Automatic Speech Recognition for Afrikaans and isiXhosa

Pectora roborant cultus recti

UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY

Lucas Meyer

Research assignment presented in the partial fulfilment
of the requirement for the degree of
MSc (Machine Learning & Artificial Intelligence)
at the University of Stellenbosch

Supervisor: Prof. H. Kamper

Degree of confidentiality: A

November 2023

PLAGIARISM DECLARATION

1. Plagiarism is the use of ideas, material and other intellectual property of another's work and to present it as my own.
2. I agree that plagiarism is a punishable offence because it constitutes theft.
3. Accordingly, all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
4. I also understand that direct translations are plagiarism.
5. I declare that the work contained in this research assignment, except otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this research assignment or another research assignment.

22614524	
Student number	Signature
L. Meyer	6 November 2023
Initials and surname	Date

Copyright © 2023 Stellenbosch University

All rights reserved

ACKNOWLEDGEMENTS

The Department of Statistics and Actuarial Science (the Department) wishes to acknowledge David Rodwell for generously creating a template based off the USB (University of Stellenbosch Business School) guidelines which have been adapted for the purposes of the department.

ABSTRACT

Insert an abstract of not more than 500 words here

Key words:

Technical guidelines; Chapter outline; Examples

CHAPTER 1

INTRODUCTION

Enter introduction here ...

CHAPTER 2

BACKGROUND

2.1 THE AUTOMATIC SPEECH RECOGNITION TASK

Speech recognition or automatic speech recognition (ASR) is the task of predicting the text, sentence, transcription, or sequence of characters for a corresponding speech recording. The general approach of ASR is to first compute feature representations called *speech features* from given audio data, and then map the speech features to characters or other tokens such as wordpieces.

2.1.1 Speech recognition data

The first step of creating an ASR model is to prepare the data that is used for the model. A single data entry for an ASR dataset is a speech recording (typically in waveform audio file format) along with a corresponding text transcription of the words which are spoken. We explain why the choice of dataset has a significant effect on the accuracy of ASR models.

The amount of training data. The more data that is available during training the better the ability of the ASR model to generalize. A small dataset (with few unique voices) may lead to overfitting to the specific voices in the dataset.

The difference between read and conversational speech. Humans tend to pronounce their speech more clearly when reading text from a transcript, and recent ASR models can predict read speech very accurately [1]. In contrast, accurately predicting conversational speech is still a major challenge in ASR.

Tonal qualities for different accents. The accent of the speaker, which depends on the gender, age and ethnicity of the speaker is another important factor to bare in mind. Generally, male speakers have a lower pitch compared to female speakers. Similarly, adult voices are generally have a lower pitch compared to children.

The audio quality of speech recordings. The position of the microphone, the quality of the microphone, the number of microphones available, and the presence of background noise contribute towards the quality of speech recordings.

2.1.2 Speech features

Computing speech features is useful because audio data consists of a one-dimensional array of integers that describe the amplitude of the recorded sound wave for small time periods called *samples* (see Figure **??**). The issue is that mapping a sequence of amplitude measurements to a sequence of characters is impractical. A common technique used to compute speech features is to transform the audio data from the amplitude-time domain to the frequency-time domain, using the Fast Fourier Transform (FFT) algorithm [2], [3]. However, in this study we discuss a more recent feature extraction approach based on contrastive learning.

2.2 WAV2VEC 2.0

wav2vec 2.0 provides a framework for learning speech representations using unlabeled speech data. wav2vec 2.0 can be applied to a variety of speech-related tasks such as speech recognition, speech translation, and speech classification. It has proved to be particularly useful in cases where a lot of unlabeled data is available, but not much labeled data is available. The authors show that using just ten minutes of labeled data and pretraining on 53k hours of unlabeled data still achieves 4.8/8.2 WER on the clean/other test sets of Librispeech [].

The general two-step approach for using wav2vec 2.0 for any speech-related task is the following. First train (or "pre-train") the wav2vec 2.0 model on a large corpus of unlabeled data, which will give you a model that converts audio data into speech features. After pretraining, fine-tune the wav2vec 2.0 model for speech recognition using a much smaller corpus of labeled data. Fine-tuning wav2vec 2.0 for speech recognition involves replacing the head of the pretrained model with an appropriate loss function such as CTC.

The wav2vec 2.0 architecture is described by the network diagram in Figure 2.1. There are three important components of the wav2vec 2.0 architecture: the feature-encoder, the quantization module, and the context network. The objective of wav2vec 2.0 becomes clear only after understanding each of the three components. Thus, the way in which wav2vec 2.0 is trained is only explained after discussing the three components in detail.

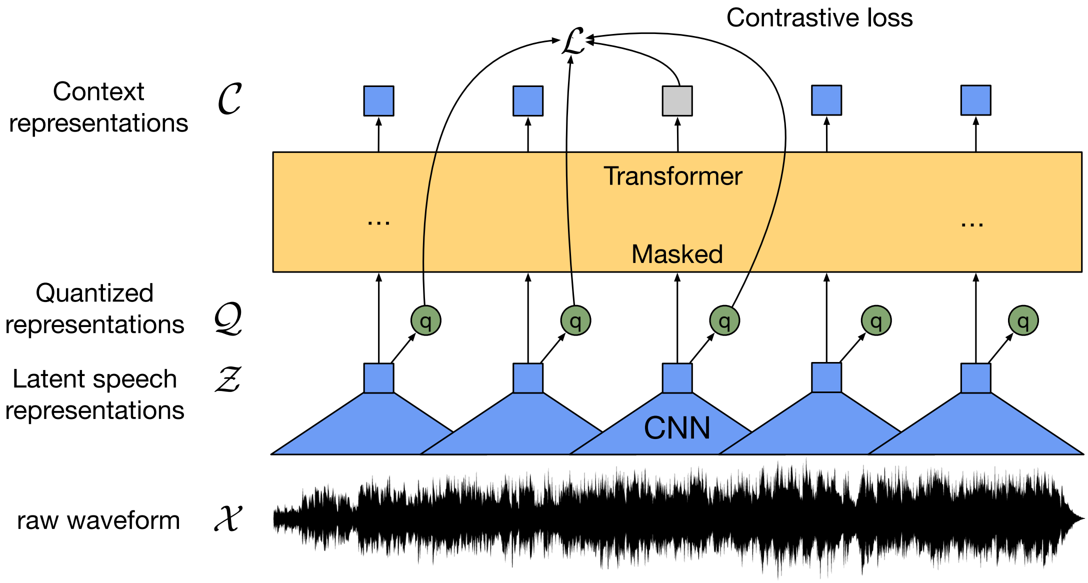

Figure 2.1: A visualization of the network architecture of wav2vec 2.0 [**?**].

2.2.1 Feature encoder

The feature encoder maps the raw audio data (speech recordings) to latent speech representations: $f : \mathcal{X} \to \mathcal{Z}$. Thus, the feature encoder f maps a sequence of audio samples $\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}$ into a sequence of latent feature vectors $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(t)}$.

The audio data is scaled to have zero mean and unit variance before going into the feature encoder. The feature encoder consists of seven convolutional blocks, where each convolutional block contains a temporal[1] convolutional layer, a layer normalization layer, and the GELU activation function.

Each temporal convolutional layer contains 512 channels. The strides of the seven temporal convolutional layers are $(5, 2, 2, 2, 2, 2, 2)$ and the kernel widths are $(10, 3, 3, 3, 3, 2, 2)$. The strides used results in each $\mathbf{z}^{(t)}$ representing 25ms of audio (or 400 input samples), strided by about 20ms.

Layer normalization scales the logits after each convolutional layer to have zero mean and unit variance, which has shown to increase the chances of earlier convergence. GELU has become a popular activation function for NLP related tasks

[1]One-dimensional convolutional layer designed for sequential data.

2.2.2 Quantization module

The quantization module maps the latent speech features into discrete speech units: $h : \mathcal{Z} \to \mathcal{Q}$. Speech is sound, and sound is represented as a continuous function. We would like to use Transformers and so continuous representations will not work. Unlike written language, which can be discretized into tokens such as characters or sub-words, speech does not have natural sub-units [4]. The quantization module is a method in which discrete speech units are automatically learned using product quantization.

To perform product quantization, the quantization module uses G *codebooks*, where each codebook contains V *codebook entries* $\mathbf{e}_1, \ldots, \mathbf{e}_V$.

The following steps describe the process of automatically assigning a discrete speech unit to each latent speech feature $\mathbf{z}^{(t)}$:

1. Transform $\mathbf{z}^{(t)}$ into $\mathbf{l}^{(t)} \in \mathbb{R}^{G \times V}$ using a linear transformation.
2. Choose one codebook entry $\mathbf{e}_g$ for each codebook $g = 1, \ldots, G$, based on the values of $\mathbf{l}^{(t)}$.
3. Concatenate the codebook entries $\mathbf{e}_1, \ldots, \mathbf{e}_G$.
4. Transform the resulting vector into $\mathbf{q}^{(t)} \in \mathbb{R}^f$ using another linear transformation.

The two linear transformations are feed-forward neural networks $\mathrm{FF}_1 : \mathbb{R}^f \to \mathbb{R}^{G \times V}$ and $\mathrm{FF}_2 : \mathbb{R}^d \to \mathbb{R}^f$. In the second step above, the codebook entry $\mathbf{e}_g$ is chosen as the one with the argmax of the logits $\mathbf{l}$. Choosing the codebook entries in this way is non-differentiable. Fortunately, we can use the Gumbel softmax to choose codebook entries in a fully differentiable way. $\mathbf{e}_g$ is chosen as the entry that maximizes

$$p_{g,v} = \frac{\exp\left(\mathbf{l}_{g,v}^{(t)} + n_v\right)/\tau}{\sum_{k=1}^{V} \exp\left(\mathbf{l}_{g,k}^{(t)} + n_k\right)/\tau}, \tag{2.1}$$

where τ is a non-negative temperature, $n = -\log(-\log(u))$, and u are uniform samples from $\mathcal{U}(0, 1)$. During the forward pass, codeword i is chosen by $i = \mathrm{argmax}_j p_{g,j}$ and in the backward pass, the true gradient of the Gumbel softmax outputs is used.

2.2.3 Context network

The context network creates contextualized representations from the feature encoder outputs. The main component of the context network is a Transformer encoder [5]. Due to the popularity of Transformers we have ommited a detailed explanation of the Transformer architecture. Interested readers should refer to [5] as well as guides such as [6].

The following steps describe how the latent feature vectors are processed before being fed into the Transformer encoder.

1. The latent feature vectors are fed into a *feature projection layer* to match the model dimension of the context network.
2. Positional embedding vectors are added to the inputs using *relative positional encoding* [7] instead of absolute positional encoding. The relative positional encoding is implemented using grouped convolution [8].
3. Inputs are fed into the GELU activation function, followed by layer normalization.

The details for the Transformer encoder of the LARGE version of wav2vec 2.0 is as follows:

- Numer of Transformer blocks: $B = 24$.
- Model dimension: $H_m = 1024$.
- Inner dimension: $H_{ff} = 4096$.
- Numer of attention heads: $A = 16$.

2.2.4 Pretraining with wav2vec 2.0

Masking. In Wav2Vec 2.0, the masking process plays a crucial role in pretraining. It involves two hyperparameters: $p = 0.065$ and $M = 10$. The masking is performed as follows:

1. All time steps from the latent speech representation space Z are considered.
2. A proportion p of vectors from the previous step is sampled without replacement. These sampled vectors determine the starting indices.
3. For each starting index i, consecutive M time steps are masked. There may be overlap

between these masked spans.

Training objective. There are two objectives (loss functions) that wav2vec 2.0 optimizes simultaneously. The first loss function is the contrastive loss $\mathcal{L}_m$ which encourages the model to identify the true quantized representation for a masked time step within a set of distractors. The second loss function is the diversity loss $\mathcal{L}_d$ which encourages the model to equally use the codebook entries from the quantization module. The full training objective is given by

$$\mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d, \tag{2.2}$$

where α is a tuned hyperparameter.

Contrastive loss. The contrastive loss is responsible for training the model to predict the correct quantized representation $\mathbf{q}_t$ from a set of candidate representations $\tilde{\mathbf{q}} \in \mathcal{Q}_t$. The set $\mathcal{Q}_t$ includes the target $\mathbf{q}_t$ and K distractors sampled uniformly from other masked time steps. The contrastive loss is given by

$$\mathcal{L}_m = -\log \frac{\exp(\mathrm{sim}(\mathbf{c_t}, \mathbf{q_t})\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathcal{Q}_t} \exp(\mathrm{sim}(\mathbf{c_t}, \tilde{\mathbf{q}})\kappa)}, \tag{2.3}$$

where κ represents a constant temperature, and $\mathrm{sim}(a, b)$ denotes the cosine similarity between context representation c_t and quantized representations q. This loss encourages the model to assign high similarity to the true positive target and penalize high similarity with negative distractors.

Diversity loss. The diversity loss is a regularization technique aimed at promoting the equal use of codebook entries. It is based on entropy and is calculated as:

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^{G} -H(\bar{p}_g) = -\frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v} \tag{2.4}$$

This loss maximizes the entropy of the softmax distribution $\bar{p}_{g,v}$ over codebook entries, encouraging the model to utilize all code words equally.

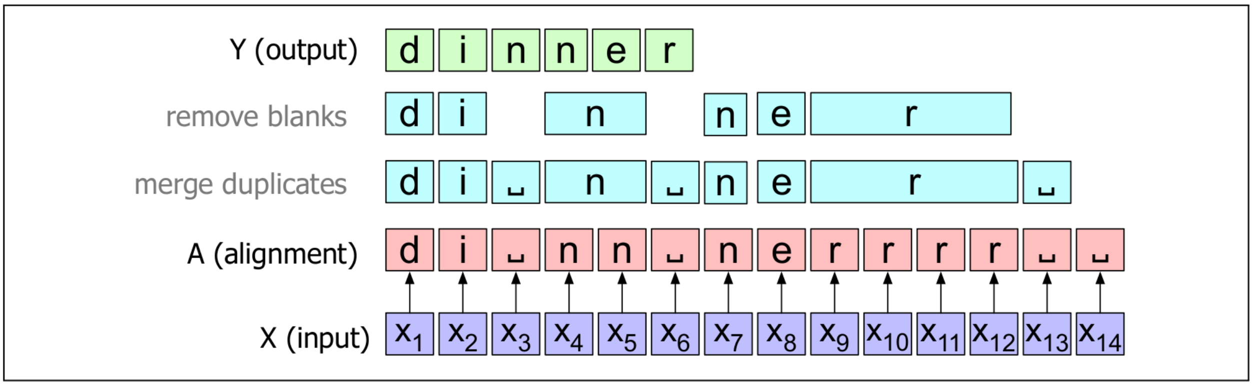

Figure 2.2: A diagram that describes the alignment procedure of CTC [1].

2.2.5 Fine-tuning with wav2vec 2.0

There are three steps required to fine-tune a pretrained wav2vec 2.0 model for automatic speech recognition:

1. Prepare a labeled dataset - a corpus of speech recordings with corresponding transcriptions.
2. Replace the head of the model with a linear layer that has an equal number of output neurons as the number of characters in the set of characters of the transcription data.
3. Optimize the model using the Connectionist Temporal Classification (CTC) loss function.

2.3 CONNECTIONIST TEMPORAL CLASSIFICATION

Connectionist Temporal Classification (CTC) [9] is a loss function (and an algorithm) developed to map a sequence of speech features to a sequence of characters. Our explanation of CTC is heavily based on the automatic speech recognition chapter in [1]. We recommend readers to use Figure 2.2 as a visual aid for our explanation.

Given a sequence of speech features, CTC maps each speech feature to a single character to form a sequence of characters called an *alignment*. Then, a *collapsing function* is used to remove consecutive duplicate characters in the alignment to form an output string (the predicted transcription). The authors of CTC propose the use of a special characters called a *blank*, which is represented by ␣. The blank character accounts for words that contain consecutive duplicate characters (such as "dinner" which contains two consecutive "n" characters). With the addition of the blank character,

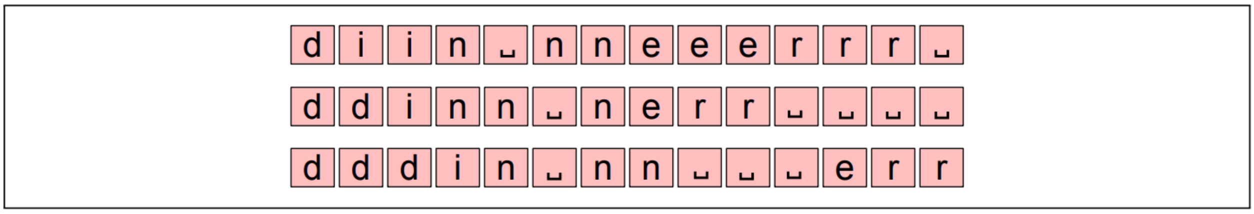

Figure 2.3: Three different alignments that produce the same output string when using the CTC collapsing function [1].

the collapsing function is responsible for removing consecutive duplicate characters and removing all instances of blank characters. Following the notation of [1], we define the collapsing function as a mapping $B : a \to y$ for an alignment a and output string y. Notice that B is many-to-one, since many different alignments can map to the same ouptut string (see Figure 2.3).

In [1], the set of all alignments that map to the same output string is denoted as $B^{-1}(Y)$, where $B^{-}1$ is the inverse of the collapsing function and Y is an output string. This notation is useful for our explanation of CTC inference and training.

2.3.1 CTC Inference

Here we discuss how CTC computes $P_{\text{CTC}(Y|X)}$ which denotes the probability of the output string Y for an input sequence of speech features $X = \{x^{(1)}, \ldots, x^{(T)}\}$. Notice that $P_{\text{CTC}(Y|X)}$ can be rewritten as $P_{\text{CTC}(Y|X)} = \sum\limits_{A \in B^{-1}(Y)} P(A|X)$, which is the sum the probabilities of all possible alignments that produce the output string Y.

CTC makes a strong conditional independence to obtain an expression $P(A|X)$. It assumes that the probability of each alignment character $a^{(t)} \in \{a^{(1)}, \ldots, a^{(T)}\}$ is computed independently of all the other alignment characters: $P(A|X) = \prod\limits_{t=1}^{T} p(a^{(t)}|X)$. Thus we can find the best possible alignment for a given X by choosing the most probable character at each time step using the argmax operator: $a^{(t)} = \text{argmax}_{c \in \mathcal{C}} p_t(c|X)$, where $\mathcal{C}$ represents the vocabulary.

There is a flaw with the greedy approach described above. The problem is that we chose the most likely alignment A, but the most likely alignment may not correspond to the most likely final collapsed output string Y. That's because there are many possible alignments that lead to the same output string, and hence the most likely output string might not correspond to the most probable alignment.

For this reason, the most probable output sequence Y is the one that has, not the single best CTC alignment, but the highest sum over the probability of all its possible alignments:

$$P_{\text{CTC}}(Y|X) = \sum_{A \in B^{-1}(Y)} P(A|X) = \sum_{A \in B^{-1}(Y)} \prod_{t=1}^{T} p(a^{(t)}|X) \tag{2.5}$$

$$\hat{Y} = \text{argmax}_Y P_{\text{CTC}}(Y|X) \tag{2.6}$$

Alas, summing over all alignments is very expensive, because there are many possible alignments. We use dynamic programming to approximate this sum by using a modified version of Viterbi beam search that cleverly keeps in the beam the high-probability alignments that map to the same output string, and sums those as an approximation of the equation above.

Improving CTC with a LM. Because of the strong conditional independence assumption mentioned earlier, CTC does not implicitly learn a language model over the data (unlike the attentionbased encoder-decoder architectures). It is therefore essential when using CTC to interpolate a language model using interpolation weights that are tuned on a validation set:

$$score_{\text{CTC}}(Y|X) = \log\left(P_{\text{CTC}}(Y|X)\right) + \lambda_1 \log\left(P_{\text{LM}}(Y)\right) + \lambda_2 L(Y) \tag{2.7}$$

2.3.2 CTC Training

TODO

2.4 PRETRAINED WAV2VEC 2.0 MODELS

TODO: Explain why I can't perform pretraining

2.4.1 XLS-R

TODO: Provide high-level overview.

CHAPTER 3

EXPERIMENTAL SETUP

3.1 DATA SETS

We collected three datasets from which we created the datasets used for pretraining and finetuning. Note that the datasets for pretraining and finetuning are mutually exclusive. We describe the three datasets in the following paragraphs.

NCHLT dataset Majority is used for pre-training, the rest is used for fine-tuning.

FLEURS dataset Used for fine-tuning

High Quality TTS dataset Used for fine-tuning

3.2 PRE-TRAINING

3.3 FINE-TUNING

3.4 EVALUATION METRICS

Word error rate The word error rate (WER) is equal to the number of character-level errors in the predicted transcript, divided by the number of words in the true transcript. One character-level error is corrected using one of three operations: inserting a new character, deleting an existing character, or substituting an existing character for a new character.

CTC Score/loss ...

CHAPTER 4

RESULTS

CHAPTER 5

CONCLUSIONS

REFERENCES

[1] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

[2] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[3] James W Cooley, Peter AW Lewis, and Peter D Welch. The fast fourier transform and its applications. *IEEE Transactions on Education*, 12(1):27–34, 1969.

[4] Jonathan Bgn. An illustrated tour of wav2vec 2.0, 2021.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[6] Jay Alammar. The illustrated transformer, 2018.

[7] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[9] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.