Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

Speech Recognition for Afrikaans and isiXhosa

by

Lucas Meyer

Research assignment presented in partial fulfilment of the requirements for the degree of Master of Machine Learning and Artificial Intelligence in the Faculty of Applied Mathematics at Stellenbosch University.

Supervisor: Dr H. Kamper

November 2023

Acknowledgements

I would like to thank Dr Herman Kamper for this amazing report template.

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   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

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

22614524 Studentenommer / *Student number*	[illegible] Handtekening / *Signature*
L. Meyer Voorletters en van / *Initials and surname*	**6 November 2023** Datum / *Date*

Abstract

Automatic speech recognition (ASR) is an important technology for education [1], helping people with disabilities [2], and captioning (especially for deaf and hard of hearing people [3]). Several high quality ASR systems for English exist today, such as Siri, Alexa, and Google Assistant. However, ASR remains a challenge for low-resource languages.

In this study, we perform ASR for two South African languages: Afrikaans and isiXhosa. Our approach involves fine-tuning well-known pre-trained models, and using seperately trained language models to boost performance. We propose a sequential fine-tuning approach, in which the model is first fine-tuned on one language, and then fine-tuned on the target language. Our sequential fine-tuning approach is compared to a simple fine-tuning approach, where one language is used to fine-tune the model. Our models are evaluated using the word error rate (WER).

Our experimental results conclude that...

Chapter 1

Introduction

TODO

Chapter 2

Background

This chapter provides an overview of Automatic Speech Recognition (ASR), the general approach of ASR, and the common challenges of ASR. We provide a detailed explanation of wav2vec 2.0, a framework for learning speech features. We discuss how the connectionist temporal classification (CTC) algorithm is used to fine-tune speech features for ASR. The chapter concludes with an explanation of why pre-training is infeasible for our study, and a brief discussion of two already pre-trained wav2vec 2.0 models.

2.1. The automatic speech recognition task

Automatic speech recognition (ASR), also known as speech recognition, is the task of identifying the spoken words for a given speech recording and returning the text, or *transcription*, of the spoken words. For example, given a recording of a speaker[1] saying the sentence "The quick brown fox jumps over the lazy dog", the goal of ASR is to predict each character in the sentence and to make as few mistakes as possible.

The general approach for ASR involves two steps. In the first step, we transform the speech recordings into a higher-dimensional feature representation called *speech features*. Then, in the second step we use supervised learning techniques to map speech features to a sequence of characters that are merged into a sentence.

Computing speech features is useful because recording data is difficult to interpret. Recording data, or audio data, consists of a sequence of floating point numbers called *samples*. The sequence of samples represent amplitude measurements of the recorded sound wave. Note that sound is a continuous wave function, but a recording is a discretized version of the original sound (see Figure 2.1). The issue is that mapping a sequence of samples to a sequence of characters is difficult. Therefore, it is common in ASR and other speech-related tasks to transform audio data into speech features. The traditional approach is to compute speech features by transforming the audio data from the amplitude-time domain to the frequency-time domain using the Fast Fourier Transform (FFT) algorithm [4], [5].

In this study, we discuss an approach to obtain speech features using self-supervised learning. Self-supervised learning is a machine learning paradigm in which unlabeled data

[1]Speaker in this context refers to a person speaking, and not a loudspeaker.

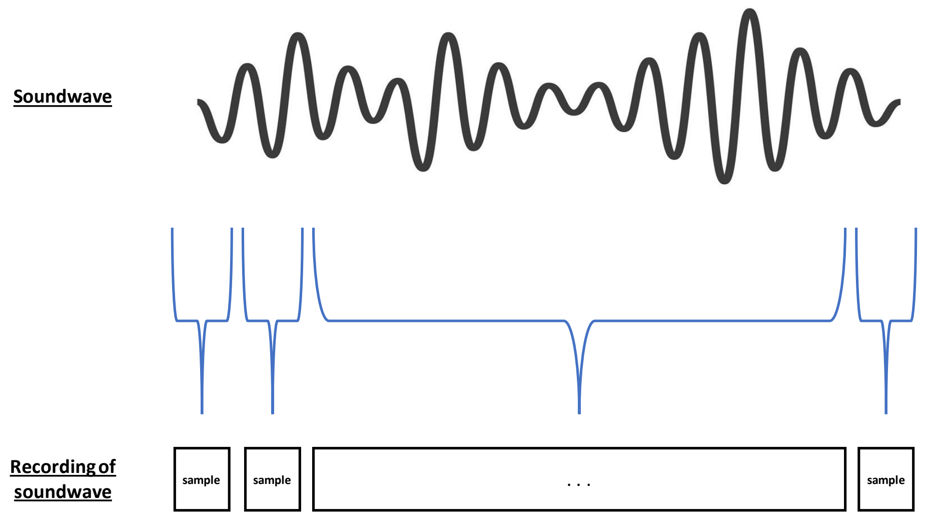


Figure 2.1: A diagram that explains what audio data represents. Note that this is an oversimplified explanation. The intent of the diagram is to explain why it is difficult to interpret raw audio data.

is used for learning, similar to unsupervised learning. Self-supervised learning differs from unsupervised learning in that it leverages the unlabeled data to create artificial labels, often through tasks designed to predict part of the input from other parts. For instance, in the context of speech features, a task of the model may be to predict the next sample of an audio sequence, using the previous samples as context. This approach transforms the problem into a supervised problem without the need for explicit labels. Before explaining this approach, we discuss the importance of constructing good ASR datasets.

2.1.1. Automatic speech recognition data

The first step of creating an ASR model is to prepare audio datasets for training, evaluating, and testing the model. A single dataset entry consists of a speech recording (of a spoken sentence) and the corresponding text of the spoken sentence. The following paragraphs demonstrate why preparing each dataset has a significant effect on the accuracy and generalization performance for ASR models.

Exclusivity around partitioning data. The validation and test set should not contain recordings of voices that also appear in the training set. This is to ensure that the model is not overfitting to the voices in the training set.

The amount of training data. The training set should be as large as possible. A larger training set often leads to better accuracy and generalization performance for ASR models. A small dataset with few unique voices may lead to overfitting to the voices in the dataset.

Voice diversity. The training set should contain a diverse set of speakers. The accent of a speaker, which depends on the gender, age, and ethnicity of the speaker, may influence the generalization ability of ASR models. Generally, male voices have a lower pitch than female voices, and adults have a lower pitch than children.

The difference between read and conversational speech. The training set should contain both read speech and conversational speech. Speakers tend to speak more clearly when reading text from a transcript, in comparison to the speech of a conversation. Recent ASR models obtain very low error rates for recordings of read speech [6]. However, ASR for recordings of conversational speech is still a major challenge [6].

In the following section, we discuss wav2vec 2.0, which is the speech feature extraction technique used in this study.

2.2. wav2vec 2.0

wav2vec 2.0 provides a framework for learning speech features using unlabeled[2] audio data. wav2vec 2.0 can be applied to a variety of speech-related tasks such as ASR, automatic speech translation (AST), and speech classification. It proves to be particularly useful in cases where a lot of unlabeled data is available, but not much labeled data. For ASR, the authors show that using just ten minutes of labeled data and pre-training on 53k hours of unlabeled data achieves 4.8/8.2 WER (3.3) on the clean/other test sets of the Librispeech [7].

The general two-step approach for using wav2vec 2.0 for any speech-related task is the following. In the first step the wav2vec 2.0 model is trained (or "pre-trained") using unlabeled data, which gives you a model that transforms audio data into speech features. In the second step the model is fine-tuned on a downstream task using labeled data. Fine-tuning wav2vec 2.0 (2.2.5) for ASR involves replacing the head of the pre-trained model with an appropriate loss function such as CTC (2.3).

The wav2vec 2.0 architecture is described by the network diagram in Figure 2.2. There are three important components of the 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. Therefore, we explain how to pre-train wav2vec 2.0 after explaining the three components in detail.

[2]Unlabeled audio data simply refers to recordings without transcription text (labels).

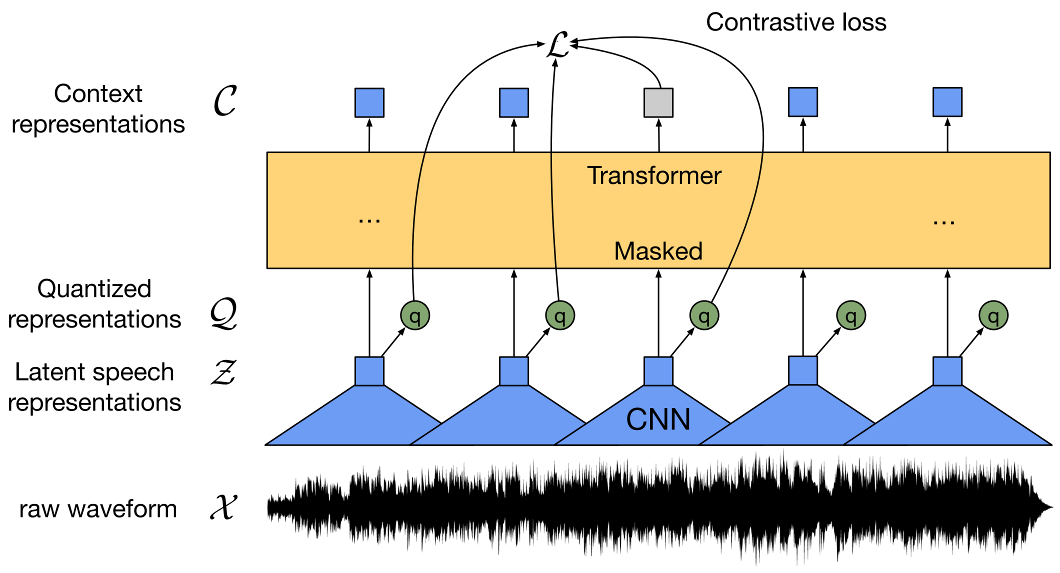


Figure 2.2: A visualization of the network architecture of wav2vec 2.0 [8].

2.2.1. Feature encoder

The feature encoder maps audio data to latent speech representations: $f : \mathcal{X} \to \mathcal{Z}$. In other words, 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 being fed into the feature encoder. The feature encoder consists of seven convolutional blocks, where each convolutional block contains a temporal convolutional layer, layer normalization [9], and the GELU [10] 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 $(10, 3, 3, 3, 3, 2, 2)$. This results in each $\mathbf{z}^{(t)}$ representing 25 ms of audio (or 400 input samples), strided by about 20 ms. Layer normalization scales the logits after each convolutional layer to have zero mean and unit variance, which has been shown to increase the chances of earlier convergence. GELU is an activation function recently used for several NLP-related tasks.

2.2.2. Quantization module

The quantization module maps the latent feature vectors into discrete speech units: $h : \mathcal{Z} \to \mathcal{Q}$. Unlike written language, which can be discretized into tokens such as characters or sub-words, speech does not have natural sub-units [11]. This is because speech is a sound wave, which is a continuous function of time. 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 feature vector $\mathbf{z}^{(t)}$:

1. Transform $\mathbf{z}^{(t)}$ into $\mathbf{l}^{(t)} \in \mathbb{R}^{G \times V}$ using a linear transformation (feed-forward neural network).

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 (feed-forward neural network).

The two linear transformations are feed-forward neural networks $\text{FF}_1 : \mathbb{R}^f \to \mathbb{R}^{G \times V}$ and $\text{FF}_2 : \mathbb{R}^d \to \mathbb{R}^f$. 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, $n = -\log\left(-\log\left(u\right)\right)$, and u are uniform samples from $\mathcal{U}(0, 1)$. During the forward pass the codeword i is chosen by $i = \text{argmax}_j p_{g,j}$, and during the backward pass the true gradient of the Gumbel softmax outputs is used.

2.2.3. Context network

The context network maps the latent feature vectors into contextualized representations: $g : \mathcal{Z} \to \mathcal{C}$. The main component of the context network is a Transformer encoder [12]. Due to the popularity of Transformers, we omit the details of the Transformer architecture. However, we recommend illustrated guides such as [13] for more information about the Transformer architecture.

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* [14] instead of absolute positional encoding. The relative positional encoding is implemented using grouped convolution [15].

3. Inputs are fed into the GELU activation function, followed by layer normalization.

The output after processing the latent feature vectorsaccording to the steps above are fed into the Transformer encoder, which results in a sequence of contextualized representations $\mathbf{c}^{(1)}, \dots \mathbf{c}^{(T)}$. The details for the Transformer encoder of the LARGE version of wav2vec 2.0 are:

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

2.2.4. Pre-training with wav2vec 2.0

Here we discuss how the three components are used to perform pre-training with wav2vec 2.0. The main objective of pre-training is a contrastive task, which involves masking a proportion of the output of the feature encoder $(\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(t)})$.

A proportion of the latent feature vectors are masked and replaced by a shared feature vector $\mathbf{z}_{\mathrm{M}}$, before being fed into the context network. Note, that the inputs to the quantization module are not masked. A proportion p of starting indices in $\{1, \dots, t\}$ are randomly sampled. Then, for each starting index i, consecutive M time steps are masked (where spans may overlap).

The dimensionality of the output of the context network $(\mathbf{c}^{(1)}, \dots \mathbf{c}^{(T)})$ matches the dimensionality of the output of the quantization module $(\mathbf{q}^{(1)}, \dots \mathbf{q}^{(T)})$. The contrastive task involves predicting the corresponding quantized target $\mathbf{q}^{(t)}$ for a context vector $\mathbf{c}^{(t)}$. Additionaly, 100 negative distractors are uniformly sampled for each masked position. The model is encouraged to minimize the distance (cosine similarity) between $\mathbf{q}^{(t)}$ and $\mathbf{c}^{(t)}$, and to maximize the distance between $\mathbf{q}^{(t)}$ and the negative distractors. A detailed explanation of the training objective follows below.

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

$$\mathcal{L}_m = -\log \frac{\exp(\text{sim}(\mathbf{c_t}, \mathbf{q_t})\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathcal{Q}_t} \exp(\text{sim}(\mathbf{c_t}, \tilde{\mathbf{q}})\kappa)}, \tag{2.2}$$

where κ represents a constant temperature, and $\text{sim}(a, b)$ 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:

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^{G} -H(\bar{p}_g) = -\frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v}. \tag{2.3}$$

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.

2.2.5. Fine-tuning wav2vec 2.0 for automatic speech recognition

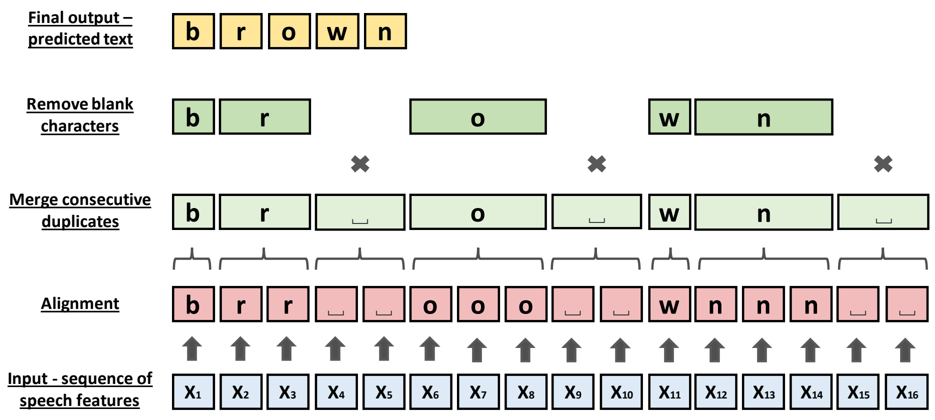
To fine-tune a pre-trained wav2vec 2.0 model for ASR, it is required to prepare a labeled dataset. Typically, the head of the model is replaced with a linear layer that has an equal number of output neurons as the size of the vocabulary[3] of the labeled dataset. The model is optimized using a supervised learning algorithm called connectionist temporal classification.

2.3. Connectionist temporal classification

Connectionist temporal classification (CTC) [16] is an algorithm developed to map a sequence of speech features to a sequence of characters. Our explanation of CTC is heavily based on the speech recognition and text-to-speech chapter in [6]. We recommend readers to use Figure 2.3 as a visual aid for our explanation.

Given a sequence of speech features, CTC maps each speech feature vector to a single character, which results in a sequence of characters known as an *alignment*. Then, a *collapsing function* is used to merge consecutive duplicate characters in the alignment. The

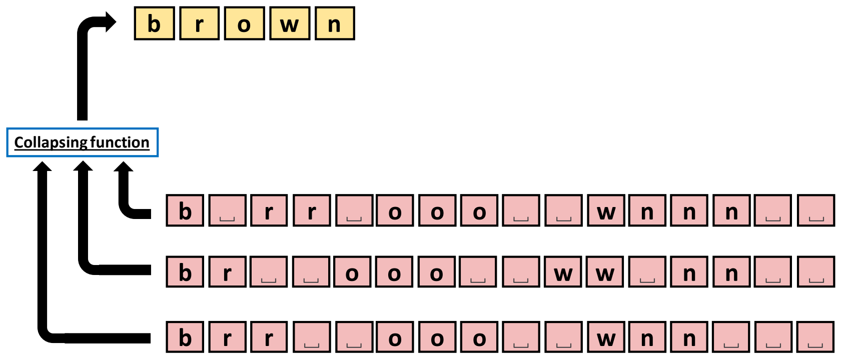
[3]The vocabulary refers to the set of unique characters contained in the transcriptions (labels) of the dataset.



Final output – predicted text
b r o w n
Remove blank characters
b r o w n
Merge consecutive duplicates
b r ␣ o ␣ w n ␣
Alignment
b r r ␣ ␣ o o o ␣ ␣ w n n n ␣ ␣
Input - sequence of speech features
X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15 X16

Figure 2.3: A diagram that describes the alignment procedure of CTC [6].

authors of CTC propose the use of a special character called a *blank*, which is represented by "␣". The blank character accounts for words that contain consecutive duplicate characters (such as the word "dinner", which contains two consecutive "n" characters). With the addition of the blank character, the collapsing function is responsible for merging consecutive duplicate characters and removing all instances of blank characters. Following the notation of [6], we define the collapsing function as a mapping $B : A \to Y$ for an alignment A and output text Y. Notice that B is many-to-one, since many different alignments can map to the same output text (see Figure 2.4).



b r o w n
Collapsing function
b ␣ r r ␣ o o o ␣ ␣ w n n n ␣ ␣
b r ␣ ␣ o o o ␣ ␣ w w ␣ n n ␣ ␣
b r r ␣ ␣ o o o ␣ ␣ w n n ␣ ␣ ␣

Figure 2.4: Three different alignments that produce the same output text when using the CTC collapsing function [6].

In [6], the set of all alignments that map to the same output text Y is denoted as $B^{-1}(Y)$, where B^{-1} is the inverse of the collapsing function. This notation is useful for our explanation of CTC inference and training.

2.3.1. Connectionist temporal classification inference

Here we discuss how CTC models the probability of the output text Y for a sequence of speech features $X = \{x^{(1)}, \ldots, x^{(T)}\}$, denoted by $P_{\text{CTC}}(Y|X)$. The conditional probability above can be expressed as the summation over the probabilities of all possible alignments that produce the output text Y. Hence, we obtain the expression:

$$P_{\text{CTC}}(Y|X) = \sum_{A \in B^{-1}(Y)} P(A|X). \tag{2.4}$$

We still need an expression for $P(A|X)$. To compute $P(A|X)$, CTC makes a strong conditional independence assumption. It assumes that each alignment character $a^{(t)} \in \{a^{(1)}, \ldots, a^{(T)}\}$ is computed independently of all the other alignment characters. Hence, we obtain the expression:

$$P(A|X) = \prod_{t=1}^{T} p(a^{(t)}|x^{(t)}). \tag{2.5}$$

Now we can find the best possible alignment for a given X, by simply choosing the most probable character at each time step. We compute each alignment character $\{a^{(1)}, \ldots, a^{(T)}\}$, apply the collapsing function, and obtain the output text Y. However, there is an issue with the greedy approach described above. The issue is that the most probable output text $\hat{Y}$ may not correspond with the most probable alignment sequence $\{\hat{a}^{(1)}, \ldots, \hat{a}^{(T)}\}$. The reason for this is that there are many possible alignments that lead to the same output text. Therefore, the most probable output text $\hat{Y}$, for a given X, corresponds to the maximum summation over the probabilities of all its possible alignments:

$$\hat{Y} = \text{argmax}_Y P_{\text{CTC}}(Y|X) = \text{argmax}_Y \left(\sum_{A \in B^{-1}(Y)} P(A|X) \right) \tag{2.6}$$

$$= \text{argmax}_Y \left(\sum_{A \in B^{-1}(Y)} \prod_{t=1}^{T} p(a^{(t)}|x^{(t)}) \right). \tag{2.7}$$

There are many possible alignments, and summing over all the possible alignments is expensive and infeasible. We can use dynamic programming to approximate this sum, by using a modified version of Viterbi beam search [17]. The beam search returns a user-specified number of potential output texts, and a loss function is used to score each output text. The text with the best score is chosen as the final prediction.

2.3.2. Connectionist temporal classification training

The CTC training objective function is the negative log-likelihood. Formally, the loss for a dataset D is equal to the summation of the negative log-likelihoods of the correct output

Y for each corresponding input X:

$$L_{\text{CTC}} = \sum_{(X,Y)\in D} -\log P_{\text{CTC}}(Y|X) \tag{2.8}$$

$$= \sum_{(X,Y)\in D} -\log \left(\sum_{A\in B^{-1}(Y)} \prod_{t=1}^{T} p(a^{(t)}|x^{(t)}) \right). \tag{2.9}$$

Once again, we cannot compute a summation over all the possible alignments. The summation is efficiently computed using a variant of the *forward-backward algorithm.* A detailed explanation of the forward-backward algorithm used for training CTC is given in [17].

2.3.3. Improving connectionist temporal classification with a language model.

Because of the strong conditional independence assumption mentioned earlier, CTC does not implicitly learn a language model over the data. Therefore, the CTC algorithm commonly predicts sentences that contain obvious spelling mistakes. We can mitigate this issue by using a seperately trained language model (LM). This is done by adding an additional term in the CTC loss function with interpolation weights:

$$\text{score}_{\text{CTC}}(Y|X) = \log\left(P_{\text{CTC}}(Y|X)\right) + \lambda_1 \log\left(P_{\text{LM}}(Y)\right) + \lambda_2 L(Y). \tag{2.10}$$

TODO: briefly unpack what $P_{\text{LM}}(Y)$ is, and how it would be constructed

We have now discussed how to create an ASR model that is pre-trained using wav2vec 2.0 and fine-tuned using CTC. However, in the following section we discuss an issue with this approach for our particular study.

2.4. The difficulty of pre-training using wav2vec 2.0

Pre-training with wav2vec 2.0 using unlabeled data is infeasible for our study because:

1. Pre-training with wav2vec 2.0 is known to be quite unstable (see "NOTE 1" of [18]).
2. Performing one experiment (run) of the large-scale wav2vec 2.0 model, using 8 GPU V100s (16 GB RAM each), takes 7 days to finish [18].

At the time of conducting this study, we did not have access to the computational resources required to perform pre-training experiments. Therefore, our study focuses on fine-tuning already pre-trained wav2vec 2.0 models, rather than performing pre-training and then fine-tuning. In this section, we discuss the pre-trained models used in this study: Wav2Vec2-Large and XLS-R. Wav2Vec2-Large [8] is a base-sized wav2vec 2.0 model pre-trained on 960 hours of English speech recordings.

2.4.1. XLS-R

XLS-R [19] is a large-scale wav2vec 2.0 model trained on 128 different languages to learn *cross-lingual* speech representations. XLS-R attempts to build better representations for low-resource languages by leveraging cross-lingual transfer from high-resource languages such as English.

The model is trained using batches that contain samples from multiple languages L. Batches are sampled from a probability distribution $p_l \sim \left(\frac{n_l}{N}\right)^a$ where:

- $l \in \{1, \ldots, L\}$ represents each language.
- N is the number of hours of all the unlabeled data and n_l is the number of hours of unlabeled data for the language l.
- α controls how much data is chosen from the high-resource and low-resource languages.

TODO: Give an intution for why this strategy is used.

The total number of hours of all the unlabeled data combined is about 436k hours. The authors use data from 24 high-resource languages, 17 mid-resource languages, and 88 low-resource languages. The high-resource languages contain more than 1k hours of data each, the mid-resource languages contain more than 100 hours (but less than 1k hours) of data each, and the low-resource languages contain less than 100 hours of data each.

XLS-R is evaluated on a wide range of downstream tasks: ASR, AST, and speech classification which includes language identification (LID) and speaker identification (SID). XLS-R achieves better evaluation results than the previous state-of-the-art for several benchmark datasets for every downstream task, which demonstrates the generalization ability of XLS-R.

Chapter 3

Experimental Setup

This chapter provides our research questions, and how we attempt to answer these questions using the results of different experiments. We discuss the datasets used for our ASR models, as well as the data for our language model (LM). The chapter concludes with a discussion of the metrics used in this study to evaluate our ASR models.

3.1. Research questions and experiments

The main research question we attempt to answer in this study is: "Does a sequential fine-tuning strategy, where the model is first fine-tuned on (for example) Afrikaans data and subsequently on isiXhosa data, result in better accuracy and generalization performance compared to fine-tuning on isiXhosa data alone?". We investigate this research question by fine-tuning Wav2Vec2-Large and XLS-R using:

1. Only Afrikaans data.
2. Only isiXhosa data.
3. isiXhosa data, and then Afrikaans data.
4. Afrikaans data, and then isiXhosa data.

Experiments (1) and (3), and experiments (2) and (4) are compared using WER (3.3). Additionally, each model is also evaluated with and without the use of a 5-gram LM with Kneser-Ney smoothing [20], [21].

3.2. Data sets

We use three different data sources to create an Afrikaans dataset and an isiXhosa dataset, which contains labeled speech recordings. The three data sources are briefly explained in the paragraphs below.

NCHLT dataset. The NCHLT [22] dataset contains speech recordings for the eleven official South African languages. The dataset contains approximately 200 speakers per language. The NCHLT recordings (on average) are the shortest of the three datasets, with most recordings being between 2 and 6 seconds. Based on brief inspection, the transcription texts of this dataset contain very few words, and rarely contain full sentences.

FLEURS dataset. The FLEURS [23] dataset contains speech recordings for 102 different languages, including Afrikaans, isiXhosa, and isiZulu. Each language has its own training, validation, and test set split. No information is given about the number of speakers for each language. The FLEURS recordings (on average) are the longest of the three datasets, with most recordings being between 7 and 20 seconds. Based on brief inspection, the transcription texts of this dataset contain full sentences.

High-quality TTS dataset. The High-quality text-to-speech (TTSs) [24] dataset contains high quality transcribed audio data for four South African languages: Afrikaans, Sesotho, Setswana and isiXhosa. There are nine Afrikaans speakers and 12 isiXhosa speakers. The duration of most recordings are between 5 and 10 seconds. Based on brief inspection, the transcription texts of this dataset contain mostly full sentences, and a few that contain short phrases.

We select the recordings for the Afrikaans and isiXhosa dataset to have similar distributions for the durations of the recordings. Figure 3.1 describes the duration histograms of the two datasets. Despite the large number of short recordings, we omit most of the NCHLT dataset, in favor of longer recordings with full sentence transcription texts.

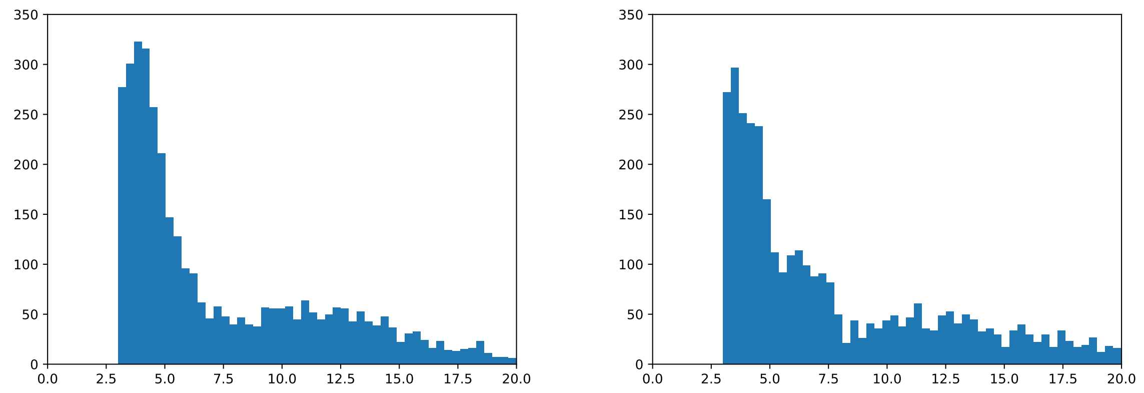


Figure 3.1: The histograms of the recording durations of the Afrikaans dataset (left) and the isiXhosa dataset (right)

We ensure that the validation and test set splits do not contain recordings of speakers that appear in the training set split. Both of the datasets contain approximately 7.5

hours of speech recordings. All of the speech recordings are downsampled to a sample rate of $16,000$ Hz, which is the expected sample rate of Wav2Vec2-Large and XLS-R. The transcription texts are pre-processed in the same way as the LM data, which is explained in the following section.

3.2.1. Language model data

The data used for training the LM (2.3.3) consists of Wikipedia dumps. The text is extracted from the dumps using a tool called the WikiExtractor [25]. The text is pre-processed by converting all characters to lowercase, removing all characters that are not used in the Afrikaans and isiXhosa language, and removing punctuation marks and other special characters.

3.3. 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.

Chapter 4

Results

Table 4.1: The WER for different models evaluated on the validation/test data of the Afrikaans dataset. The hyperlinks for each model provide more information about the hyperparameters and training results.

	Word error rate (WER)	
Model	Validation	Test
Run 1	0.3789	0.3841
Run 1 (with LM)	0.2753	0.2803
Run 2	0.3798	0.3801
Run 2 (with LM)	0.3149	0.3119

Table 4.2: The WER for different models evaluated on the validation/test data of the isiXhosa dataset. The hyperlinks for each model provide more information about the hyperparameters and training results.

	Word error rate (WER)	
Model	Validation	Test
Run 1	?	?
Run 1 (with LM)	?	?
Run 2	0.5436	0.5494
Run 2 (with LM)	0.4373	0.4211

The two isiZulu runs (intended for fine-tuning again):

- Run 1
- Run 2

Chapter 5

Summary and Conclusion

TODO

Bibliography

[1] M. Wald, "Using automatic speech recognition to enhance education for all students: Turning a vision into reality," in *Proceedings Frontiers in Education 35th Annual Conference*. IEEE, 2005, pp. S3G–S3G.

[2] N. Terbeh, M. Labidi, and M. Zrigui, "Automatic speech correction: A step to speech recognition for people with disabilities," in *Fourth International Conference on Information and Communication Technology and Accessibility (ICTA)*. IEEE, 2013, pp. 1–6.

[3] M. Wald, "Captioning for deaf and hard of hearing people by editing automatic speech recognition in real time," in *International Conference on Computers for Handicapped Persons*. Springer, 2006, pp. 683–690.

[4] W. T. Cochran, J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch, "What is the fast fourier transform?" *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664–1674, 1967.

[5] J. W. Cooley, P. A. Lewis, and P. D. Welch, "The fast fourier transform and its applications," *IEEE Transactions on Education*, vol. 12, no. 1, pp. 27–34, 1969.

[6] D. Jurafsky and J. H. Martin, "Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition."

[7] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[8] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," 2020.

[9] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[10] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[11] J. Bgn. (2021) An illustrated tour of wav2vec 2.0. [Online]. Available: https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[13] J. Alammar. (2018) The illustrated transformer. [Online]. Available: https://jalammar.github.io/illustrated-transformer/

[14] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *CoRR*, vol. abs/1803.02155, 2018. [Online]. Available: http://arxiv.org/abs/1803.02155

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[16] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[17] A. Hannun, "Sequence modeling with ctc," *Distill*, 2017, https://distill.pub/2017/ctc.

[18] P. von Platen and other contributors. (2021) Speech recognition pre-training. [Online]. Available: https://github.com/huggingface/transformers/tree/main/examples/pytorch/speech-pretraining

[19] A. Babu, C. Wang, A. Tjandra, K. Lakhotia, Q. Xu, N. Goyal, K. Singh, P. von Platen, Y. Saraf, J. Pino *et al.*, "Xls-r: Self-supervised cross-lingual speech representation learning at scale," *arXiv preprint arXiv:2111.09296*, 2021.

[20] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948. [Online]. Available: http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

[21] H. Ney, U. Essen, and R. Kneser, "On structuring probabilistic dependences in stochastic language modelling," *Computer Speech & Language*, vol. 8, no. 1, pp. 1–38, 1994.

[22] E. Barnard, M. H. Davel, C. van Heerden, F. De Wet, and J. Badenhorst, "The nchlt speech corpus of the south african languages." Workshop Spoken Language Technologies for Under-resourced Languages (SLTU), 2014.

[23] A. Conneau, M. Ma, S. Khanuja, Y. Zhang, V. Axelrod, S. Dalmia, J. Riesa, C. Rivera, and A. Bapna, “Fleurs: Few-shot learning evaluation of universal representations of speech,” *arXiv preprint arXiv:2205.12446*, 2022. [Online]. Available: https://arxiv.org/abs/2205.12446

[24] D. van Niekerk, C. van Heerden, M. Davel, N. Kleynhans, O. Kjartansson, M. Jansche, and L. Ha, “Rapid development of tts corpora for four south african languages,” in *Proc. Interspeech 2017*, 2017, pp. 2178–2182. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2017-1139

[25] G. Attardi, “Wikiextractor,” https://github.com/attardi/wikiextractor, 2015.