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.

Stellenbosch UNIVERSITY IYUNIVESITHI UNIVERSITEIT

forward together
sonke siya phambili
saam vorentoe

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

TODO: The English abstract.

Uittreksel

TODO: Die Afrikaanse uittreksel.

Chapter 1

Introduction

TODO

Chapter 2

Background

2.1. The Automatic Speech Recognition Task

Automatic speech recognition (ASR), also known as speech recognition, involves predicting the sentence for a given speech recording. For example, given a recording of a person 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 convert speech audio data into a higher-dimensional feature representation called *speech features*. Then, in the second step we use supervised learning techniques to map the feature vectors obtained in the first step to a sequence of characters that make up the predicted transcription.

Computing speech features is useful because audio data is difficult to interpret. A recording, or audio file, consists of a one-dimensional array of integers 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 that was recorded. This is explained by Figure 2.1 below, which is an oversimplification of what an audio file represents.

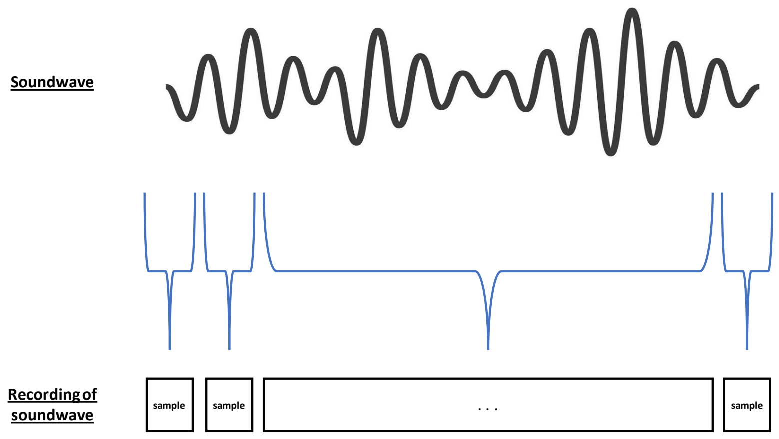

Figure 2.1: A diagram that explains how a recording (audio file) is a discretized version of the soundwave being recorded.

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 first convert audio data into speech features. 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 [1], [2].

In this study, we will discuss an approach to obtain speech features using self-supervised learning. Before discussing this approach, we would like to discuss the significance of choosing the speech audio data used to train and evaluate ASR models.

2.1.1. Speech recognition data

The first step of creating an ASR model is to prepare a training, validation, and test dataset. 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 the choice of training and validation data has a significant effect on the accuracy and generalization ability 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. We believe that humans tend to speak more clearly when reading text from a transcript, in comparison to conversational speech. Recent ASR models obtain very low error rates for recordings of read speech [3]. However, ASR for recordings of conversational speech is still a major challenge [3].

Differences in accents. The accent of the speaker, which depends on the gender, age, and ethnicity of the speaker may also 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 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 audio quality of speech recordings.

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

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 proves 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 pre-training on 53K hours of unlabeled data still achieves 4.8/8.2 WER on the clean/other test sets of Librispeech [4].

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 using unlabeled data, which will give you a model that converts 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 speech recognition involves replacing the head of the pre-trained model with an appropriate loss function such as CTC (2.3).

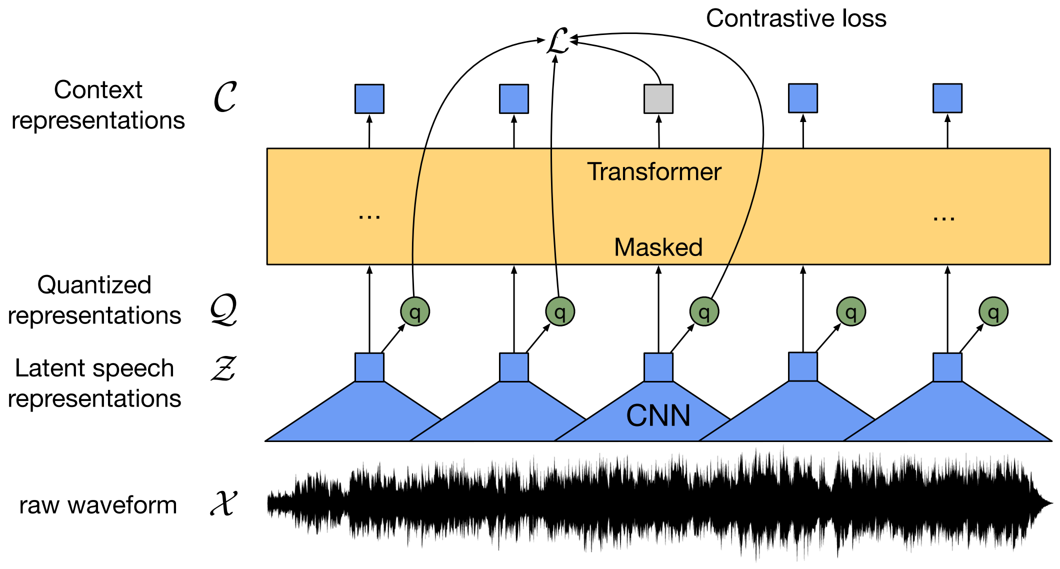

Figure 2.2: A visualization of the network architecture of wav2vec 2.0 [5]. TODO: create own version of this.

The wav2vec 2.0 architecture is described by the network diagram in Figure 2.2. 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.

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 $\mathbf{x}^{(1)}, \dots \mathbf{x}^{(N)}$ into a sequence of latent feature vectors $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(t)}$.

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

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 [6]. 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* $\mathbf{e}_1, \dots, \mathbf{e}_V$.

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 $g = 1, \dots, G$, based on the values of $\mathbf{l}^{(t)}$.

3. Concatenate the codebook entries $\mathbf{e}_1, \dots, \mathbf{e}_G$.

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 $\text{FF}_1 : \mathbb{R}^f \to \mathbb{R}^{G \times V}$ 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

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

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

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

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, codeword i is chosen by $i = \text{argmax}_j p_{g,j}$ 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 [7]. Due to the popularity of Transformers we have ommited a detailed explanation of the Transformer architecture. Interested readers should refer to [7] as well as guides such as [8].

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* [9] instead of absolute positional encoding. The relative positional encoding is implemented using grouped convolution [10].
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:

- 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

Masking. In Wav2Vec 2.0, the masking process plays a crucial role in pre-training. 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 $\mathcal{L} = \mathcal{L}_m + \alpha\mathcal{L}_d$, 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

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

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:

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

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 with wav2vec 2.0

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

1. Prepare a labeled dataset - a corpus of speech recordings with corresponding sentences.
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 sentence data.
3. Optimize the model using the Connectionist Temporal Classification (CTC) algorithm.

2.3. Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) [11] is an algorithm developed to map a sequence of speech features (vectors) to a sequence of characters. Our explanation of CTC is heavily based on the automatic speech recognition chapter in [3]. We recommend readers to use Figure 2.3 as a visual aid for our explanation.

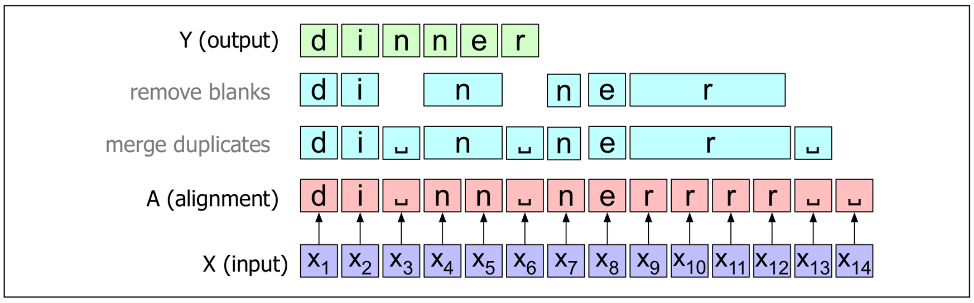

Figure 2.3: A diagram that describes the alignment procedure of CTC [3]. TODO: create own version of the same visual explanation and use the running example of "The quick brown fox ...".

Given a sequence of speech features, CTC maps each speech feature to a single character, which results in a sequence of characters called an *alignment*. Then, a *collapsing function* is used to merge consecutive duplicate characters in the alignment. 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, the collapsing function is responsible for merging consecutive duplicate characters and removing all instances of blank characters. Following the notation of [3], 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 output string (see Figure 2.4).

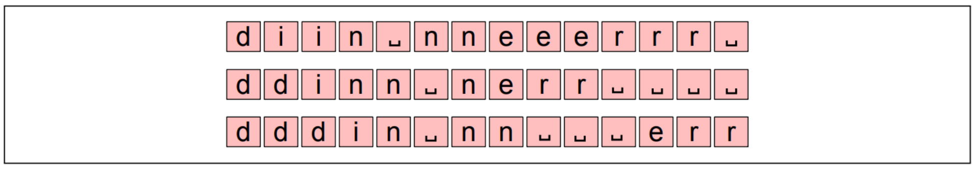
d i i n ␣ n n e e e r r r ␣
d d i n n ␣ n e r r ␣ ␣ ␣ ␣
d d d i n ␣ n n ␣ ␣ ␣ e r r

Figure 2.4: Three different alignments that produce the same output string when using the CTC collapsing function [3].

In [3], the set of all alignments that map to the same output string 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. CTC Inference

Here we discuss how CTC models the probability of the output string Y for a sequence of speech features $X = \{x^{(1)}, \dots, x^{(T)}\}$, 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 string 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 $a^{(t)} \in \{a^{(1)}, \dots, a^{(T)}\}$ 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 greedily choosing the most probable character at each time step. We compute each alignment character $\{\hat{a^{(1)}}, \dots, \hat{a^{(T)}}\}$, apply the collapsing function, and obtain the output string Y. However, there is an issue with the greedy approach described above. The issue is that the most probable output string $\hat{Y}$ may not correspond with the most probable alignment sequence $\{\hat{a^{(1)}}, \dots, \hat{a^{(T)}}\}$. The reason for this is that there are many possible alignments that lead to the same output string. Therefore, the most probable output string $\hat{Y}$ for a given X corresponds to the highest 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. The beam search returns a user-specified number of potential output strings, and a loss function is used to score each output string. The string with the best score is chosen as the final prediction.

2.3.2. CTC Training

To train a CTC-based ASR (Automatic Speech Recognition) system, the negative log-likelihood loss is employed in conjunction with a specialized CTC loss function. Formally, the loss for a dataset D is represented as the cumulative sum of the negative log-likelihoods of the correct output Y for each corresponding input X:

$$\begin{aligned} L_{\text{CTC}} &= \sum_{(X,Y)\in D} -\log P_{\text{CTC}}(Y|X) \qquad (2.8) \\ &= \sum_{(X,Y)\in D} -\log \left(\sum_{A\in B^{-1}(Y)} \prod_{t=1}^{T} p(a^{(t)}|x^{(t)}) \right). \qquad (2.9) \end{aligned}$$

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 [12].

2.3.3. Improving CTC 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, it is typical that the CTC algorithm predicts a sentence that has obvious spelling mistakes. We can mitigate this issue by using a seperate language model (LM). This is done by adding an additional term in the CTC loss function and using interpolation weights that are tuned on a validation set:

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

We have now discussed everything we need to know to create a self-supervised learning-based ASR model. First, we pre-train our model using wav2vec 2.0 and a large corpus of unlabeled data, and then we fine-tune our model using CTC. However, in the following section will discuss a major issue with this approach. We will also discuss how our approach differs from the approach above.

2.4. Fine-tuning existing wav2vec 2.0 models

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 [13]).
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 [13].

At the time of conducting this study, we did not have access to the computational resources required to perform pre-training experiments. This is why our study focuses on fine-tuning existing pre-trained wav2vec 2.0 models, rather than performing pre-training and then fine-tuning. In this section, we discuss two pre-trained models used in this study: Wav2Vec2-Large and XLS-R.

2.4.1. Wav2Vec2-Large

TODO: Need to make sure about the choice of model here. One option: Wav2Vec2-Large.

2.4.2. XLS-R

XLS-R [14] 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.

The total number of hours of all the unlabeled data combined is about 436K hours. The authors used 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, automatic speech translation (AST), and speech classification which includes language identification and speaker identification. The evaluation results at the time improved on the previous state of the art for several benchmarks for every downstream task. This demonstrates the generalization ability of XLS-R.

Chapter 3

Experimental Setup

3.1. Experiments

TODO:

- Explain each experiment.

3.2. Data sets

TODO:

- Explain that we combined three different datasets to create two datasets for Afrikaans and isiXhosa.
- Provide huggingface links to datasets.
- Describe each of the three datasets with some very basic statistics such as counts and means.
- Explain how we made sure that our final datasets have similar duration histograms and total duration (just less than 7.5 hours).
- Discuss preprocessing steps of audio (SR = 16000). Dicuss transcription preprocessing in LM subsection.

3.2.1. Language model data

TODO:

- Explain where we got the Wikipedia data.
- Cite WikiExtractor.
- Explain text preprocessing steps.

3.3. Models and Hyperparameters

TODO: I am not really sure what to say here.

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.

Chapter 4

Results

TODO

Chapter 5

Summary and Conclusion

TODO

Bibliography

[1] 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.

[2] 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.

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

[4] 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.

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

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

[7] 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

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

[9] 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

[10] 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

[11] 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.

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

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

[14] 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.