Speech-To-Text Engine for Forced Alignment

Daniel Tiefenauer (daniel.tiefenauer@students.fhnw.ch)

Institute for Data Science (I4DS), University of Applied Sciences FHNW, Windisch (Switzerland)

Abstract

Forced Alignment is concerned with aligning chunks of text (e.g. parts from a transcript) with parts of a corresponding audio signal. This involves enriching the textual data with temporal data. The process is usually very time consuming because it is often done by hand. This paper introduces an method that reduces said time by producing alignments automatically. The result can be used as-is or with little manual adjustments. The method uses a pipeline which automatically detects voiced segments, transcribes them and aligns the partial transcript with the full transcript using Global Sequence Alignment. Transcriptions are produced by a *Speech-To-Text (STT)* engine building upon simplified architecture of the *DeepSpeech* architecture. The main idea beind this is that Global Alignment can be done even if the partial transcripts match the ground truth only poorly. It has been shown that such transcripts can be obtained from a STT engine trained on very little training data.

Introduction

Automatic Speech Recognition (ASR) has made huge progress in recent years. One idea that had a significant impact on ASR was *Connectionist Temporal Classification (CTC)* (Graves, Fernández, and Gomez 2006), which can be used in *Sequence-to-Sequence (S2S)* models and produces an output sequence of labels for an input sequence of features. CTC is often used in STT engines where the input sequences are audio features (e.g. spectrograms) and the output sequences are their transcripts (i.e. a sequence of characters). Such engines use probabilistic models whereas for each input frame a distribution over the output labels is calculated. The input sequence is typically much longer than the output sequence. The advantage of CTC is that it is *alignment-free*. The optimal alignment is found by calculating the probabilities of each alignments using dynamic programming and then marginalizing over the set of valid alignments (alignments collapsing to the actual transcription).

A famous architecture for a S2S model using CTC has been proposed by *DeepSpeech* (Hannun et al. 2014). A slightly modified implementation is continuously being developed by Mozilla[1], achieving *Word Error Rates (WER)* of as low as 6.5% (Morais 2017). One approach for *Forced Alignment* could now be to simply transcribe the audio and run some sort of spell-checker over the result in order to get the full transcript. However, such a transcript would then include unwanted effects like stuttering, repetitions, mispronunciation etc. Another approach would be to align the produced transcript with a known ground truth. However, this is only possible for languages where a STT engine is readily available. This is often not the case for minority languages like Swiss German. Training an ASR model for those language usually requires large amounts of training data which are often not available.

This paper focuses on aligning chunks of text, not individual characters. It draws upon the assumption that for this purpose a STT engine producing only very low-quality transcripts might be sufficient. Such inferior transcripts can still be good enough for a Sequence Alignment algorithm which aligns them with some sort of ground truth. A fully-fletched STT engine like the one provided by Mozilla on the other hand are targeted at high recognition rates. This quality is probably not needed in the first place.

This paper therefore uses a simplified version of the *DeepSpeech* model as implemented by Mozilla, which uses a *Recurrent Neural Network (RNN)* with CTC as its cost function. The main idea behind this is that a simpler model requires significantly less data than a sophisticated variant and can therefore be trained for languages where such data might be scarce.

The pipeline approach

This paper introduces a pipelined approach for automated *Forced Alignment*. A combination of audio (the *signal*) and text (the *ground truth*) can be put through this pipeline, resulting in a list of aligned text passages. Each item in this list contains the desired temporal information (i.e. start and end frame in the audio signal) as well as the partial transcript that lead to the alignment. The pipeline consists of four stages:

[1] `https://github.com/mozilla/DeepSpeech`

1. *Preprocessing*: Preparing the signal and ground truth for inference resp. for training

2. *Voice Activity Detection (VAD)*: Splitting the audio signal into voiced segments

3. *Automatic Speech Recognition (ASR)*: Transcribing each voiced segment

4. *Global Sequence Alignment (GSA)*: Aligning each partial transcript with the ground truth

All stages were implemented in Python. Since the GSA stage operates on character level, a low *Label Error Rate (LER)* is desireable. The key stage for this is the STT engine in the ASR stage because – although being simple and trained on little data – it should be able to produce *good enough* transcripts fit for the downstream GSA stage.

Preprocessing

Since the model architecture is based on the Mozilla implementation of *DeepSpeech*, the input data must be transformed to a compatible format. Audio signals are converted to mono PCM-16 wave files encoded with 16-bit raw bytes (little-endian). The text data is normalized by restricting it to an alphabet used by the corresponding language. Such an alphabet usually consists of the letters of the latin alphabet as well as other characters commonly found in the language (like the apostrophe in English or umlauts in German). Normalization is done by lowercasing the text and removing punctuation.

Voice Activity Detection (VAD)

The preprocessed audio signal is split into non-silent parts by using the Python port[2] of the *WebRTC* project[3]. This algorithm has proved to be very efficient both in time needed to make the split as well as in the quality of the splits.

Automatic Speech Recognition (ASR)

The STT model is implemented using the *Keras* framework and is based on the Mozilla implementation of *DeepSpeech*. The Mozilla architecture was simplified by not using a Language Model (LM) in the training process, discarding the convolution in the first layer and not using context frames. The model uses a sequence of T_x MFCC features with $n = 26$ features in its input layer. The input is put through three fully connected layers with 1,024 units followed by a bidirectional layer. All intermediate layers use LSTM cells with 1,024 units and the *Rectified Linear Units (ReLU)* activation function because this has been proved to work well for acoustic models (Maas 2013). The output layer is again fully-connected using softmax to produce a vector of probabilities for each feature vector. Experiments were made with and without regularizing the model using dropouts between each layer. The LER and WER on the validation set was slightly lower with the regularized model. Figure 1 shows the simplified model graph.

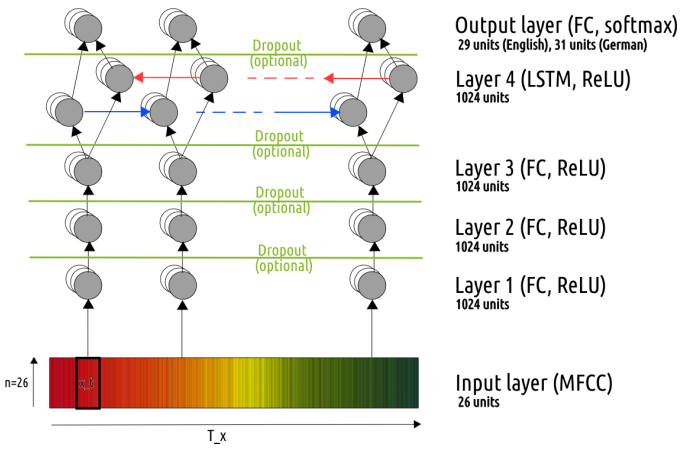

Figure 1: Simplified RNN architecture proposed in this paper. Each input frame t_x in the $(n \times T_x)$ input layer is put through a series of fully-connected (FC) layers. Layer 4 is a bidirectional recurrent layer. The output layer assigns a probability distribution for each input frame.

Global Sequence Alignment (GSA)

The partial transcripts produced by the ASR stage are concatenated and separated by a single space to a string B . This string is then aligned with the full transcript as reference string A. A modified version of the *Needleman-Wunsch* algorithm is used for global alignment. This algorithm calculates a matrix of alignment costs and then backtracks it to find the optimal alignment for characters of string B with string A by dropping or inserting characters.

Because in its original form, the *Needleman-Wunsch* algorithm has no notion of partial transcripts it does not keep track of where they start and end. The modified algorithm therefore produces a list of alignments by annotating the string of concatenated partial transcripts with information about the boundaries of the partial transcripts. Because the optimal alignment is found by tracing the cost matrix from the back, the end index j_{end} of a partial transcript is encountered before its starting index j_{start}. Each time an end j_{end} index is aligned with some index i_{end} in the reference string, this index is noted. As soon as the start index j_{start} is encountered, the corresponding start index i_{start} in the reference string is also noted. The aligned text for the partial transcript can then be created by extracting the substring $A[i_{start} : i_{end}]$ from the reference string.

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

Training the simplified model

Two different variants of the simplified model were trained to recognize German and English. For the English variant the output layer consisted of 29 units because the probability was calculated for 29 labels[4]. Because the apostrophe is far less common in German, this character was dropped for the German variant. Instead, umlauts were added to the set of labels because they are very common in German. Therefore the German model used 31 units in its output layer[5]. Both variants were regularized by adding dropouts between the layers.

English model variant

The English variant of the simplified model was trained on the *CommonVoice (CV)* corpus, which is built and maintained by Mozilla[6] and is also used to create a pre-trained model using Mozilla implementation of *DeepSpeech*[7]. The samples from the training set were sorted by their audio length. Only the first 1,000 minutes of audio were used for training. The CTC training loss was calculated for these samples. Samples from the validation set were used to calculate the validation loss. Figure 2 shows the curve for both losses. It is evident that the model does not improve anymore after epoch 15 and start to overfit after that.

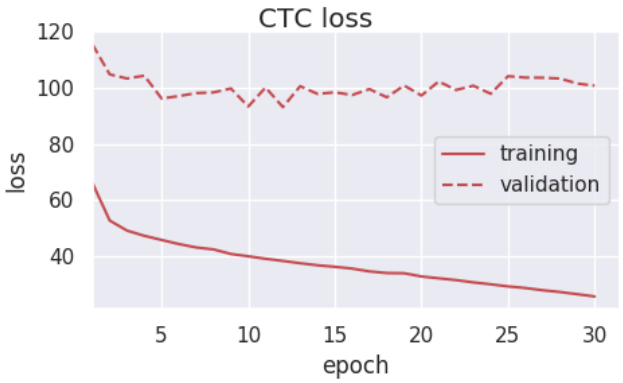

Figure 2: Plot of the CTC loss for training and validation data. Because the validation loss does not decrease after epoch 15, the model will start to overfit after that point.

Additionally, the model was used after each epoch to make inferences on the samples from the validation set using *Beam Search Decoding* (Graves, Fernández, and Gomez 2006). Each inference was compared against its ground truth by calculating the LER. Figure 3 shows how this metric evolves with the training progress.

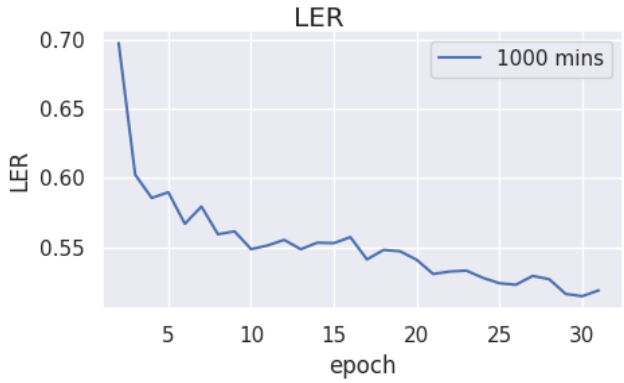

Figure 3: Plot of the average LER between inference and ground truth. The inferences were made for samples from the validation set using the model trained on 1,000 minutes of data from the CV corpus after each.

German model variant

For the German variant of the simplified model labelled data with a total audio length of approximately two hours was used. This data consists of recordings of German speakers and their transcripts. The split into training-, validation- and test-set was done automatically and not with the same diligence as the CV corpus meaning the distribution of sexes, accents etc. might not reflect a realistic setting. The split was done so the training set contained 80% of the labelled data and the validation and test set 10% each. The training set hence contained only about 80 minutes of audio data which were augmented by adding distortion (change of pitch, tempo or volume and adding delay or echo). Each of these distortions was applied in isolation and combination until 1,000 minutes of audio data were available. Training was then done like with the English variant. Figure 4 shows the progress of the training loss.

Apart from an awkward spike in epoch 16 the plot is similar to the one from the English variant in that there is a slight trend to increasing validation loss after epoch 15. Figure 5 shows the progress of the LER on validation data.

The trend for the LER is less smooth than with the English model. The spike from the CTC-loss is also visibile here.

Using a simple spell-checker

Since the simplified model produces a transcript as an arbitrary sequence of characters, experiments were made whether these results could be improved by applying some sort of spell checking. For this a simple spell-checker was implemented that uses a vocabulary V of the 80,000 most frequent word of the corpus it was trained on. The space

[4] 26 lowercase letters from the alphabet, space, apostrophe and the blank token used by CTC

[5] 26 lowercase letters from the alphabet, 3 umlauts, space, blank token

[6] https://voice.mozilla.org

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

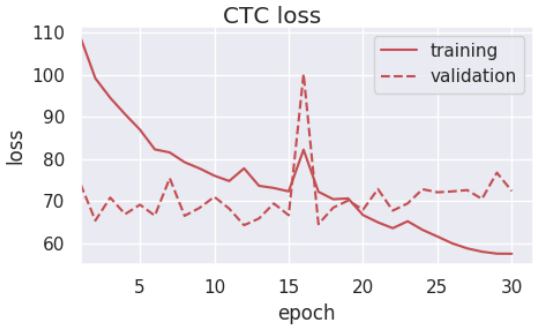

Figure 4: Plot of CTC training- and validation- loss when training on German samples. 1000 minutes of original and synthesized data were used. There is an inexplicable spike at epoch 16.

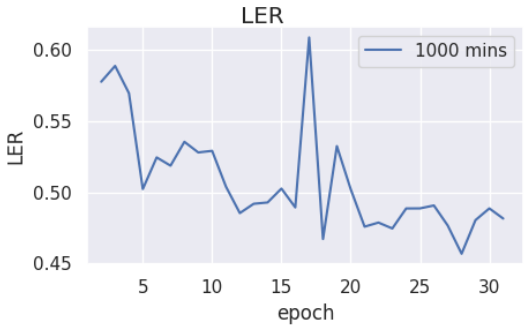

Figure 5: Curve of the LER when calculated for inferences from the validation set. Note the spike in epoch 16 is also visible here.

separated word tokens from the inference were compared against this vocabulary. If the word token appeared in the vocabulary it was deemed valid and the next word was compared. If it did not appear in the vocabulary, all variants of the word with edit distance 1 were created by adding, removing or changing individual characters in the word with characters from the alphabet. Only the set W_1 of variants that appeared in the vocabulary were kept. If this set was empty ($W_1 = \emptyset$), the set W_2 of variants with edit distance 2 was created from W_1 similarly. If $W_2 = \emptyset$ the unchanged original word was kept.

All word tokens from an inference were processed from left to right with above procedure. Partial sentences were created by producing the cartesian product of all combinations. After each word the likelihood of each partial sentences was estimated using a 5-gram *Language Model (LM)* and only the 1,024 most likely combinations were kept. After all words were processed, the most likely sentence was kept as the corrected sentence.

For English, the *KenLM* ·(Heafield 2011) model available from the pre-trained model from Mozilla was used to make the estimation. For German a custom model was trained on a text corpus derived from about 2.2M Wikipedia articles. The articles were normalized by removing Wiki markup, lowercasing the text and tokenizing the text into lines of individual sentences. Furthermore, numeric tokens were replaced by the `<num>` token because such tokens represent values (e.g. year numbers) and do not carry semantic meaning.

With above procedure a rudimentary spell-checker was implemented. Although some informal experiments showed the spell-checker is working provided the LER is sufficiently low, it did not improve the inferences produced by the simplified model in most cases. Often the corrected inference had a higher LER than the original inference when compared to the ground truth because the character sequences were changed to a wrong word. Apparently the LER of the original inference is still too high for such a spell checker. Therefore the original uncorrected inferences were used for the GSA stage.

Tests and results for English samples

The alignments produced by the pipeline using the simplified model were evaluated by comparing them against alignments produced the same pipeline but using the pre-trained model from Mozilla as a reference model in the ASR stage. The test set consisted of audio and text samples from the *LibriSpeech* corpus[8]. This corpus contains speech samples from recordings of books read by various speakers. Audio and text data were derived by exploiting the corpus metadata. The audio files were created on a per-speaker basis by cropping the original recordings so that the signal begins with the first sample from the corpus entry and ends with the last sample. The transcript was extracted by looking up the transcript of the first and the last speech sample in the original book text. Some of the transcripts had to be manually adjusted to make sure they only contain the spoken text for each recording. Both audio and text were preprocessed as described above.

The quality of the alignments was measured by calculating the precision (P) of the alignemnts as the average *Levenshtein Similarity* between all partial transcripts and their corresponding aligned text. Further, the recall (R) was measured by calculating how much of the ground truth was aligned. Both values were combined to a single metric by calculating the F-Score (F). P, R and F were calculated for each sample from the test set. Figure 6 shows box plots for these metrics calculated for both pipelines. Table 1 lists the mean and median values.

It is evident that P and thus F is significantly higher for the pipeline using the reference ASR model. This is no surprise because the transcript produced by the reference

[8] `http://www.openslr.org/12/`

	P		R		F	
	mean	median	mean	median	mean	median
Reference pipeline	0.865	0.879	0.999	1.0	0.926	0.935
Simplified pipeline	0.435	0.443	0.999	1.0	0.602	0.614

Table 1: Average Precision (P), Recall (R) and F-Score (F) of alignments produced over all test samples for a pipeline using the reference model in the ASR stage compared to the results of the same pipeline using a simplified model

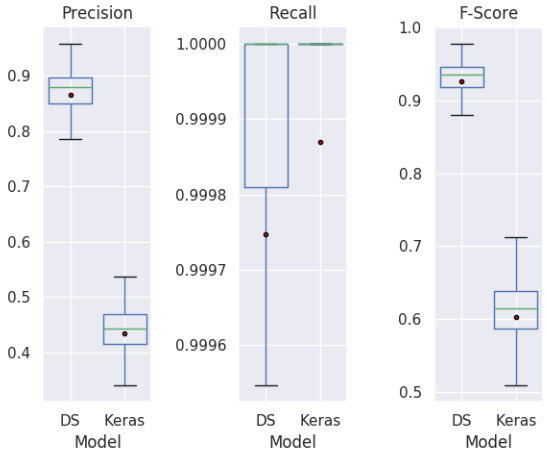

Figure 6: Precision, Recall and F-score of a pipeline using the Mozilla implementation of *DeepSpeech* in its ASR stage compared against the same pipeline using the simplified model.

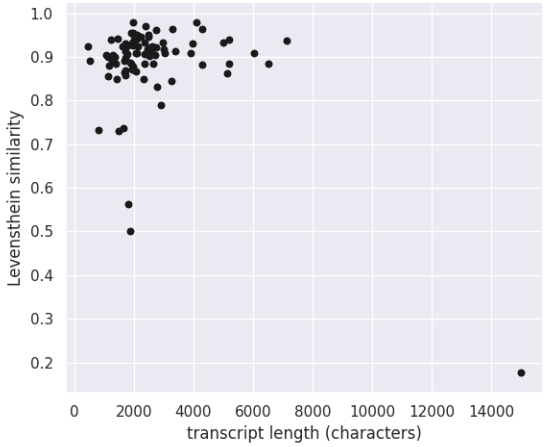

Figure 7: Precision, Recall and F-score of a pipeline using the Mozilla implementation of *DeepSpeech* in its ASR stage compared against the same pipeline using the simplified model.

model are generally much better quality. However, P, R and F measure the quality of alignment for each pipeline in isolation. To capture the quality of alignments from the pipeline using the simplified model better, they were compared against the reference alignments produced by the pipeline using the pre-trained model from Mozilla. The latter is considered a hypothetical optimum and the comparison is made by calculating the average *Levenshtein Similarity* between the alignments for each sample. Figure 7 visualizes these values in a scatter plot.

Obviously the lower values of P and F for alignments produced by the pipeline using the simplified model did not affect the result very much. Alignemts produced by the pipeline using the simplified model are very similar to the ones produced with the pipeline using the reference model, with the average similarity ranging betwen 0.9 and 1.0 for most samples. This is true regardless of the length of the transcript with only a few outliers. This came a bit as a surprise. Obviously, the *Needleman-Wunsch* algorithm can handle faulty partial transcripts very well and makes the alignments based on only few character sequences. The high quality of alignments produced by pipeline using the simplified model is justified when visualizing the alignment by highlighting the aligned text as the audio is being played. The perceived quality is then considered very high, with only few words at the beginning or end of an alignment being assigned the wrong voiced segment.

Transfer to other languages

Above tests were repeated with a pipeline using the simplified model trained on German samples (including synthesized data). The test set consisted of six short and 6 longer recordings of German speakers together with their transcripts. Because the split into training-, validation- and test-set was done automatically, all samples happen to be read by female speakers. Figure 8 contains the box plots for P, R and F calculated for the alignments produced by this pipeline. The values are very similar to the ones achieved on English samples. Obviously the change in language did not affect the result very much. This has to be taken with a pinch of salt however, because the distribution of the test set does not reflect a realistic scenario. It is also too small to draw generalizable conclusions from it.

Since there was no reference ASR model for German, the

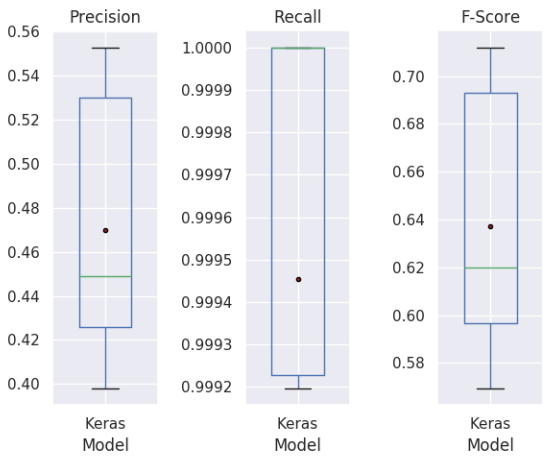

Figure 8: Precision, Recall and F-score of a pipeline using the simplified model that was trained on 1,000 minutes of German data (original and synthesized).

alignments could not be compared to reference alignments. Instead, they were compared to alignments derived from manual inspection. Figure 9 shows the similarities between both alignments. It is evident that the shorter and longer samples form clusters in the plot. Apparently the similarity between alignments decreases for longer samples.

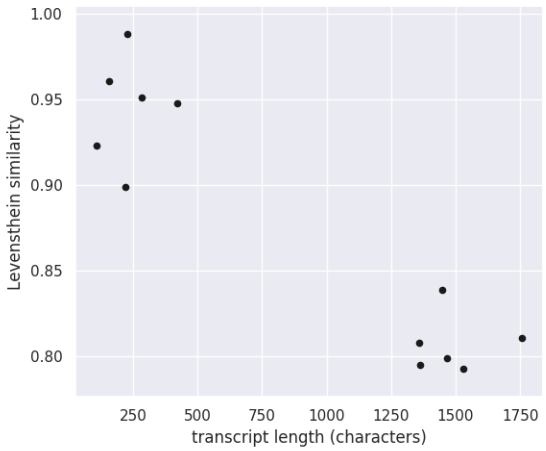

Figure 9: Levenshein similarity between alignments produced by the pipeline using the simplified model and manually produced alignments. Note how the shorter and longer samples form clusters in the plot, while the alignments for the shorter samples are generally more similar to the manual alignments.

Conclusion and further work

Above results were made on English and German samples and might be valid for other languages as well. The pipeline approach generally works very well on these languages. However, it is expected to fail when being applied to samples of languages of other families. Such languages (e.g. Asian languages) can use completely different phonetic and/or graphological concepts. However the alignment produced by the pipeline might be considered a good starting point for many cases, needing only minor manual adjustments (sometimes even none).

Since the results for German samples were achieved with only 80 minutes of training data, they are promising for languages where labelled data is hard and/or expensive to acquire. Furthermore, for the alignments being a combination of audio/transcript, it would be interesting whether this pipeline could be used to generate labelled data to train a STT model, maybe even the one used in the pipeline.

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.

Maas, Andrew L. (2013). "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In:

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).