Sivert Rosberg

Better coding for learning speech representations

Master's thesis in Electronic Systems Design and Innovation
Supervisor: Torbjørn Karl Svendsen
June 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of Science and Technology

Sivert Rosberg

Better coding for learning speech representations

Master’s thesis in Electronic Systems Design and Innovation
Supervisor: Torbjørn Karl Svendsen
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Better coding for learning speech representations

Sivert Rosberg

Masters thesis in Electronic Systems Design
Supervisor: Torbjørn Karl Svendsen

Department of Electronic Systems
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Trondheim, Norway
June 2022

Abstract

In this thesis, the possibility of using Non-autoregressive Predictive Coding (NPC) for learning speech representations is investigated. NPC is a self-supervised deep learning method that, as opposed to other common self-supervised methods, is not autoregressive, and can therefore be trained faster. Three different NPC models are trained, one English, one, Norwegian and one adaptation model trained into Norwegian with the English model as a basis.

Examination of the learned representations shows that the representations are on a sub-phonemic level and many different vectors are used to represent different stages of one phoneme. Different speakers have some overlap in the representations used for the same phonemes, but it is not completely equal.

Testing the learned representations with a phoneme recognizer shows that in these experiments the NPC features were not able to outperform Mel Frequency Cepstral Coefficients (MFCC), neither the English, the Norwegian, nor the adaptation model. The adaptation model was able to outperform the the Norwegian model and that can mean that adaptation training might be useful for speech representations in languages with small speech databases.

Acknowledgements

First, I want to thank my supervisor for this thesis, Professor Torbjørn Karl Svendsen, for the invaluable support he provided, both with theoretical knowledge about the subjects and practical computer issues, during the course of writing this thesis.

I also want to thank Janine Rugayan for her helping me with coding and providing me with access to useful code and libraries useful for this thesis and her experience with a similar masters thesis.

Lastly, I want to thank my wife and my family for their support during the process of writing this thesis and conducting the experiments.

Contents

Figures

Tables

Chapter 1

Introduction

Speech recognition systems as well as speaker recognition systems are increasingly becoming more and more accessible to the common user today. Due to the increase in accessibility of mobile phones, computers, smart devices and more computing power and storage, these systems will only increase in usage. Because of these technological advances, new options and innovations have become available for speech recognition systems. One of those are the possibility to use machine learning and Deep Neural Networks (DNN) in speech recognition.

Earlier, MFCCs (Mel Frequency Cepstral Coefficients) have been seen as the gold standard for speech features, but throughout the last years, several new options have emerged. Non-Autoregressive Predictive Coding (NPC) is one such method utilizing deep neural networks with the perspective of learning speech representations in a better way than before.

The main goals in this thesis are to examine the properties of the learned representations and latent representations by using NPC and to test their performance in a phoneme recognition system. Their performance will be compared to MFCCs. Both an English and a Norwegian NPC model will be trained. In addition, adaptation training between different languages will also be examined, and its performance will be assessed in an phoneme recognizer as well.

Chapter 2

Theoretical background

2.1 Speech recognition systems

An Automatic Speech Recognition (ASR) system is a system which tries to digitally process natural language and recognize the spoken words. It differs from an Automatic *Speaker* Recognition system by focusing on the content and meaning of the speech as opposed to differentiating between speakers or verifying a known speaker. A simplified overview of the main elements in a ASR system is shown in Figure 2.1.

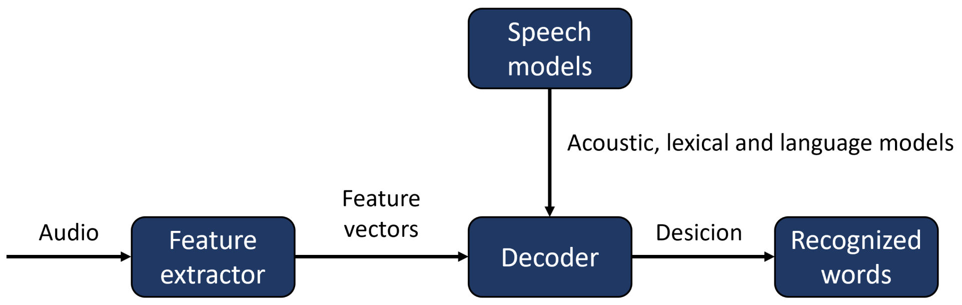

Figure 2.1: A simplified ASR system

A complete ASR system will take raw audio as input before extracting features from the audio in the *Feature extractor*. Afterwards, speech models (acoustic, lexical and language models) and the feature vectors are input to the decoder. The decoder will, by use of the applied speech models, try to figure out what words or phones the input audio consists of, before a decision is done and and recognized words will be the output of the system. In this thesis, feature extraction from the audio into corresponding feature vectors, stays the main focus and thus, the other parts of the system will not be discussed any further from this point on.

2.2 Feature extraction

Feature extraction in an ASR system is as mentioned in 2.1, the process in which the most important information in the input, i.e. the *features*, is extracted and isolated from the full sound waveform or audio input to the ASR system as well as disregarding the less important information in the speech signals. The goal in feature extraction is to reduce dimensionality but still enable the system to perform as good as possible. However, there are some trade-offs. The more information in the features, the more processing is necessary in the next steps in the ASR system [1].

2.3 Speech features

Good speech features have some differing demands for different applications. According to Wolf [2, p.2044-2045], should features in a *speaker* recognition system contain the following kinds of information:

- occur naturally and frequently in normal speech,
- be easily measurable,
- not change over time or be affected by the speaker's health,
- not be affected by reasonable background noise nor depend on specific transmission characteristics, and
- not be modifiable by conscious effort of the speaker, or, at least, be unlikely to be-affected by attempts to disguise the voice.

A last point he noted is that the features should "vary as much as possible among speakers, but be as consistent as possible for each speaker". The last point would not be valid for a pure *speech* recognition system as is the target for this thesis, but the other points he noted would also apply for speech recognition systems as well as a speaker recognition system. In a speech recognition system, the variance between speakers should not be very large as the focus is to classify utterances from different speakers together, where as in a speaker recognition system, the main goal should be to find the differences between the speakers from the features. Therefore, the features used in these two kinds of systems, should not be the same. The other points noted by Wolf, however, seems to fit a speech recognition system very well.

Today, there are several different techniques used for feature extraction and most of them, if not all, are using spectral representations of the audio signals. Some of the most commonly used are PLP (Perceptual Linear Prediction), LPC (Linear Prediction Coefficients) and MFCC (Mel Frequency Cepstral Coefficients), which is maybe the best known and most used of them all [3]. By using spectral representations, the computational burden of extracting these features, are quite low, compared to what is possible with the modern technologies of today. In the last few years, however, other variants of speech feature extraction using deep neural networks have emerged as good options and they have been shown very promising results.

2.3.1 Log mel spectrograms

Log mel spectrograms is a basic representation of speech, commonly used as a basis for several different speech feature representations such as for instance MFCC. The log mel spectrogram can be obtained by taking a short time Fourier Transform on the audio before mapping to the mel scale and finally applying the logarithm to the whole signal. The mel frequency scale is thus given by

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.1}$$

where m is the mel frequency and f is frequency. Its relationship with the regular frequency is shown in Figure 2.2. As can be seen from the figure, the relationship between the two is almost linear up to 1000 Hz before flattening more out in the higher frequency range.

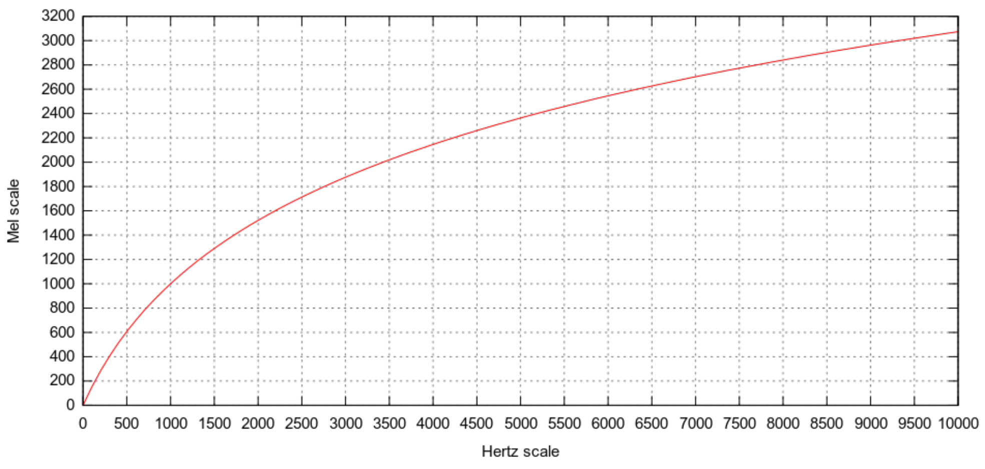

Figure 2.2: Hertz frequency vs Mel frequency. Figure from [4].

The mel scale is a perceptually motivated scale used in order to try to model the human hearing more accurately than the regular frequency axis. The scale is experimentally based and tries to model the human hearing in that the ears' sensitivity for frequencies is not purely linear, but rather more logarithmic, especially for frequencies above 1000 Hz. Since the human ear can more easily distinguish between lower frequencies than higher with the same relative difference, the mel scale is widely used in speech speech recognition because of these properties [5, p.33].

2.3.2 MFCC

Mel Frequency Cepstral Coefficients, or MFCC for short, is arguably the most commonly used speech feature used in automatic speehc recognition systems today. Davis and Mermelstein used the MFCC as speech features originally in 1980 [6] with great success, and since they have been used extensively for such applications. In order to obtain the MFCCs, the first thing that needs to be done is to window the signal and emphasizing higher frequencies. Then, the Fast Fourier Transform is applied before transforming the signal into the mel scale. Afterwards a log is taken before applying the discrete cosine transform (DCT). The DCT ensures that the coefficients are decorrelated [3]. Figure 2.3 shows in more detail how to extract the MFCCs.

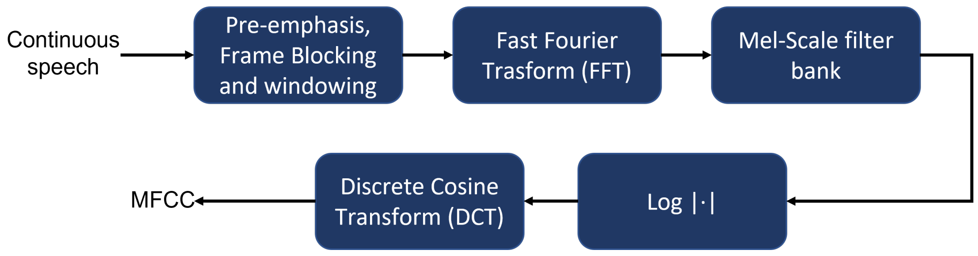
Continuous speech
Pre-emphasis, Frame Blocking and windowing
Fast Fourier Trasform (FFT)
Mel-Scale filter bank
Log |·|
Discrete Cosine Transform (DCT)
MFCC

Figure 2.3: Block diagram of MFCC extraction. Based on figure 1 in [3].

Equation 2.2 shows how to calculate the MFCCs:

$$\hat{C}_n = \sum_{n=1}^{k} \left(\log \hat{S}_k\right) \cos\left[n(k-\frac{1}{2})\frac{\pi}{k}\right] \tag{2.2}$$

where k is the total number of coefficients, $\hat{S}_k$ the output of the mel filter bank and $\hat{C}_n$ is the output coefficients.

In speech recognition, the coefficients used from the MFCC are mostly the 2nd and up to the 13th. These coefficients are usually the most important regarding the spectral information in the speech and therefore most useful for speech and speaker recognition.

2.3.3 Phonemes

A phoneme is the smallest unit of speech, distinguishing one word from another by by representing a single sound [7]. Phonemes, as opposed to phones, are language-specific, and changing a phoneme in a word in one language, changes the meaning of the word. A phone is rather the sound itself and in fact, two phones can represent the same phoneme. E.g. in Norwegian there are two different pronunciations of the 'r' consonants in different dialects even though they are considered the same phoneme. The number of phonemes in a language is usually between 30 and 50, but a common choice in phoneme recognition is 39. Correct recognition of phonemes, or rather reducing the phoneme-error-rate, PER, is the goal of phoneme recognizers.

Monophone is a term used when considering only one phone and not taking into account the previous or latter phone. Triphones are context-dependent and considers both the previous phone and the following. These terms are often used in phoneme recognizers. A recognizer using triphones will likely have a better PER due to more information being considered.

2.4 Predictive coding

The term Predictive coding, coined in 1955 by Peter Elias [8] is the process where, based on the current and previous inputs to a network, a system tries to predict future, missing or contextual information. These methods have been shown to provide good results in learning speech representations for speech recognition and speaker recognition tasks. Some noteworthy examples are Contrastive Predictive Coding [9], Autoregressive Predictive Coding [10] and Vector-Quantized Autoregressive Predictive Coding [11].

2.4.1 Non-autoregressive Predictive Coding

Non-autoregressive Predictive Coding is a self-supervised method for learning speech representations. This method was proposed in 2021 by Liu, Chung and Glass as an alternative to other methods for learning speech representations [12]. NPC was shown to have good qualities in representation learning as well as being of lower complexity than other comparable methods.

The goal of NPC is to derive high-level representations $(h_1, h_2, ...h_T)$ from surface feature audio $(x_1, x_2, ...x_T)$ where T is the length of the sequences. Compared to APC and CPC, NPC is not autoregressive, and can therefore be parallelized in time, since outputs are not dependent on previous outputs. This makes NPC more useful for tasks requiring less time usage [12].

A diagram of the NPC structure is shown in Figure 2.4.

In order for the NPC to work and to acquire its features, h_t, at time t, a receptive field is chosen with a size of $R = 2r + 1$, where r is the distance from the center.

The main goal of the NPC is to minimize the L1 difference between the input feature, x_t, and the output, y_t, based on the learned representation, h_t at all time instances:

$$\sum_{t=1}^{T} |y_t + x_t|. \tag{2.3}$$

Because of the model's willingness to learn the representation simply as a copy of the input, some of the inputs are masked and prohibited to use in the model.

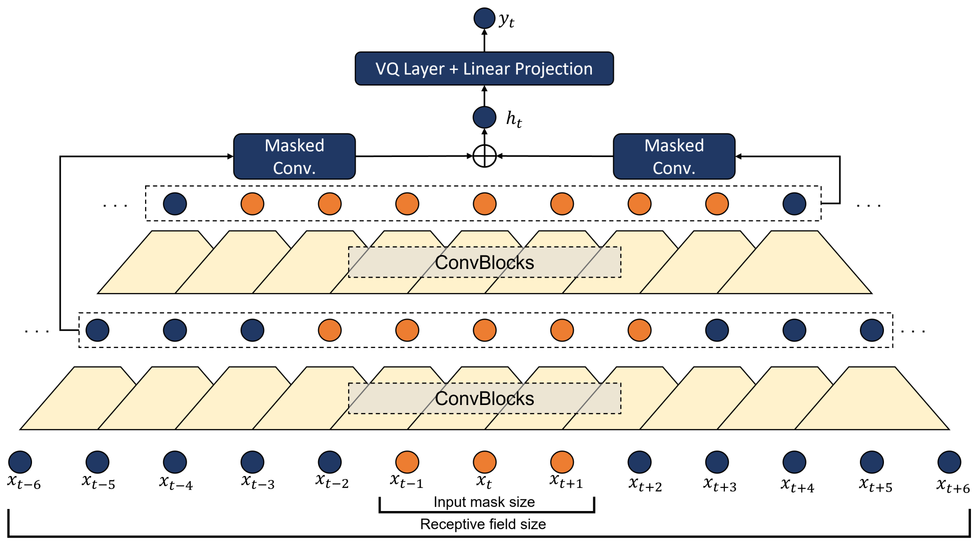

Figure 2.4: An example NPC structure. Based on Fig 1(a) in [12]. The orange frames are prohibited to use in NPC. Here, the receptive field size is 13, and the input mask size is $M_{in} = 3$.

Explicitly, the input mask size, M_{in} decides which inputs the model are not allowed to use. These are marked with orange in Figure 2.4.

The first layer of the model is a row of convolutional blocks, as shown in Figure 2.5.

In the ConvBlock, there is first a one-dimensional convolutional layer, before batch normalization (BN) which is basically re-scaling and re-centering of the features before a ReLU activation function. The next layer consists of a linear layer, batch normalization and dropout. After that, a residual connection is added before a final ReLU activation. The features will go through two layers of ConvBlocks before the masked convolutional blocks. The masked ConvBlocks are shown in Figure 2.6.

The masked ConvBlock hides the center inputs and only takes inputs from the edges into a one-dimensional convolutional layer, before a tanh activation function. The kernel-wise convolution in the masked ConvBlock can be described as:

$$(W \odot D) * Z, \tag{2.4}$$

where $Z \in \mathbb{R}^{T\times d}$ denotes the intermediate features from the model with sequence length T and dimension d. $W \in \mathbb{R}^{k\times d}$ is the learnable kernel weight with size k and $D \in \{0,1\}^{k\times d}$ is the mask with each element $d_{ij} = \mathbb{1}_{i\leq \frac{k}{2}-m} + \mathbb{1}_{i\geq \frac{k}{2}+m}$ [12].

Finally, a Vector Quantization (VQ) layer and a linear projection layer is added. The VQ is used as a bottleneck and an extra constraint for the model in order to only learn the most essential information from the speech. The linear projection

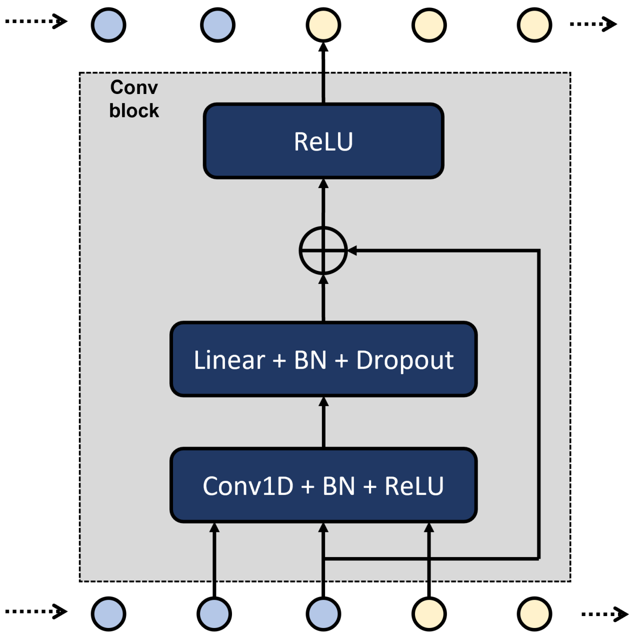

Figure 2.5: The ConvBlock is made up of a convolutional layer, Conv1D, batch normalization, BN and a rectified linear unit activation function before a linear layer, batch normalization, BN, and dropout layer. Finally the residual is added before another ReLU activation. Based on fig 1(b) in [12].

layer will act as a transformation back to a more "spectrogram-like" domain. When a latent representation is used later in the experiments, it is extracted between the VQ layer and the linear projection layer, meaning that the latent representation will probably not resemble a spectrogram, since it has not been through the linear projection layer.

2.5 Other methods for representation learning

One of the best end-to-end automatic speech recognition systems in use today is the Wav2Vec2.0 [13]. Its representation learning is based on contrastive learning instead of predictive coding as in NPC. However, there are several similarities between the two. Wav2Vec2.0 also uses self-supervised learning, convolutional layers, masking and vector quantization, although the full structure is a little bit different than NPC. This method have shown very good results in phoneme recognition, even better than NPC [12].

2.6 Adaptation

Transfer learning or more specifically domain adaptation is when previous knowledge is used as a basis for training a deep neural network into a similar, but not

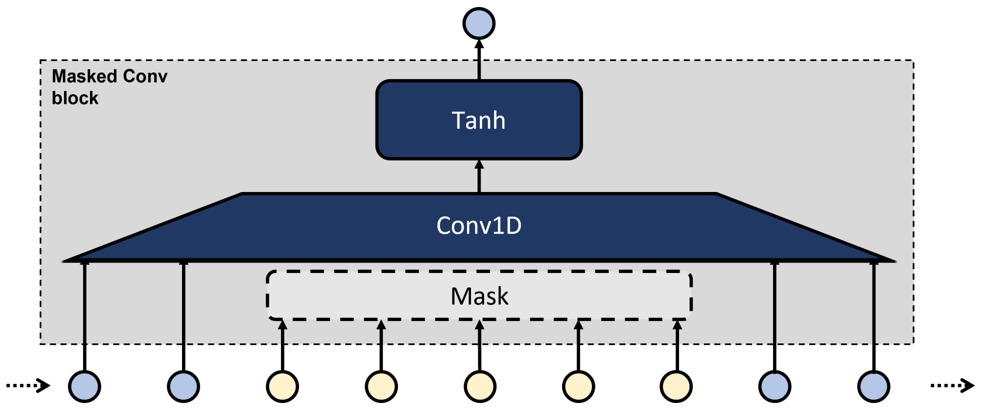

Figure 2.6: The Masked ConvBlock is made up of a single convolutional layer, Conv1D, only taking the edges of its receptive field as input and then a Tanh activation function.Based on fig 1(c) in [12].

equal domain [14]. In feature-extraction-for-speech-terms, this can for instance be training an English feature model with Norwegian training data in order to be used on Norwegian datasets. This process could ease the need for large amounts of training data for several languages if only a small adaptation training set is necessary.

Chapter 3

Methods

3.1 Datasets

In order to conduct the experiments for this thesis, some datasets are needed. Both English and Norwegian datasets are used, with one labeled and one unlabeled dataset for each language. The chosen datasets are shown in Table 3.1. The unlabeled datasets are used for self-supervised training of the speech representation models while the labeled are used for testing and and phoneme recognition tasks.

Table 3.1: The datasets used for the conducted experiments.

	Unlabeled	**Labeled**
English	Librispeech	TIMIT
Norwegian	NST	NbTale

3.1.1 Unlabeled datasets

The Librispeech [15][1] and has several different sets included. The chosen training set consists of 100 hours of read audio books denoted as 'clean'. This means by Librispeechs own definition that there is little noise and only American English accents used in the dataset. The NST dataset[2] from the SVoG research project is used for training the Norwegian models. The training set used from NST consists of about 365 hours of speech, mostly from read manuscripts. It contains texts in both Nynorsk ($\approx 12\%$) and Bokmål ($\approx 88\%$) and speakers with different dialects, but since it is read manuscripts in either Bokmål or Nynorsk, the dialect differences are not as audible as they would have been in natural speech. The corresponding development and test sets are also used for both Librispeech and NST. An overview of the training set contents and length and duration of the datasets used are given in Table 3.2.

[1]Librispeech datasets are available at `https://www.openslr.org/12`

[2]NST datasets are available at `https://www.nb.no/sprakbanken/en/resource-catalogue/oai-nb-no-sbr-54/`

Table 3.2: The unlabeled training sets used in the experiments.

Dataset	Speakers			Duration		Context
	Total	Male	Female	Per speaker	**Total**	
train-clean-100	251	126	125	25 min	100 h	Audio books
NST trainset	900	415	485	312 sentences	365 h	Manuscripts

3.1.2 Labeled datasets

For testing and phoneme recognition, labeled datasets are used. For the English model, the TIMIT dataset was used[3]. The dataset consists of 10 30 second sentences from each of 630 speakers from 8 different major dialect regions in the United States. The audio recordings are from read manuscripts, and have a total duration of around 52.5 hours. The phonetic transcriptions in the TIMIT dataset are based on a set of 61 phones, but with some alterations training are done based on a set of 48 phones while scoring and testing usually is done with 39 phones. For the Norwegian models, the NBTale dataset is used[4]. More specifically, part 1 of the dataset. This consists of 240 speakers with 20 sentences each. These speakers are from different regions of Norway and thus speak different dialects. But still, as for the NST dataset, only manuscripts are used, so the differences are not very large. Part 3 of the NBTale dataset contains free speech, and thus more difference in dialects, but this is not used in these experiments. NBTale's transcriptions uses a set of 50 phones, so it is not completely comparable to the 48 used by TIMIT (Usually transformed into 39 phonemes). Table 3.3 shows a summary of the labeled datasets.

Table 3.3: The labeled datasets used in the experiments.

Dataset	Number of speakers	Duration		Context
		Per speaker	**Total**	
TIMIT	630	5 min	52.5 h	Manuscripts
NBTale (Part 1)	240	20 sentences	4800 sentences	Manuscripts

3.2 Code

A simplified overview of the flow of the data between different programs used in this master thesis is shown in Figure 3.1.

The dataset inputs are the datasets mentioned in section 3.1. The NPC training block represents the representation learning by the neural networks using the NPC method. NPC model represents the trained models, be it the English, the Norwegian or the adaptation models. Inference / Feature extraction represents

[3]TIMIT datasets are available at `https://catalog.ldc.upenn.edu/LDC93S1`

[4]NBTale datasets are available at `https://www.nb.no/sprakbanken/en/resource-catalogue/oai-nb-no-sbr-31/`

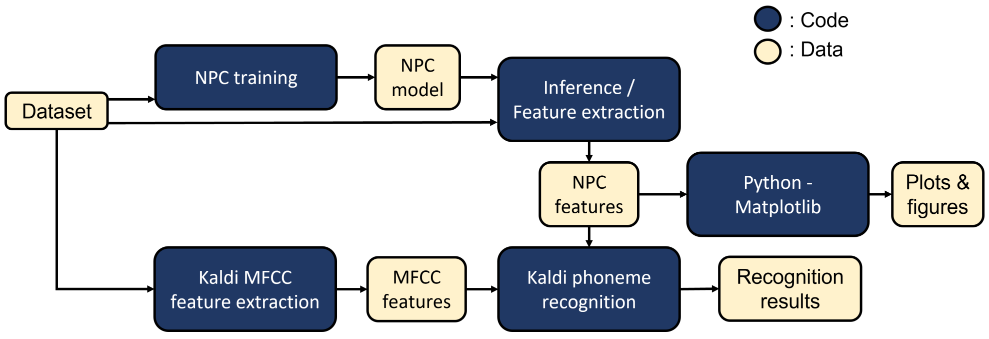

Figure 3.1: The flow of data between different programs used.

the forward methods from the NPC and creation and storage of features generated by using the NPC model. The NPC features are the learned features and representations which will be used for plotting and phone recognition later on. The Python - Matplotlib block represents the code for visualization and plotting of the learned representations and comparisons with other types of features. The Kaldi MFCC feature extraction block represents simply extracting MFCC features from datasets by using the Kaldi codebase and scripts. Kaldi phoneme recognition represents the code used for a phone recognizer both by using the NPC features and MFCC features. Finally, recognition results are stored and used in tables for comparisons.

3.2.1 Representation learning

The basis for the experiments in this thesis is the code from Liu, Chung and Glass' github repository for the NPC [12][5]. This repository is mainly used for NPC training. For inference and feature extraction both code from the NPC repository and some of Janine Rugayans work[6] for her masters thesis in 2021 about deep learning in spoken language acquisition [16] is used. In addition to this, self-written code is used for tying it all together and creating a more streamlined program for running the experiments.

3.2.2 Examination of the representations

Examination of the learned representations and the speech features are done with Python and mostly its Pandas [17] and Numpy [18] library for data processing and Matplotlib [19] library for visualization. In addition, Praat sowftware[7] is used for making figures with waveforms, MFCCs and phonetic labels combined from audio files.

[5]Code available at `https://github.com/Alexander-H-Liu/NPC`.
[6]Code available at `https://github.com/janinerugayan/masterthesis/tree/master/VQ-APC`
[7]`https://www.fon.hum.uva.nl/praat/`

3.2.3 Testing trained speech representation features in ASR system

The testing of the trained speech representation models are done with Kaldi [20]. Kaldi is a speech recognition toolkit used for speech recognition and similar applications. For this thesis, the TIMIT example included in the Kaldi codebase is used as a basis for testing the English model as well as the Norwegian. For the Norwegian models, more modifications to the original TIMIT examples have been made in order to read the speech corpora correctly.

When testing the trained NPC model features in Kaldi, some additional modifications to the code were needed. Most of the modifications were done following the guidance in [21], in addition to some own modifications needed to tie all the different programs better together and to make the speech recognizer working correctly.

3.3 Training

Training of the NPC models are done with the same parameters as in [12]. These are shown in Table 3.4.

Table 3.4: NPC parameters

	Parameter	**Value**
Dataset	batch_size	32
	audio_max_frames	1500
Audio	feat_type	fbank
	feat_dim	80
	frame_length	25
	frame_shift	10
	cmvn	True
Model parameters	kernel_size	15
	mask_size	5
	n_blocks	4
	hidden_size	512
	dropout	0.1
	residual	True
	batch_norm	True
	activate	relu
	disable_cross_layer	False
VQ parameters	codebook_size	[64,64,64,64]
	code_dim	[128,128,128,128]
	gumbel_temperature	1.0
Hyper parameters	optimizer	Adam
	lr	0.001
	epoch	100

The batch size used in training is 32 and the maximum length of the audio frames is 1500 samples. Audio features used as inputs for training will be 80 dimensional fbank features which are simply log mel filterbank features. They will use a frame length of 25 ms and have a 10 ms frame shift. Cepstral mean and variance normalization (CMVN) is also applied to the mel spectrograms.

For the model parameters, the kernel size is 15, mask size is 5 and the number of stacked ConvBlocks, n_blocks, is 4. This results in a receptive field size, R, of:

$$R = kernel\_size + 2 \cdot n\_blocks = 15 + 2 \cdot 4 = 23. \qquad (3.1)$$

The hidden dimension size of all layers is 512. In the ConvBlocks, the dropout is 0.1, a residual connection is added, and batch normalization is done. The activation function of the ConvBlocks is the ReLU. Not disabling cross layer means that the Masked ConvBlocks are at only the last layer. For the VQ parameters, the codebooks are four codebooks of size 64 with a dimension of each at 128, and finally, the temperature of the Gumbel Softmax is 1.0.

Adam is used as the optimizer, the learning rate is set to 0.001 and the number of epochs is set to 100.

3.3.1 Models

Three different NPC models are trained for the experiments. The English model is trained according to the parameters in Table 3.4 with the Librispeech training and development datasets. The Norwegian model is trained with the NST dataset with the same parameters as the English model. The adaptation model is trained with the same parameters as the English and Norwegian models, with the only difference that is only trained for 20 epochs with the Norwegian dataset on top of the previously trained English model. Training is done with the NST dataset. This makes the total number of epochs trained for the adaptation model 120. In addition to the regular output from the NPC, latent representations extracted after the VQ-layer and before the linear projection layer, are used as well.

3.4 Inference

Running inference of the trained NPC models is done separately with the three different models. For these experiments, inference is done in two occasions as well. One for the creation of features for visualization and plotting and one in combination with the phoneme recognizer. During inference for the speech recognizer, labeled datasets are used. For the visualization of features, labeled datasets are also most practical. It is easier to examine the features in combination with labels.

3.5 Phoneme recognition

Phoneme recognition for the experiments in this thesis will be done with the Kaldi speech recognition toolkit. There are several different options for testing phoneme recognition in Kaldi. In this thesis I have looked at the tests called 'Mono', 'Tri1', 'Tri2' and 'Tri3'.

'Mono' is the simplest and is a monophone training and decoding which does not take into account surrounding phones, e.g. a context-independent test, but also uses the delta features. The 'Tri1' uses a triphone model and uses delta and delta-delta as well. 'Tri2' is also triphones, but it uses LDA and MLLT (Linear Discriminant Analysis and Maximum Likelihood Linear Transform). 'Tri3' the same as 'Tri2' with the addition of SAT (Speaker Adaptive Training).

The Kaldi phoneme recognition also has the option to use more advanced methods like Deep Neural Networks (DNN) and Subspace Gaussian Mixture Models (SGMM) for decoding as well, but these are not used in these experiments for simplicity.

Chapter 4

Results

Results in this chapter will include both visual and phonetic inspection of the learned representations as well as results from using the learned representations in a working phone recognition system. Latent representations will not be examined visually as the information in those are very hard to interpret visually, but they are used in the phoneme recognition. In addition, comparisons with MFCCs are shown and discussed.

4.1 Learned representations

By training NPC models, and running inference we can save the representations learned. By plotting the output from inference, the NPC representations can be visualized and examined. Figure 4.1 shows the how the NPC features can look like compared to the waveform, spectrogram and MFCCs.

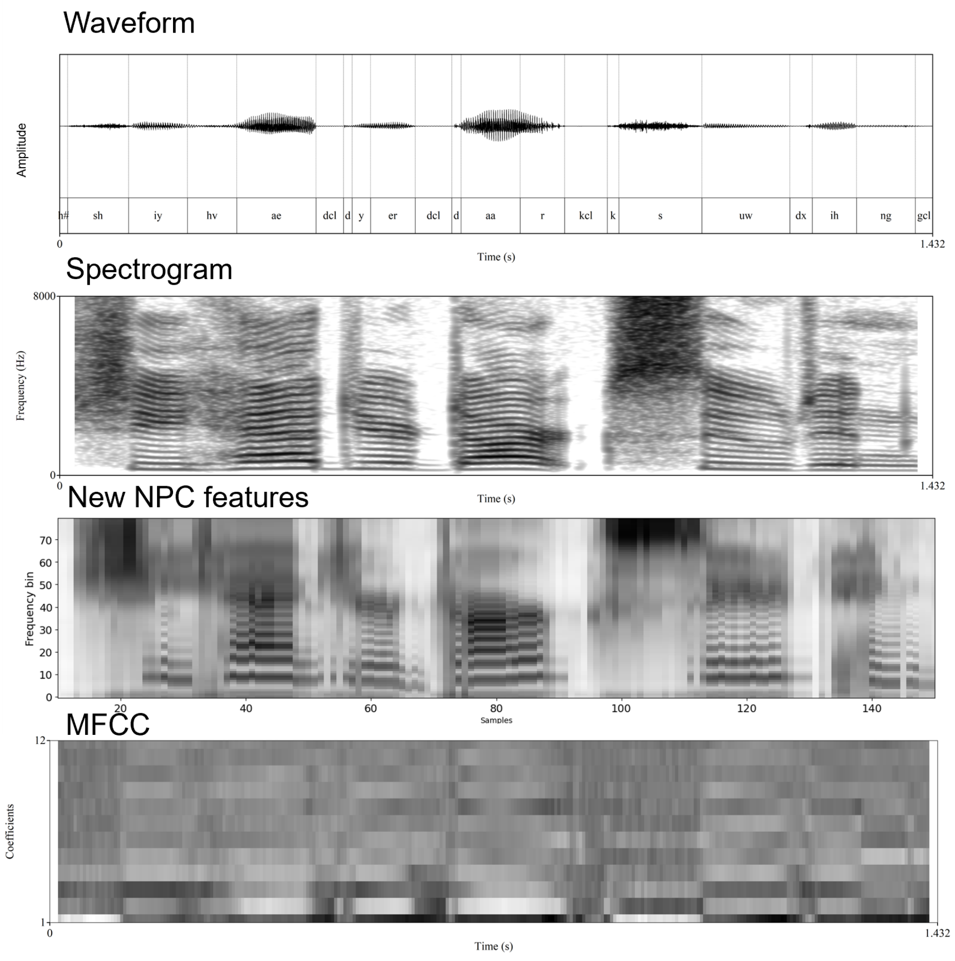

Figure 4.1: Waveform, phonetic transcription, NPC representation and MFCCs of one single sound clip.

Purely visually we can see that the NPC features resembles the spectrogram more than the MFCCs. It is also clear that it contains more information than the MFCC by having 80-dimensional features while the MFCCs (at least in this figure) utilize only 12 coefficients. That again is 6.7 times more information if the window length and stride is the same, which it is in this case.

Form Figure 4.1 we can also see that the learned representations are on a sub-phonemic level because of their short time span. Each vector represents 10 ms of speech so it is natural that most phonemes are represented with more than one vector. For example the 's'-phoneme at around sample 90-110 in the NPC features is represented by approximately 20 vectors. An other thing to note is that several different vectors are used for each phoneme, meaning they can represent different stages of a single phoneme, not only one vector for each phoneme. Since there are 256 different possible VQ vectors used in the output from the NPC, we can with great confidence assume that more than one vector is associated with each phoneme since the number of phonemes used ranges between 39-50. This is very clear from the figure, as the within the time frame from one phoneme, several different vectors are used to represent this phoneme. Figure 4.2 shows how the NPC representation of an ’s’-phoneme will look like and supports that observation further.

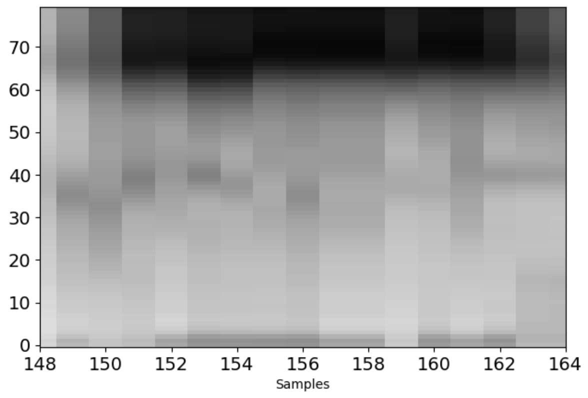

Figure 4.2: NPC representation of an ’s’-phoneme.

The ’s’-phoneme in this case with this speaker consists of several different vectors with some of them being multiples. Figure 4.3 shows how the NPC representation of two different speakers saying the same sentence will be.

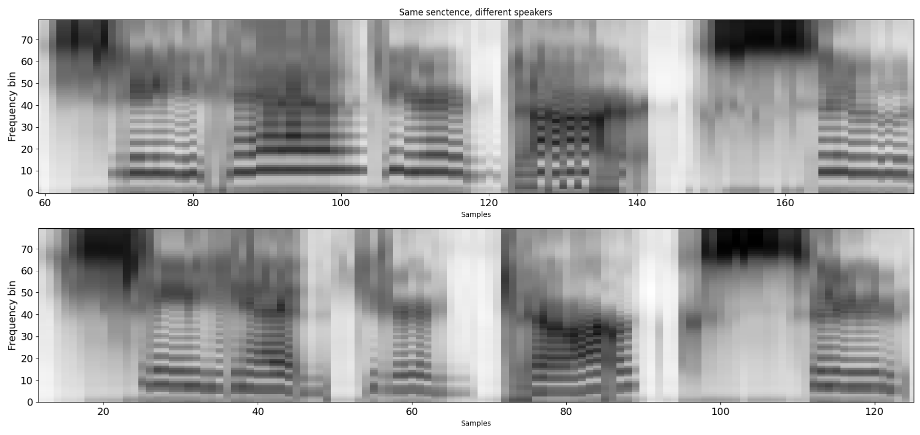

Figure 4.3: Two speakers saying the same sentence. NB: The alignment is a bit off due to the time used pronouncing the words is a little different for the two speakers.

Here we can see that there are some minor differences between the representations used, even though the main shapes and looks are quite similar. For easier distinguishing between the two, one can look at the VQ-codebook index associated with each representation. Figure 4.4 shows which codebook index is used for every output vector of the NPC for two different speakers of the same text. Note that two equal codebook indices does not necessarily correspond to two equal output vectors, because of the context dependency and the linear projection in NPC.

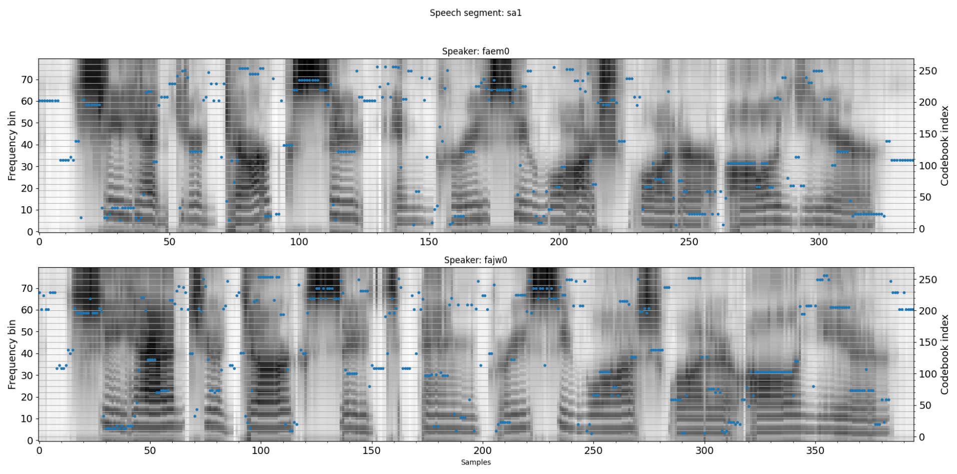

Figure 4.4: NPC output and corresponding VQ-vectors for two different speakers with the same sentence. The blue dots correspond to VQ vectors and their y-position to an arbitrary indexation of those vectors.

Here we see that for some phonemes we have long stretches using only one VQ vector or alternating between two. This will mean that in many cases one vector can be used to represent a phoneme. On the edges between phonemes we see more jumping between several VQ vectors meaning that there are several vectors representing transitions between phonemes on a more sub-phonemic level. We can also see that for some phones, the same codebook indices are used for both speakers, meaning that the latent representation will be the same for both speakers in those cases. This means that there are some generalization in the codebook vectors, giving us a set of codebook vectors used for each phoneme or sub-phoneme. Determining exactly which vectors are used for which phoneme (or sub-phoneme) for different speakers, would be a very extensive task and is therefore left for future work. From this figure alone, we can see that there is overlapping between different users, and we can safely assume there are sets of vectors used for each sub-phoneme in general.

Figure 4.5 shows how the Norwegian and English model differs when applied to the same (English) sound clip.

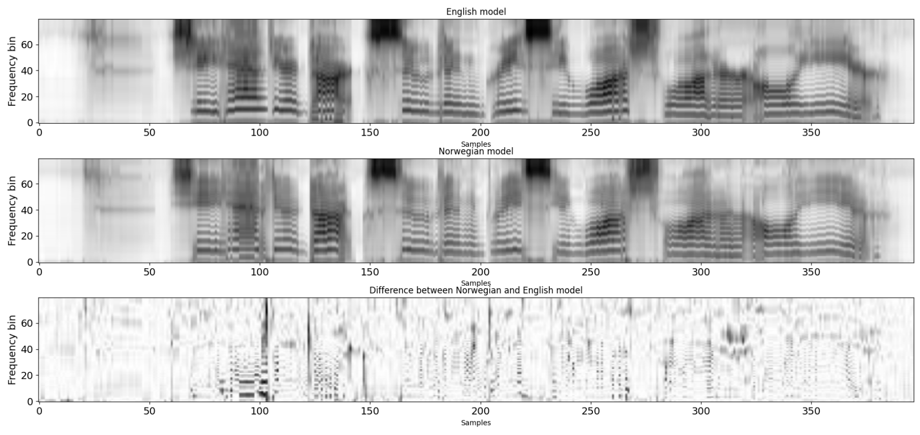

Figure 4.5: English and Norwegian NPC model applied to English audio. The third row is shows the absolute difference between the two.

As can be seen from the figure, the absolute difference between the two is not very large, with mainly just small differences at the edges of some phonemes. The largest differences are found around sample 100. Here we find 'ae' (IPA: /æ/) and 'dcl' phonemes, which is the closing phase before a 'd' in the word 'had'. All of these phonemes are prevalent in Norwegian so the difference here was not very expected. It would be reasonable to expect more differences in sounds that are not in both languages, not common sounds. The main thing to note here is that most of the differences are in transitional areas between phonemes.

4.2 Phoneme recognition tests

4.2.1 English phone recognition test

The results from testing the Phoneme-error-rate (PER) on the English NPC features (both regular output and latent) versus the standard MFCC features is shown in Table 4.1.

Table 4.1: PER for the english NPC model features compared to MFCC. E-NPC-L denotes latent English NPC features.

	Mono	**Tri1**	**Tri2**	**Tri3**
E-MFCC	31.7	26.3	23.7	22.3
E-NPC	51.1	49.4	36.0	35.2
E-NPC-L	50.2	49.3	37.2	36.1

These results show that the MFCC outperforms the NPC features in all four tests, and quite substantially as well. MFCCs oupterforms the NPC with 12.3 ('Tri2') to 23.0 ('Tri1') percentage points difference with the 'Tri2' test being the closest and the ’Tri1’ having the largest difference. The latent representation is slightly better than the regular in the ’Mono’ and ’Tri1’ test, while the regular is better in the last two tests.

4.2.2 Norwegian phone recognition test

The results from testing the PER of the Norwegian NPC features (both regular output and latent) versus the standard MFCC features is shown in Table 4.2.

Table 4.2: PER for the norwegian NPC model features compared to MFCC. N-NPC-L denotes latent NPC features.

	Mono	**Tri1**	**Tri2**	**Tri3**
N-MFCC	43.4	34.1	31.5	29.7
N-NPC	67.4	67.1	44.2	42.5
N-NPC-L	66.3	63.2	47.3	46.1

Here we see the same picture as for the English model with the MFCC features outclassing the NPC features with from 12.7 to 29.1 percentage points for the different tests. Here the difference between the two NPC models are a bit larger than for the English one, but still the latent model is best for ’Mono’ and ’Tri1’ while the regular is best in ’Tri2’ and ’Tri3’.

4.2.3 Adaptation model phone recognition test

The test results of the PER of the adaptation model is shown in Table 4.3. This test is done with only the different Norwegian models.

Table 4.3: PER for the adaptation model features compared to the Norwegian NPC models. The best results are highlighted with a bold font. N-NPC the regular Norwegian NPC model, N-NPC-L is the latent Norwegian NPC model, A-NPC is the adapted-to-Norwegian NPC model and A-NPC-L is the latent adapted-to-Norwegian NPC model.

Model	**Mono**	**Tri1**	**Tri2**	**Tri3**
N-NPC	67.4	67.1	44.2	42.5
N-NPC-L	66.3	63.2	47.3	46.1
A-NPC	67.2	65.8	**43.4**	**41.9**
A-NPC-L	**61.2**	**61.4**	45.6	44.3

Here we can see that the adaptation models actually are performing better than its counterpart on their respective best tests. For ’Mono’ and ’Tri1’ the A-NPC-L model is clearly the best and outperforms N-NPC-L with 5.1 and 1.8 percentage points, while A-NPC is slightly better than the N-NPC model. For ’Tri2’ and ’Tri3’ A-NPC is the best and beats N-NPC with 0.8 and 0.6 percentage points respectively. A-NPC-L is better than N-NPC-L here, but is slightly worse than N-NPC.

It is clear from all these experiments that the latent representations are better than the regular for 'Mono' and 'Tri1', but the standard NPC model is better for 'Tri2' and 'Tri3'. Another thing to note from these experiments is that the MFCCs are outclassing all of the trained NPC models in all tests, which was not expected.

4.2.4 Types of errors

It can be interesting to look at what kinds of errors occur the most often while using the NPC-features for phoneme recognition. Table 4.4 shows a simple overview of error types and percentage of correct phonemes in the 'Tri3' decoding test.

Table 4.4: Correct phonemes (%) and error type percentages in the 'Tri3' decoding test. The E-prefix denotes tests with English datasets, N-prefix is Norwegian, and A-prefix is the adaptation model. The L-postfix denotes latent representation features.

Category	**E-MFCC**	**E-NPC**	**E-NPC-L**	**N-MFCC**	**N-NPC-L**	**A-NPC-L**
Correct	80.9	70.3	70.3	76.0	62.7	63.9
Substitutions	14.0	22.0	22.0	15.3	25.6	25.0
Deletions	5.1	7.7	7.6	8.7	11.7	11.0
Insertions	3.2	5.5	6.4	3.9	8.8	8.3
Total errors	22.3	35.2	36.1	27.9	46.1	44.3

It is clear that substitutions is the main source for errors in this phoneme recognition test, both for the MFCC-features and the NPC-features. Almost half of all errors for all features, both MFCC and NPC, are substitutions, so here the main difference between the two is simply the error rate, and not the type of errors.

4.2.5 Substitutions

Substitutions is the most frequent errors in the phoneme recognition and is therefore the most interesting to investigate. A substitution is here defined as one phoneme changed with another, and substitutions are listed both ways. Table 4.5 shows the top ten confusion pairs and their relative error rate for A-NPC-L in the 'Tri3' decoding test.

Table 4.5: Top ten confusion pairs (substitutions) for latent NPC adaptation model, A-NPC-L, in the 'Tri3' decoding test.

	% of total subs.	Correct phone	Recognized phone
1	2.02	a:	a
2	1.97	m	n
3	1.84	i:	i
4	1.66	a	a:
5	1.61	n	m
6	1.36	l	r
7	1.31	o	a
8	1.31	o	o:
9	1.31	e:	e
10	1.29	i	i:

This shows that the errors mostly are very similar sounds, with the mos common error being between 'a:' and 'a' which is just a long and short a-sound. In fourth place we have the same phonemes only the other way around. In fact, all of the top five errors have its counterpart also in the top ten substitution list.

Table 4.6 shows the top ten substitutions for the N-NPC-L model.

Table 4.6: Top ten Norwegian NPC latent, N-NPC-L, confusion pairs in the 'Tri3' decoding test.

	Correct	Recognized
1	a:	a
2	m	n
3	a	a:
4	n	m
5	i:	i
6	l	r
7	i	i:
8	o	a
9	e:	e
10	r	l

This table show almost exactly the same errors as Table 4.5, which means that the same problems are evident both for the adaptation model as the standard Norwegian NPC model. Table 4.7 shows the errors from the MFCC features.

Table 4.7: Top ten Norwegian MFCC, N-MFCC, confusion pairs in the 'Tri3' decoding test.

	Correct	**Recognized**
1	a:	a
2	i:	i
3	a	a:
4	m	n
5	i	i:
6	o	o:
7	o	a
8	e:	e
9	y	i
10	d	t

These errors are not exactly the same as for the NPC models, but still, the top 8 of these are in both of the top ten for the NPC features. One thing to notice from these results is that the most common substitutions are vowels, especially substitutions between long and short versions of the same vowel, with the exception of the 'l-r' and 'm-n' confusion pairs. The results from these tests show that even though the error rates by using the different features are different, we get roughly the same types of errors.

Table 4.8 shows the substitution errors for the English features in the 'Tri3' decoding test.

Table 4.8: Substitutions for the English features as percentage of total substitutions in the 'Tri3' decoding test.

	E-MFCC			**E-NPC**			**E-NPC-L**		
	Corr.	Rec.	%	Corr.	Rec.	%	Corr.	Rec.	%
1	z	s	4.6	ah	ih	3.7	ah	ih	3.8
2	ah	ih	4.3	z	s	2.5	ih	ah	2.8
3	ih	ah	3.6	ih	ah	2.4	z	s	2.3
4	m	n	2.8	m	n	2.3	eh	ih	2.0
5	eh	ih	2.5	ih	iy	1.8	m	n	1.8

The results by using English features is quite similar to what was shown for the Norwegian features. Here again we see that mostly the same errors are reoccurring, and especially the 'ah'-'ih' confusion pair is prevalent being listed twice for all features. Once again we see that errors are quite similar using different features.

4.2.6 Deletions

The error type defined as deletions is when the recognizer simply removes a phoneme, without replacing it with another. Table 4.9 shows the five most frequent deletion errors as a percentage of the total deletion errors for the English decoding test in a 'Tri3' system.

Table 4.9: Deletion errors for the English features as percentage of total deletion errors.

	E-MFCC		**E-NPC**		**E-NPC-L**	
	Phone	%	Phone	%	Phone	%
1	sil	16.0	sil	11.1	sil	11.1
2	r	8.4	ih	8.9	ih	9.5
3	ih	7.9	ah	6.2	ah	6.4
4	ah	6.5	r	6.0	r	6.2
5	t	4.6	n	5.1	n	5.1

Silence is clearly the most common deletion with 'r', 'ih' and 'ah' being the second to fourth most common deletion for all, including MFCC. This shows that the same deletion errors occur both for MFCCs and NPC features even though the error rates are somewhat different.

Table 4.10 shows the same tests but for Norwegian features.

Table 4.10: Deletion errors for the Norwegian features as percentage of total deletion errors.

	N-MFCC		**N-NPC-L**		**A-NPC-L**	
	Phone	%	Phone	%	Phone	%
1	r	12.0	r	11.7	r	12.4
2	e:	6.8	l	6.2	l	6.0
3	e	6.7	i	5.5	e:	5.6
4	i	6.2	e:	5.4	i	5.3
5	d	4.8	v	5.2	e	5.2

Here the 'r' phoneme is the most common deletion with 'e:' and 'i' occurring in all three tests. One thing to notice here is that 'l' is also a struggle for the NPC models, while not being in the top five deletions for the MFCC. But in general, the deletion errors are quite similar for all the models tested in the same language.

4.2.7 Insertions

Insertion errors is when the phone recognizer inserts an extra phone into the recognition results compared to the transcription. The most common insertion error is by far insertion the silence phone into the recognition results for all models and features. An overview of the top three insertion errors is shown in Table 4.11.

Table 4.11: Insertion errors by percentage of total insertion errors for the different English features in 'Tri3' decoding test.

	E-MFCC		**E-NPC**		**E-NPC-L**	
	Phone	%	Phone	%	Phone	%
1	sil	23.4	sil	19.9	sil	16.8
2	ih	10.8	ih	8.8	ih	8.8
3	r	8.6	l	5.0	k	6.0

In addition to the silence phone, the 'ih' phoneme for the English features seem to be a problem for all, being a clear number two of the insertion errors. One thing to notice here is that the NPC models seem to make the silence-phone-error less than the MFCC relative to the total number of errors, but since the total number of insertion errors is larger, this may or may not be of relevance.

Table 4.12 shows the insertion errors for the Norwegian features.

Table 4.12: Insertion errors by percentage of total insertion errors for the different Norwegian features in 'Tri3' decoding test.

	N-MFCC		**N-NPC-L**		**A-NPC-L**	
	Phone	%	Phone	%	Phone	%
1	sil	17.0	sil	23.6	sil	23.3
2	i	8.6	r	10.0	r	8.6
3	r	8.1	i	6.2	i	6.4

Here we see that the NPC features struggles more with insertion of silence phones than the MFCC, both the pure Norwegian model and the adaptation model. The 'i' and 'r' phones are number two and three for all of the feature tests, which shows that all of the features have a problem with inserting these phones.

In general all of the models struggles with very similar errors and error types, so the main difference between the MFCC and NPC is not what kinds of errors they make, but rather the amount of errors.

Chapter 5

Discussion

5.1 NPC vs VQ-APC

Even though it has been proven that VQ-APC can perform better that NPC PER-wise[12], the time needed and complexity of training and inference of the NPC is so much lower that for many cases, NPC will be the best option of the two. The theoretical complexity of the VQ-APC is $O(T \cdot d^2)$, with T usually 512. NPC's theoretical complexity is $O(k \cdot d^2)$ with k in this case 15. Empirical inference time show that NPC is 29 times faster than VQ-APC [12].

5.2 Phoneme recognition

The results in the phoneme recognition for the NPC representations compared to MFCCs was not as good as expected. While the best results achieved here is a PER of 35.2 for English and 41.9 for Norwegian (Adaptation model), the MFCC achieved 22.3 and 29.7 PER in the same tests. In [12], the PER achieved was 27.9 and they noted a log-Mel spectrogram performance of 50.3, which is a significant difference. Although log-Mel spectrograms are not the same as MFCC, the difference is very large here. Another difference from these experiments and [12], is that they use a 360 hour training set where as only 100 hours have been used here. Still, the difference in performance is a bit strange.

Some other possible explanation to the relatively poor performance of these NPC features is that there might have been some unnoticed errors somewhere in the implementation or the extraction of the features. Also, the test sets used in this thesis is not the same as in [12]. The variability in the test sets might be larger than in the training sets and causing some of the poor results. However, all datasets used, both in training and testing, are recordings from read manuscripts or audiobooks, so the dataset differences should not be very large, and not skew the results too much. More robustness against dataset variations could have been achieved by using part three of the NST dataset for training. Since it contains free speech, it might have had an positive effect on dataset variations.

The latent representations outperform the regular NPC representations in the ’Mono’ and ’Tri1’ tests but not in the other ones. The reason of this is hard to say, but it is still an interesting find. The overall best NPC results are then from the regular NPC representations in the ’Tri3’ tests.

5.3 Adaptation

The results from the adaptation model features are interesting. At least one of the adaptation models outperforms the pure Norwegian model in all the tests conducted, with the latent being best for ’Mono’ and ’Tri1’ and the regular adaptation model for the other two tests. There might be several explanations to this, and simply more available training data can be one. Since the Norwegian and English are quite phonetically similar, the adaptation training might in practice only be more available training data, and thus resulting in better performance. Testing this with other less similar languages would be very interesting.

It could also be the case that the representations learned is on a sub-phonemic level that is more generalized across languages and that these kinds of adaptations will work in many different languages. If that is the case, the need for large amounts of training data for many languages can be greatly reduced, and adaptation training can be done in a larger scale for speech recognition systems. The amount of training time could also be reduced as the need of training models from scratch would be gone. Even though the representations are quite well generalized between languages, it is still reasonable to prefer using as similar languages as possible for the best performance.

5.4 Issues and time wasting

Even though a lot of code used for this thesis was already written by others, there have been a lot of issues regarding running the experiments. Especially, the phoneme recognition have caused problems. At first, the preferred solution was to find a working phoneme recognizer on Github, and change the feature extraction from MFCC features to the NPC features. This proved to be more difficult than expected. Differences in library versions, Python version and other problems regarding GPU usage made this option unfeasible even when trying five different phone recognition repositories. Then, a choice was made to change to Kaldi for phone recognition, which is mostly written in other languages than Python that were quite unfamiliar, which was also a new challenge. But after a lot of work, trial and error, generation of some phone recognition results were possible.

In addition to code-specific problems, there have been issues with the GPUs in the server being used by multiple users at the same time and consuming all the available RAM, disrupting the execution of some programs further delaying the experiments.

5.5 Future work

For future work, more studying into adaptation training between languages would be interesting. The possibility of maybe proving that the performance can be good even with adaptation training on a more general basis would be very exciting. Also figuring out exactly why the MFCCs outperforms the NPC features in these experiments will be left for future work. Determining which VQ-vectors corresponding to which phonemes, sub-phonemes and transitions will also be something that can be examined more thoroughly in the future.

Chapter 6

Conclusion

In this thesis, speech representations learned with NPC have been investigated and tested in a phoneme recognition system. The learned representations can, on a sub-phonemic level, be used for representing speech and representing transitions between phonemes as well as phonemes itself. They can be used with some degree of success in phoneme recognition systems. The experiments conducted showed that the NPC representations were not able to outperform MFCC, managing a PER of 35.2 and 42.5 for the English and Norwegian representations respectively compared to the MFCCs which achieved 22.3 and 29.7 on the same tests. The latent representations performed better than the regular NPC features in some tests, but the overall best PER results with NPC representations were the regular NPC in the 'Tri3' tests. The types of errors for both MFCCs and the NPC feature are very similar, with the main difference simply being the error rate.

Adaptation training have also been investigated for the speech representations. It has been shown that there are possibilities of training NPC models into other languages. Adaptation can both reduce time and reduce the need of large amounts of training data. This have shown promising results with the adaptation model representation outperforming the Norwegian with a PER of 41.9 compared to 42.5 for the Norwegian.

Bibliography

[1] C. Hao, M. Xin and Y. Xu, 'A study of speech feature extraction based on manifold learning,' *Journal of Physics: Conference Series*, vol. 1187, no. 5, p. 052 021, Apr. 2019. DOI: `10.1088/1742-6596/1187/5/052021`. [Online]. Available: `https://doi.org/10.1088/1742-6596/1187/5/052021`.

[2] J. J. Wolf, 'Efficient acoustic parameters for speaker recognition,' *The Journal of the Acoustical Society of America*, vol. 51, no. 6B, pp. 2044–2056, 1972. DOI: `10.1121/1.1913065`. eprint: `https://doi.org/10.1121/1.1913065`. [Online]. Available: `https://doi.org/10.1121/1.1913065`.

[3] S. A. Alim and N. K. A. Rashid, 'Some commonly used speech feature extraction algorithms,' in R. Lopez-Ruiz, Ed., IntechOpen, 2018. DOI: `10.5772/intechopen.80419`. [Online]. Available: `https://www.intechopen.com/chapters/63970`.

[4] K. Vedala, *Hertz frequency vs mel frequency*. [Online]. Available: `https://commons.wikimedia.org/w/index.php?curid=3775197` (visited on 12/12/2021).

[5] X. Huang, A. Acero and H.-W. Hon, *Spoken Language Processing: A guide to theory, algorithm, and system development.* Prentice Hall, 2001.

[6] S. Davis and P. Mermelstein, 'Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,' *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980. DOI: `10.1109/TASSP.1980.1163420`.

[7] T. E. of Encyclopedia Britannica, 'Phoneme,' in *Encyclopedia Britannica*, 2009. [Online]. Available: `https://www.britannica.com/topic/phoneme` (visited on 19/12/2021).

[8] P. Elias, 'Predictive coding–i,' *IRE Transactions on Information Theory*, vol. 1, no. 1, pp. 16–24, 1955. DOI: `10.1109/TIT.1955.1055126`.

[9] A. van den Oord, Y. Li and O. Vinyals, 'Representation learning with contrastive predictive coding,' *CoRR*, vol. abs/1807.03748, 2018. arXiv: `1807.03748`. [Online]. Available: `http://arxiv.org/abs/1807.03748`.

[10] Y.-A. Chung, W.-N. Hsu, H. Tang and J. Glass, 'An unsupervised autoregressive model for speech representation learning,' in *Interspeech*, 2019.

[11] Y.-A. Chung, H. Tang and J. Glass, 'Vector-quantized autoregressive predictive coding,' in *Interspeech*, 2020.

[12] A. H. Liu, Y.-A. Chung and J. Glass, 'Non-Autoregressive Predictive Coding for Learning Speech Representations from Local Dependencies,' in *Proc. Interspeech 2021*, 2021, pp. 3730–3734. DOI: 10.21437/Interspeech.2021-349.

[13] A. Baevski, H. Zhou, A. Mohamed and M. Auli, *Wav2vec 2.0: A framework for self-supervised learning of speech representations*, 2020. DOI: 10.48550/ARXIV.2006.11477. [Online]. Available: https://arxiv.org/abs/2006.11477.

[14] W. Xu, J. He and Y. Shu, 'Transfer learning and deep domain adaptation,' in *Advances and Applications in Deep Learning*, M. A. Aceves-Fernandez, Ed., Rijeka: IntechOpen, 2020, ch. 3. DOI: 10.5772/intechopen.94072. [Online]. Available: https://doi.org/10.5772/intechopen.94072.

[15] 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)*, 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[16] J. Rugayan, 'A deep learning approach to spoken language aquisition,' Department of Electronic Systems, NTNU – Norwegian University of Science and Technology, [Master Thesis], Jun. 2021. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2778188.

[17] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[18] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, 'Array programming with NumPy,' *Nature*, vol. 585, pp. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.

[19] J. D. Hunter, 'Matplotlib: A 2d graphics environment,' *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[20] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer and K. Vesely, 'The kaldi speech recognition toolkit,' in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Catalog No.: CFP11SRW-USB, Hilton Waikoloa Village, Big Island, Hawaii, US: IEEE Signal Processing Society, Dec. 2011.

[21] D. Baby, *Training kaldi models with custom features*. [Online]. Available: `https://deepakbaby.in/post/kaldi-custom-features/` (visited on 07/06/2022).