Kulturcafés an der Zürcher Europaallee.

[6]note that *KenLM* offers a s.c. *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

- 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.

The score for each sentence was calculated. Then the words of the sentences were shuffled and the score was calculated again. A good LM should calculate a higher probability for the original sentence, because the shuffled sentence is most likely to be gibberish. All sentences have been normalized the same way sentences were preprocessed for training. Table 2 shows the results of the comparison.

original sentence (normalized)	score	permuation	score
seine pressebeauftragte ist ratlos	0	foo	0
fünf minuten später steht er im eingang des kulturcafes an der zürcher europaallee	0	foo	0
den leuten wird bewusst dass das system des neoliberalismus nicht länger tragfähig ist	0	foo	0
doch daneben gibt es die beeindruckende zahl von `<unk>` bienenarten die man unter dem begriff wildbienen zusammenfasst	0	foo	0
bereits `<unk>` plante die usairline pan american touristische weltraumflüge für das jahr `<unk>`	0	foo	0

Table 2: Comparison of scores calculated for news sentences and a permutation of their words

List of Figures

1 Learning curve for training on 1/10/100/1000 minutes of transcribed audio from *ReadyLingua* using a 2-gram LM and Beam-Search decoding 6
2 Learning curve for training on 1/10/100/1000 minutes of transcribed audio from *ReadyLingua* using a 2-gram LM and Beam-Search decoding 7

List of Tables

1 examples of infered transcripts with pre-trained DeepSpeech model with and without LM (sample `20161203potusweeklyaddress` from the ReadyLingua corpus 4
2 Comparison of scores calculated for news sentences and a permutation of their words 10

References

Graves, Alex, Santiago Fernández, and Faustino Gomez (2006). “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks”. In: *In Proceedings of the International Conference on Machine Learning, ICML 2006*, pp. 369–376.

Hannun, Awni Y. et al. (2014). “Deep Speech: Scaling up end-to-end speech recognition”. In: *CoRR* abs/1412.5567. arXiv: 1412.5567. URL: http://arxiv.org/abs/1412.5567.

Heafield, Kenneth (July 2011). “KenLM: Faster and Smaller Language Model Queries”. In: *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland, United Kingdom, pp. 187–197. URL: https://kheafield.com/papers/avenue/kenlm.pdf.

Heafield, Kenneth et al. (Aug. 2013). “Scalable Modified Kneser-Ney Language Model Estimation”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. Sofia, Bulgaria. URL: http://kheafield.com/professional/edinburgh/estimate%5C_paper.pdf.

Jurafsky, Daniel and James H. Martin (2019). *Speech and Language Processing (Draft of 3rd Edition)*. URL: https://web.stanford.edu/~jurafsky/slp3/.

Loper, Edward and Steven Bird (2002). “NLTK: The Natural Language Toolkit”. In: *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*.

Morais, Reuben (2017). *A Journey to <10% Word Error Rate*. URL: https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate (visited on 09/14/2018).

Acronyms used in this document

ASR	Automatic Speech Recognition
CTC	Connectionist Temporal Classification
E2E	end-to-end
FA	Forced Alignment
FHNW	University of Applied Sciences
GPU	Graphics Processing Unit
LER	Label Error Rate
LM	Language Model
LSTM	Long Short Term Memory
LSA	Local Sequence Alignment
MFCC	Mel-Frequency Cepstral Coefficients
NN	Neural Network
RNN	Recurrent Neural Network
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

6 Ehrlichkeitserklärung

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, September 27, 2018

Daniel Tiefenauer