Deep Scattering and End-to-End Speech Models towards Low Resource Speech Recognition

University of HUDDERSFIELD

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Iyalla John Alamina

January 8, 2020

Abstract

This thesis investigates and acknowledges the various limitations of Deep Neural Network (DNN) techniques when applied to low resource speech recognition. Various aspects of developing corpora for speech recognition systems are explored. In particular, various Recurrent Neural Network (RNN) techniques were explored to implement end-to-end speech and Language Model (LM). Gated Recurrent Unit (GRU) RNNs were used employed for the language model for a low resourced Wakirike language while bidirectional recurrent neural networks (bi-RNNs) were used to create end-to-end speech recognition model for English language.

Previous systems employed for low resource speech recognition involving deep networks included various knowledge transfer mechanisms including hybrid hidden markov models (HMM) to deep neural networks (HMM-DNN) models and those that are HMM alone-based include subspace Gaussian Mixture Models (GMMs). These models are based on the HMM generative model and N-gram language models. However, the model developed in this thesis makes use of an end-to-end discriminative model using the bi-RNN acoustic/speech model augmented using speech features from a specialised light weight convolution network-the Deep Scattering Network (DSN). While the light weight DSN helped to reduce the training complexity, at the same time by focusing on end-to-end with Connectionist Temporal Classification (CTC) decoding, the speech model was compressed into a one step process rather than a three-step process requiring an Acoustic Model (AM), Language Model (LM) and phonetic dictionary. The research therefore shows that it is possible to use this compacting strategy in addition to augmented speech features required for speech pattern recognition by deploying deep scattering network features with higher dimensional vectors when compared to traditional speech features.

Dedication

To the praise and glory of our God and of His Christ.

Acknowledgements

I thank the members supervisory team including Dr David Wilson and Dr Simon Parkinson for the invaluable guidance and keen interest throughout my research.

I also acknowledge my parents (Prof. Mrs. Jane Alamina and Dr. P. T. Alamina) for immense support shown. My wife, children (Topaz and Jade) and family members have also stood by given and given all the encouragement I could ever need. Thank you. Finally, to all who have said a prayer and have contributed towards my studies or well being, I am grateful to you all.

Copyright statement

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns any copyright in it (the "Copyright") and s/he has given The University of Huddersfield the right to use such copyright for any administrative, promotional, educational and/or teaching purposes.

ii. Copies of this thesis, either in full or in extracts, may be made only in accordance with the regulations of the University Library. Details of these regulations may be obtained from the Librarian. This page must form part of any such copies made.

iii. The ownership of any patents, designs, trademarks and any and all other intellectual property rights except for the Copyright (the "Intellectual Property Rights") and any reproductions of copyright works, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property Rights and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property Rights and/or Reproductions

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Automatic Speech Recognition (ASR) is a subset of Machine Translation that takes a sequence of raw audio information and translates or matches it against the most likely sequence of text as would be interpreted by a human language expert. In this thesis, Automatic Speech Recognition will also be referred to as ASR or speech recognition for short.

It can be argued that while ASR has achieved excellent performance in specific applications, much is left to be desired for general purpose speech recognition (Yu and Deng, 2016). While commercial applications like Google voice search and Apple Siri give evidence that this gap is closing, there still are yet other areas within this research space that speech recognition task is very much an unsolved problem.

It is estimated that there are close to 7,000 human languages in the world (Besacier et al., 2014a) and yet for only a fraction of this number have there been efforts made towards practical ASR systems. The level of ASR accuracy that has been so far achieved are based on large quantities of speech data and other linguistic resources used to train models for ASR. These models which depend largely on pattern recognition techniques degrade tremendously when applied to different languages other than the languages that they were trained or designed for (Besacier et al., 2014b, Rosenberg et al., 2017). More specifically, the collection of sufficient amounts of linguistic resources required to create accurate models for ASR are particularly laborious and time consuming sometimes extending to decades (Goldman, 2011, Stan et al., 2016). Research, therefore, geared towards alternative approaches towards developing is ASR systems that are reproducible across languages lacking

the resources required to build robust systems is apt.

1.1 ASR As a Machine Learning problem

Automatic Speech Recognition can be put into a class of Machine Learning problems described as sequence pattern recognition because an ASR attempts to discriminate a pattern from the sequence of speech utterances.

One immediate problem realised with this definition leads us to discuss statistical speech models that address how to handle the problem described in the following paragraph.

Speech is a complex phenomena that begins as a cognitive process and ends up as a physical process (Becchetti and Ricotti, 1998). The process of automatic speech recognition attempts to reverse engineer steps back from the physical process to the cognitive process giving rise to latent variables or mismatched data or loss of information from interpreting speech information from one physiological layer to the next.

It has been acknowledged in the research community (Deng and Li, 2013, Watanabe and Chien, 2015) that work being done in Machine Learning has enhanced the research of automatic speech recognition. Similarly any progress made in ASR usually constitute contributions to enhancements made in the Machine Learning field. This also may be attributed to the fact that speech recognition in itself is a sequence pattern recognition problem subclass of machine learning. Therefore techniques within speech recognition could be applied generally to sequence pattern recognition problems at large.

The two main approaches to Machine Learning problems historically involve two methods rooted in statistical science. These approaches are generative and discriminative models. From a computing science perspective, the generative approach is a brute-force approach while the discriminative model uses a rather heuristic approach to Machine Learning. This chapter presents the introductory ideas behind these two approaches and establishes the motivation for the proposed models used in this research for low resource speech recognition, as well as introducing the Wakirike language as the motivating language case study.

1.2 Generative-Discriminative Speech Models disambiguation

In chapter 2, the Hidden Markov Model (HMM) is examined as a powerful and major driver behind generative modelling of sequential data like speech. Generative models are data-sensitive models because they are derived from the data by accumulating as many different features which can be seen and make generalisations based on observed parameters. The discriminative model, on the other hand, has a heuristic approach to form a classification. Rather than using features of the data directly, the discriminative method attempts to parameterise the data based on initial constraints(Lasserre et al., 2006). It is therefore concluded that the generative approach uses a bottom-to-top strategy starting with the fundamental structures to determine the overall structure, while the discriminative method uses a top-to-bottom approach starting with the big picture and then drilling down to determine the fundamental structures.

Ultimately, generative models for Machine Learning learning can be interpreted mathematically as a joint distribution that produces the highest likelihood of outputs and inputs based on a predefined decision function. The outputs for speech recognition being the sequence of words and the inputs for speech being the audio waveform or equivalent speech sequence. More specifically,

$$d_y(\mathbf{x};\lambda) = p(\mathbf{x}, y;\lambda) = p(\mathbf{x}|y;\lambda)p(y;\lambda) \tag{1.1}$$

where $d_y(\mathbf{x};\lambda)$ is the decision function of y for data labels $\mathbf{x}$. This joint probability expression given as $p(\mathbf{x}|y;\lambda)$ can also be expressed as the conditional probability product in equation (1.1). In this equation, λ predefines the nature of the distribution referred to as model parameters (Deng and Li, 2013).

Similarly, Machine Learning discriminative models are described mathematically as the conditional probability defined by the generic decision function below:

$$d_y(\mathbf{x};\lambda) = p(y|\mathbf{x};\lambda) \tag{1.2}$$

It is clearly seen that the discriminative paradigm follows a much more direct

approach to pattern recognition. Although this approach appears cumbersome to model, this research leans towards this direct approach. However, what the discriminative model gains in discriminative modularity, it loses in the model parameter estimation of (λ) in equation (1.1) and (1.2) (Gales et al., 2012). As this research investigates, although the generative process is able to generate arbitrary outputs from learned inputs, its major drawback is the direct dependence on the training data from which the model parameters are learned. Specific characteristics of various Machine Learning models are reserved for later chapters, albeit the heuristic nature of the discriminative approach, which means not directly dependent on the training data, gains over the generative approach as discriminative models are able to better compensate for latent variables.

In the case of speech signals, the original signal is corrupt and the intended information message attenuated when the signal undergoes physiologic transformations of the speaking and hearing process and moves from one speech production mechanism mentioned in section 1.1 to the next. The theme of pattern recognition through arbitrary layers of complexity is reinforced in the notion of deep learning Deng et al. (2014) as an attempt to learn patterns from data at multiple levels of abstraction. Thus while shallow Machine Learning models like Hidden Markov Models (HMMs) define latent variables for fixed layers of abstraction, deep Machine Learning models handle hidden/latent information for arbitrary layers of abstraction determined heuristically. As deep learning mechanisms are typically implemented using Deep Neural Networks, this work applies deep Recurrent Neural Networks as an end-to-end discriminative classifier for speech recognition. This is a so called "end-to-end model" because it adopts the top-to-bottom Machine Learning approaches. Unlike the typical generative classifiers that require sub-word acoustic models, the end-to-end models develop algorithms at higher levels of abstraction as well as the lower levels of abstraction. In the case of the model utilised in this research, the levels of abstraction include sentence/phrase, words and character discrimination. A second advantage of the end-to-end model is that because the traditional generative models require various stages of modeling including an acoustic, language and lexicon, the end-to-end discriminating multiple levels of abstractions simultaneously only requires a single stage process, greatly reducing the quantity of resources required

for speech recognition. From a low resource language perspective this is a desirable behaviour meaning that the model can be learned from an acoustic only source without the need of an acoustic model or a phonetic dictionary. Thus techniques involving deep learning and end-to-end modelling are proposed and have been found to be self-sufficient (Hannun et al., 2014a) with modest results without a language model. However, applying a language model was observed to serve as a correction factor further improving recognition results (Hannun et al., 2014a).

1.3 Low Resource Languages

Another challenge observed in complex Machine Learning models for both generative as well as discriminative learning models is the data intensive nature of the work required for robust classification models. Saon et al. (2015) recommends around 2000 hours of transcribed speech data for robust speech recognition system. As is covered in the next chapter, for new languages, which are low in training data such as transcribed speech, there are various strategies devised for low resource speech recognition. Besacier et al. (2014a) outlines various matrices for bench-marking low resource languages. From the generative speech model interest perspective, reference is made to languages having less than ideal data in transcribed speech, phonetic dictionary and a text corpus for language modelling. For end-to-end speech recognition models interests, the data relevant for low resource evaluation is the transcribed speech and a text corpus for language modelling. It is worth noting that it was observed in Besacier et al. (2014a) that speaker-base often does not affect a language resource status of a language and was often observed that large speaker bases could in fact lack language/speech recognition resources and that some languages having small speaker bases did in fact have sufficient language/ speech recognition resources.

Speech recognition methods investigated in this work are motivated by the Wakirike language discussed in the next section, which is a low resource language by definition. Thus this research looked at low research language modelling for the Wakirike language from a corpus of Wakirike text available for analysis. However, due to the insufficiency of transcribed speech for the Wakirike language, English lan-

guage was substituted and used as a control variable to study low resource effects of a language when exposed to speech models developed in this work.

1.4 The Wakirike Language

The Wakirike municipality is a fishing community comprising 13 districts in the Niger Delta area of the country of Nigeria in the West African region of the continent of Africa. The first set of migrants to Wakirike settled at the mainland town of Okrika between AD860 and AD1515 at the earliest. These early settlers had migrated from Central and Western regions of the Niger Delta region of Nigeria. As the next set of migrants also migrated from a similar region, when the second set of migrants met with the first settlers they exclaimed "we are not different" or "Wakirike" (S., 2008).

Although the population of the Wakirike community from a 1995 report (Simons and Fennig, 2018) is about 248,000, the speaker base is significantly less than stipulated. The language is classified as Niger-Congo and Ijoid languages. The writing orthography is Latin and the language status is 5 (developing) (Simons and Fennig, 2018). This means that although the language is not yet an endangered language, it still isn't thriving and it is being passed on to the next generation at a limited rate.

The Wakirike language was the focus for this research. And End-to-end deep neural network language model was built for the Wakirike language based on the availability of the new testament bible printed edition that was available for processing. The corpus utilized for this thesis work is approximately 9,000 words.

1.5 Research aim and objectives

In this research, we develop speech processing models and language models which deliver robust deep and recurrent neural network implementations towards low resource speech recognition. In particular, we develop a language model based on Gated Recurrent Unit (GRU) for the Wakirike language and a bi-directional Recurrent Neural Network (bi-RNN) speech model for the English language. The aim of

this research is therefore to build competitive and cross-lingual ASR systems in a well-rounded resource conservative manner

The research objectives were as follows:

- Discover fundamental tasks relating to Language learning;
- Discover criteria for creating ASR platforms for new languages;
- Build robust ASR systems using methods that also system resource robust; and
- Build robust ASR systems using fewer resources, that is reduce the amount and/or number of resources required to build ASR systems.

Within this framework, our focus on language learning tasks was on Automatic Speech recognition while the intention was to achieve the last two objectives through one or more of the following means:

i. Reduction of time to train speech models;
ii. Optimisation of sub-tasks and training architecture within the ASR pipeline;
iii. Reduction in the overall time taken to develop speech models;
iv. Make efficient use of training parallelism;
v. Obtain better or state of the art performance; and
vi. Induce model simplicity thereby reducing training and development time without compromising performance.

Furthermore, following the Interspeech 2015 Zero Resource Speech Challenge (Versteegh et al., 2015), this research also fulfilled the objectives of modelling speech at subword, word and syntax level. The Zero Resource Speech Challenge is inspired from infants ability to construct acoustic and language models of speech in an end-to-end manner. At the word and syntax level this research develops a character-based language model that reinforces subword, word and syntax level speech model based on Character-Temporal-Classification CTC.

1.6 Main Contribution to knowledge

This work uses a well-established neural language model for the low resourced language of Wakirike. At the same time this work implements a unique combination of end-to-end deep recurrent neural network models with a robust and state of the art audio signal processing mechanism involving a hierarchical Deep Scattering Network (DSN) to engineer high-dimensional features to compete with current acoustic and deep architectures for speech recognition. While the base-line model gave state of the art performance from a multi-feature input on a Medium Vocabulary Continuous Speech Recognition corpus, with a Character Error Rate of 9.5%; the DSN-feature network was shown to be heading towards model saturation.

1.7 Thesis outline

The engineered systems, methods and supporting literature contained in this thesis report follows the following outline and describe the research outputs of an end-to-end speech recogniser and develops the theory based on the building blocks of the research outputs.

Chapter two introduces the speech recognition pipeline and the generative speech model. Chapter two also outlines the weaknesses in the generative model and describes some of the Machine Learning techniques applied to improve speech recognition performance. The methods and techniques and description of the various tools and metrics for analysis of the research outputs are described and examined in Chapter three.

Various Low speech recognition methods are reviewed and the relevance of this study is also highlighted. Chapter four describes Recurrent Neural Networks (RNNs). Starting with Multi Layer Perceptrons (MLPs), we go on to specialised recurrent neural networks including Long Short-Term Memory (LSTM) networks and the Gated Recurrent Unit (GRU) are detailed. These recurrent neural network units form building blocks of the language model for Wakirike language implemented in this work.

Chapter five explains the wavelet theorem as well as the deep scattering spectrum. The chapter develops the theory from Fourier transform and details the

significance of using the scattering transform as a feature selection mechanism for low resource recognition.

Chapters six and seven give descriptions of the models developed by this thesis and details the experimental setup along with the results obtained. Chapter eight is the conclusion of the work and recommendations for further study.

1.8 Chapter Summary

Amidst seeming large success of speech-to-text technology referred to as Automatic Speech Recognition (ASR), there are still areas in which ASR technology struggle to perform up to the minimum acceptable level. Situations such as very noisy environments and far field speech recognition constitute common physical scenarios where ASR performance degrades significantly. Another non-physical area in which ASR falls short of acceptable performance and chosen as the focus of this research is the area of low-resource speech recognition. This is the scenario where languages not rich in linguistic resources are unable to use existing resources and algorithms used in languages rich in linguistic and ASR resources, to perform automatic speech recognition.

As this chapter identifies, the ASR problem is traditionally a () problem that models where speech models are trained from language-specific data. While these speech models may perform well for the languages the models were trained for, when introduced to a different language, having a different set of learning features, these pre-trained models fall short of expected performances for these new languages. Moreover, if the new languages do not possess a rich set of linguistic features, including resources such as aligned speech and an online text corpus amongst others (Besacier et al., 2014b), it becomes time-consuming and extremely laborious to develop new ASR models for speech recognition for these so-called ASR "low-resource" languages.

This chapter also introduces the Wakirike language as a low resource language and the motivating language for this research. In addition, the various machine learning architectures used in this research for low resource speech recognition for the Wakirike and for English language are reviewed. In particular, Deep Neural

Networks (DNNs) are highlighted as choice algorithms in speech recognition, and then, the Chapter goes on to describe the research novelty and the outline of this thesis.

Chapter 2

Literature Review

The speech recogniser developed in this thesis is based on an end-to-end discriminative deep recurrent neural network. Two models were developed. The first model, a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN), was used to develop a character-based (). The second model is a bi-directional Recurrent Neural Network (bi-RNN) is an end-to-end speech model capable of generating word sequences based on learned character sequence outputs. This chapter describes the transition from generative speech models to these discriminative end-to-end recurrent neural network models. Low speech recognition strategies are also discussed and the contribution to knowledge gained by using character-based discrimination as well as introducing deep scattering features to the bi-RNN speech model is brought to light.

2.1 Speech Recognition Overview

Computer speech recognition takes raw audio speech and converts it into a sequence of symbols. This can be considered as an analog to digital conversion as a continuous signal becomes discretised. The way this conversion is done is by breaking up the audio sequence into very small packets referred to as frames and developing discriminating parameters or features for each frame. Then, using the vector of features as input to the speech recogniser.

A statistical formulation (Young et al., 2002) for the speech recogniser follows given that each discretised output word in the audio speech signal is represented as

a vector sequence of frame observations defined in the set $\mathbf{O}$ such that

$$\mathbf{O} = \mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T. \tag{2.1}$$

Equation 2.1 says that, at each discrete time t, we have an observation $\mathbf{o}_t$, which is, in itself is a vector in R^D. From the conditional probability, it can be formulated that certain word sequences from a finite dictionary are most probable given a sequence of observations. That is:

$$arg\max_t\{P(w_i|\mathbf{O})\} \tag{2.2}$$

Section 2.1.2 outline some challenges of speech recognition which result in the analysis of $P(w_i|\mathbf{O})$ being no trivial task. The divide and conquer strategy therefore employed uses Bayes formulation to simplify the problem. Accordingly, the argument that maximises the probability of an audio sequence given a particular word multiplied by the prior probability of that word is equivalent to the original posterior probability required to solve the original speech recognition problem. This is summarised by the following equation

$$P(w_i|\mathbf{O}) = \frac{P(\mathbf{O}|w_i)P(w_i)}{P(\mathbf{O})} \tag{2.3}$$

According to Bayes' rule, the posterior probability is obtained by multiplying a certain likelihood probability by a prior probability. The likelihood in this case, $P(\mathbf{O}|w_i)$, is obtained from a Hidden Markov Model (HMM) parametric model such that rather than estimating the observation densities in the likelihood probability, these are obtained by estimating the parameters of the HMM model. The HMM model explained in the next section gives a statistical representation of the latent variables of speech at a mostly acoustic level.

The second parameter in the speech model, interpreted from Bayes' formula, is the prior probability of a given word. This aspect of the model is the language model which is reviewed in section 2.2.1.

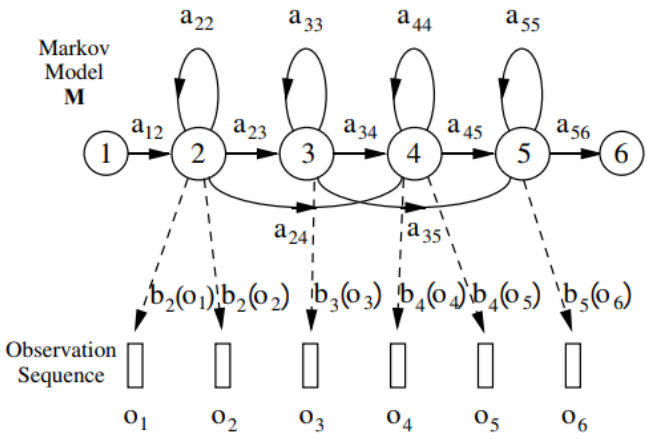

Figure 2.1: HMM Generative Model
Young et al. (2002)

2.1.1 HMM-based Generative speech model

A HMM represents a finite state machine where a process transits a sequence of states from a set of fixed states (Gales et al., 2008, Young et al., 2002). The overall sequence of transitions will have a start state, an end state and a finite number of intermediate states all within the set of finite states. Each state transition emits an output observation that represents the current internal state of the system.

In an HMM represented in Figure 2.1 there are two important probabilities. The first is the state transition probability given by a_{ij} this is the probability to move from state i to state j. The second probability b_j is the probability that an output observation is emitted when in a particular state.

Where $\mathbf{O}$, are the output observations and M is the HMM. Given that X represents the sequence of states transitioned by a process, a HMM defines the joint probability of X and the output probabilities given the HMM in the following representation:

$$P(\mathbf{O}|M) = \sum_{X} a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(\mathbf{o}_t) a_{x(t)x(t+1)} \tag{2.4}$$

Generally speaking, the HMM formulation presents 3 distinct challenges. The first is the likelihood of a sequence of observations given in equation 2.4 above. The next two, described later, is the inference and the learning problem. While the inference problem determines the sequence of steps given the emission probabilities, the learning problem determines the HMM parameters, that is the initial transition and emission probabilities of the HMM model.

For the case of the inference problem, the sequence of states can be obtained by determining the sequence of states that maximises the probability of the output sequences.

2.1.2 Challenges of Speech Recognition

The realised symbol is assumed to have a one to one mapping with the segmented raw audio speech. However, the difficulty in computer speech recognition is the fact that there is a significant amount of variation in speech that would make it practically intractable to establish a direct mapping from segmented raw speech audio to a sequence of static symbols. The phenomena known as co articulation has it that there are several different symbols having a mapping to a single waveform of speech in addition to several other varying factors including the speaker mood, gender, age, the medium of speech transduction, the room acoustics, et cetera.

Another challenge faced by automated speech recognisers is the fact that the boundaries of the words are not apparent from the raw speech waveform. A third problem that immediately arises from the second is the fact that the words from the speech may not strictly follow the words in the selected vocabulary database. Such occurrence in speech recognition research is referred to as "out Of vocabulary" (OOV) terms. It is reasonable to approach these challenges using a divide and conquer strategy. In this case, the first step would be to make provision for word boundaries. This first step in speech recognition is referred to as the isolated word recognition case (Young et al., 2002).

2.1.3 Challenges of low speech recognition

Speech recognition for low resource languages poses another distinct set of challenges. In chapter one, low resource languages were described to be languages lacking in resources required for adequate Machine Learning of models needed for generative speech models. These resources are described basically as a text corpus for language modelling, a phonetic dictionary and transcribed audio speech for acoustic modelling. Figure 2.2, illustrates how resources required for speech recognition are utilised. It is observed that in addition to the three resources identified other processes are required for the speech decoder to function normally. For example,

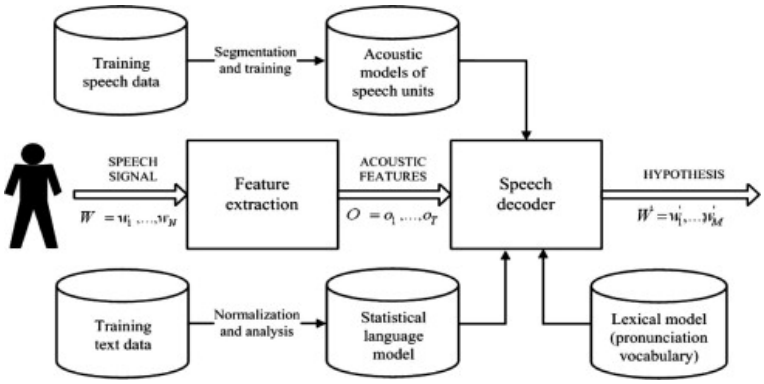

Figure 2.2: Automatic Speech Recognition Pipeline
Besacier et al. (2014a)

aligned speech would also need to be segmented into speech utterances to ensure that the computer resources are used conservatively.

In terms of data collection processing Besacier et al. (2014a) enumerate the challenges for developing low resource ASR systems to include the fact that phonologies (or language sound systems) differ across languages, word segmentation problems, fuzzy grammatical structures, unwritten languages, lack of native speakers having technical skills and the multidisciplinary nature of ASR constitute impedance to ASR system building.

2.2 Low Resource Speech Recognition

In this system building speech recognition research, the focus was on the development of a language model and an end-to-end speech model comparable in performance to state of the art speech recognition system consisting of an acoustic model and a language model. Low resource language and acoustic modelling are now reviewed keeping in mind that little work has been done on low-resource end-to-end speech modelling when compared to general end-to-end speech modelling and general speech recognition as a whole.

From an engineering perspective, a practical means of achieving low resource speech modelling from a language rich in resources is through various strategies of the Machine Learning sub-field of transfer learning.

Transfer learning takes the inner representation of knowledge derived from training algorithm used from one domain and applying this knowledge in a similar domain

having different set of system parameters(Ramachandran et al., 2016). Early work of this nature for speech recognition is demonstrated in (Vu and Schultz, 2013) where multi-layer perceptrons were used to train multiple languages rich in linguistic resources. In a later section entitled "speech recognition on a budget", a transfer learning mechanism involving deep neural networks from (Kunze et al., 2017) is described.

2.2.1 Low Resource language modelling

General language modelling is reviewed and then Low resource language modelling is discussed in this section. In section 2.1, recall from equation 2.3, the general speech model influenced by Bayes' theorem.

$$P(w_i|\mathbf{O}) = \frac{P(\mathbf{O}|w_i)P(w_i)}{P(\mathbf{O})} \tag{2.5}$$

The speech recognition model is a product of an acoustic model (likelihood probability),$P(\mathbf{O}|w_i)$ and the language model (prior probability),$P(w_i)$. The development of language models for speech recognition is discussed in Juang and Furui (2000) and Young (1996).

Language modelling formulate rules that predict linguistic events and can be modelled in terms of discrete density $P(W)$, where $W = (w_1, w_2, ..., w_L)$ is a word sequence. The density function $P(W)$ assigns a probability to a particular word sequence W. This value determines how likely the word is to appear in an utterance. A sentence with words appearing in a grammatically correct manner is more likely to be spoken than a sentence with words mixed up in an ungrammatical manner, and, therefore, is assigned a higher probability. The order of words therefore reflect the language structure, rules, and conventions in a probabilistic way. Statistical language modeling therefore, is an estimate for $P(W)$ from a given set of sentences, or corpus.

The prior probability of a word sequence $\mathbf{w} = w_1, \ldots, w_k$ required in equation (2.2) is given by:

$$P(\mathbf{w}) = \prod_{k=1}^{K} P(w_k|w_{k-1}, \ldots, w_1) \tag{2.6}$$

The N-gram model is formed by the conditioning of the word history in equation

2.6. This therefore becomes:

$$P(\mathbf{w}) = \prod_{k=1}^{K} P(w_k|w_{k-1}, w_{k-2}, \ldots, w_{k-N+1}) \tag{2.7}$$

N is typically in the range of 2-4.

N-gram probabilities are estimated from training corpus by counting N-gram occurrences. This is plugged into maximum likelihood (ML) parameter estimate. For example, Given that N=3 then the probability that three words occurred is assuming $C(w_{k-2}w_{k-1}w_k)$ is the number of occurrences of the three words $C(w_{k-2}w_{k-1})$ is the count for $w_{k-2}w_{k-1}w_k$ then

$$P(w_k|w_{k-1}, w_{k-2}) \approx \frac{C(w_{k-2}w_{k-1}w_k)}{C(w_{k-2}w_{k-1})} \tag{2.8}$$

The major problem with maximum likelihood estimation scheme is data sparsity. This can be tackled by a combination of smoothing techniques involving discounting and backing-off. The alternative approach to robust language modelling is the so-called class based models (Brown et al., 1992, **?**) in which data sparsity is not so much an issue. Given that for every word w_k, there is a corresponding class c_k, then,

$$P(\mathbf{w}) \prod_{k=1}^{K} P(w_k|c_k)p(c_k|c_{k-1}, \ldots, c_{k-N+1}) \tag{2.9}$$

In 2003, Bengio et al. (2003) proposed a language model based on neural Multi-Layer Perceptrons (MLPs). These MLP language models resort to a distributed representation of all the words in the vocabulary such that the probability function of the word sequences is expressed in terms of these word-level vector representations. The performance of the MLP-based language models was found to be, in cases for models with large parameters, better than the traditional n-gram models.

Improvements over the MLPs still using neural networks over the next decade include works of Luong et al. (2013), Mikolov et al. (2011), Sutskever et al. (2014), involved the utilisation of deep neural networks for estimating word probabilities in a language model. While a Multi-Layer Perceptron consists of a single hidden layer, in addition to the input and output layers, a deep network, in addition to having several hidden layers, is characterised by complex structures that render the architecture

beyond the basic feed forward nature. Particularly, for Recurrent Neural Network (RNN) architectures, we also have some feedback neurons in addition to the forward neurons where data flows in the reverse direction, from output to input.

Furthermore, the probability distributions in these deep neural networks were either based upon word or sub-word models, this time having representations which also conveyed some level of syntactic or morphological weights to aid in establishing word relationships. These learned weights are referred to as token or unit embedding (Pennington et al., 2014).

For the neural network implementations so far seen, a large amount of data is required due to the nature of words to have large vocabularies, even for medium-scale speech recognition applications. Kim et al. (2016) on the other hand took a different approach to language modelling taking advantage of the long-term sequence memory of long-short-term memory cell recurrent neural network (LSTM-RNN) to model a language based on characters rather than on words. This greatly reduced the number of parameters involved and therefore the complexity of implementation. This method is forms the basis of the Wakirike language model implementation in this work due to the low resource constraints gains made when using a character-level language model.

Other low resource language modelling strategies employed for the purpose of speech recognition was demonstrated by Xu and Fung (2013). The language model developed in that work was based on phrase-level linguistic mapping from a high resource language to a low resource language using a probabilistic model implemented using a Weighted Finite State Transducer (WFST). This method uses WFST rather than a neural network due to scarcity of training data required to develop a neural network. However, it did not gain from the high non linearity ability of a neural network model to discover hidden patterns in data, being a shallower Machine Learning architecture.

The language model implemented in this thesis report uses a character-based Neural network language model that employs a recurrent neural network similar to that of Kim et al. (2016), however based on Gated Recurrent Unit (GRU) RNNs (Cho et al., 2014), for the Okrika language which is a low resource language, bearing in mind that the character level network will reduce the number of parameters

required for training, just enough to develop a working language model for the purpose of speech recognition.

2.2.2 Low Resource Acoustic and speech modelling

Two transfer learning techniques for acoustic modelling investigated by Povey et al. (2011a) and Ghoshal et al. (2013) respectively include the Sub-space Gaussian Mixture Model (SGMM) and the use of pretrained hidden layers of a deep neural network trained multilingually as a means to initialise weights for an unknown language. This second method of low resource modelling has been informally referred to as the swap-hat method.

Recall that one of the challenges associated with new languages is that phonetic systems differ from one language to another. Transfer learning approaches attempt however to recover patterns common to seemingly disparate systems and model these patterns.

The physiologic speech production mechanism is based on the premise that sounds are produced by approximate movements and positions of articulators that comprise the human speech production system and that this mechanism is common to all humans. It is possible to model dynamic movement from between various phones as tied state mixture of Gaussians. These dynamic states modelled using Gaussian Mixture Model (GMM) are also called senones. Povey et al. (2011a) postulated a method to factorize these Gaussian mixtures into a globally shared set of parameters that are non-dependent individual HMM states. These factorisations model senones that are not represented in original data and thought to be a representation of the overall acoustic space. While preserving individual HMM states, the decoupling of the shared space and its reuse makes SGMMs a viable candidate for transfer learning of acoustic models for new languages.

The transfer learning procedure proposed in Ghoshal et al. (2013) employed the use of Deep Neural Networks, in particular Deep Belief Network (DBN)s (Bengio et al., 2007). Deep Belief Networks are pretrained, layer-wise stacked s (RBMs)(Smolensky, 1986). The output of this network trained on senones correspond to HMM context dependent states. However, by decoupling hidden layers from outer and output layers and fine-tuned to a new language, the network is shown to be insensitive to the

choice of languages analogous to global parameters of SGMMs. The 7-layer, 2000 neuron per layer network used did not utilise a bottleneck layer corresponding to triphone states trained on MFCC features (Grezl and Fousek, 2008).

2.3 Groundwork for low resource end-to-end speech modelling

The underpinning notion of this work is firstly a departure from the extra processing required for bottom-to-top that comes as a byproduct of the generative process sponsored by the HMM-based speech models. This has an advantage of simplifying the speech pipeline from acoustic, language and phonetic model to just a speech model that approximates the same process. Secondly, the model developed seeks to overcome the data intensity barrier and was seen to achieve measurable results for GRU RNN language models. Therefore adopting the same character-based strategy, this research performed experiments using the character-based bi-directional recurrent neural networks (BiRNN). However, BiRNNs researchers have found them like other deep learning algorithms, too be quite data intensiveHannun et al. (2014a). The next paragraphs introduce Deep-speech BiRNNs and the two strategies for tackling the data intensity drawback as related with low resource speech recognition.

2.3.1 Deep speech

Up until recently, speech recognition research has been centred on improvements of the HMM-based acoustic models. This has included a departure from generative training of HMM to discriminative training (Woodland and Povey, 2000) and the use of neural network precursors to initialise the HMM parameters (Mohamed et al., 2012). Although these discriminative models brought improvements over generative models, being HMM dependent speech models they lacked the end-to-end nature. This means that they were subject to training of acoustic, language and phonetic models. With the introduction of the Connectionist Temporal Classification (CTC) loss function, Graves and Jaitly (2014) finally found a means to end-to-end speech recognition departing from HMM-based speech recognition.

The architecture of the Deep-speech end-to-end speech recognition model Hannun et al. (2014b) follows an end-to-end Bi-directional Recurrent Neural Network (BiRNN) and CTC loss function (Graves et al., 2006). The CTC loss function uses a modified beam search to sum over all viable sequences of the input and output sequence space alignments so as to maximise the likelihood of the output sequence characters.

2.3.2 Speech Recognition on a low budget

In this section, a recent transfer learning speech model (Kunze et al., 2017) that has some characteristics similar to the speech model developed in this thesis is reviewed. The end-to-end speech model described by Kunze et al. (2017) is based on that developed by Collobert et al. (2016) and is based on deep convolutional neural networks rather than the Bi-RNN structure proposed by this work. In addition it uses a loss function based on the AutoSegCriterion which is claimed to work competitively with raw audio waveform without any preprocessing. The main strategy for low resource management in their system was the freezing of some layers within the convolutional network layer. The low resource mechanisms used in this work includes the use of a unique scattering network being used as input features for the BiRNN model. The fascinating similarity between the end-to-end BiRNN speech model developed in this work and the transfer learning model in Kunze et al. (2017) is the fact that the scattering network input is equivalent to the output of a light-weight convolutional neural network Hannun et al. (2014b). Therefore the proposed system then approximates a combination of a recurrent neural network as well as a convolution neural network without the overhead of actually training a convolutional neural network (CNN)(Szegedy et al., 2015).

Introduction of the unique scattering network is discussed in the next section. It is worthy to note however that Kunze et al. (2017) uses a CNN network only while (Amodei et al., 2016) uses both RNN and CNN network. The speech model in this thesis uses a BiRNN model and combines an RNN model with the scattering layer which represents a light-weight low resource friendly pseudo enhanced CNN backing. What is meant by pseudo enhanced CNN backing is reserved for the next section, however, therefore, the proposed speech model in this thesis stands to gain

from an enhanced but lightweight CNN combined with RNN learning.

2.3.3 Adding a Scattering layer

In Machine Learning, training accuracy is greatly improved through a process described as feature engineering. In feature engineering, discriminating characteristics of the data are enhanced at the same time non-distinguishing features constituting noise are removed or attenuated to a barest minimum. A lot of the components signal speech signal are due to noise in the environment as well as signal channel distortions such as losses due to conversion from audio signals to electrical signal in the recording system.

In Figure 2.2, feature engineering is done at the feature extraction stage of the ASR pipeline. It has been shown that a common technique using Mel Frequency Cepstral Coefficients (MFCC) (Davis and Mermelstein, 1980) can represent speech in a stable fashion that approximate how the working of the human auditory speech processing and is able to filter useful components in the speech signal required for human speech hearing. Similar feature processing schemes have been developed include Perceptual Linear Prediction (PLP) (Hermansky, 1990) and RelAtive SpecTrAl (RASTA) (Hermansky and Morgan, 1994).

The scattering spectrum defines a locally translation invariant representation of a signal resistant to signal deformation over extended periods of time spanning seconds of the signal (Andén and Mallat, 2014). While Mel-frequency cepstral coefficients (MFCCs) are cosine transforms of Mel-frequency spectral coefficients (MFSCs), the scattering operator consists of a composite wavelet and modulus operation on input signals.

Over a fixed time, MFSCs measure signal energy having constant Q bandwidth Mel-frequency intervals. This procedure is susceptible to time-warping signal distortions since these information often reside in the high frequency regions discarded by Mel-frequency intervals. As time-warping distortions is not explicit classifier objective when developing these filters, there is no way to recover such information using current techniques.

In addition, short time windows of about 20 ms are used in these feature extraction techniques since at this resolution speech signal is mostly locally stationary.

Again, this resolution adds to the loss of dynamic speech discriminating information on signal structures that are non-stationary at this time interval. To minimize this loss Delta-MFCC and Delta-Delta-MFCCs (Furui, 1986) are some of the means developed to capture dynamic audio signal characterisation over larger time scales.

By computing multi-scale co-occurrence coefficients from a wavelet-modulus operation, Andén and Mallat (2011) show that non-stationary attributes of a signal lost by MFSC coefficients is regained in multi scale co-occurrence coefficients. The scattering transform therefore, derives a scattering representation with an interpretation similar to MFSC-like measurements. Together with higher-order co-occurrence coefficients, deep scattering spectrum coefficients represent audio signals similar to models based on cascades of constant-Q filter banks and rectifiers. In particular, second-order co-occurrence coefficients contain relevant signal information capable of discriminating dynamic information lost to the MFCC analog over several seconds and therefore a more efficient discriminant than the MFCC representation. Second-order co-occurrence coefficients calculated by cascading wavelet filter banks and rectified using modulus operations have been evaluated as equivalent to a light-weight convolutional neural networks whose output posteriors are computed at each layer instead of only at the output layer (Mallat, 2016).

The premise for this work is that low speech recognition can be achieved by having higher resolution features for discrimination as well as using an end-to-end framework to replace some of the cumbersome and time-consuming hand-engineered domain knowledge required in the standard ASR pipeline. In addition, this research work makes contributions to the requirements for the two tracks specified in the Zero Resource challenge of 2015 (Versteegh et al., 2015). The first requirement is sub-word modelling satisfied by using deep scattering network and the second that of spoken term discovery criteria being satisfied by the end-to-end speech model supplemented with a language model.

2.4 Chapter Summary

Chapter 1 introduces the key terms Discriminative and Generative classification. In this Chapter, these two different classification mechanisms are compared and con-

trasted as they relate to speech recognition. The Hidden Markov Model (HMM) is considered as the key Generative algorithm used in speech recognition. This chapter discusses the HMM algorithm and outlines its limitations in speech recognition. Other challenges associated with speech recognition and low speech recognition are discussed.

The method taken by this research towards low resource recognition is described as well as current related research in speech recognition involving low resource discriminative strategies. In addition, transfer learning approaches in low speech speech recognition are previewed. This chapter also outlines the addition of a scattering layer towards increasing discriminating feature tangibility for speech recognition.

Chapter 3

Methodology

This chapter describes the system building methodology (Nunamaker Jr et al., 1990) as applied to deep recurrent architectures for speech recognition. As this approach involves theory building, system development, experimentation and observation, this chapter describes the procedures which were incorporated in order to achieve the aims and objectives of this research.

In order to arrive at the initial research questions and hypothesis a literature survey of speech processing advances was carried out.

The initial research topic was centred around a language learning companion. Thus, a mini survey was conducted on recipients' use of technology in general learning. However, after the literature survey, the research was narrowed down to core language technology assistive features and speech recognition for low resource languages was the chosen area of research focus.

This research develops several software systems based on knowledge acquired from the literature survey in order to gain deeper understanding into the state of the art research results as well as building upon baseline systems in order to achieve the research aims and objectives. It was through this methodology that the final systems developed in chapters six and seven were designed and developed as a unique combination of existing research systems. While the system built in chapter seven is a combination of systems in order to generate knowledge in the field of speech recognition, the value added from the system built in chapter six relates to using already successful methods in speech recognition on a new language having linguistic data challenges.

3.1 Assumptions

This research makes the following assumptions and claims.

i. The first assumption is that Software engineering systems are successfully developed using an incremental and iterative manner of increasing complexity.

ii. End-to-end speech models are more conservative on actual software engineering complexity and in that respect are said to be utilised towards low resource speech recognition.

iii. End-to-end speech recognition has been made possible using recurrent neural networks (RNNs) and connectionist temporal classification (CTC) algorithm.

iv. By having a higher number of features, Deep scattering networks (DSNs) can better detect speech than state of the art Mel Frequency Cepstral Coefficients (MFCCs).

v. There is knowledge to be gained in the application of speech models to new languages.

Based on the above assumptions this research proposes that there is much knowledge to be gained from combining the use of Scatter transform features with RNNs and application of current deep RNNs in the modelling of the Wakirike language.

3.2 Speech Processing software and tools

This research set out to build and evaluate several speech processing systems. Some of the systems were built by hand from scratch; however, the end products were adaptations of already existing open source speech recognition research projects. The systems and platforms adapted for this research include the following:

- CMUSphinx
- Kaldi
- Mozilla DeepSpeech
- Scatternet toolbox

- Matlab
- Tensorflow
- Choregraphe

While the research sought to focus on speech models for the Wakirike language, several other sub systems were required for but development of the baseline models in addition to the final model the following system development steps were taken to arrive at the final output models:

- Auto-correlation experiments
- Experiments with Nao robot
- CMUSphinx Digits speech recogniser
- Digit speech recogniser using Kaldi
- Python based speech alignment experiments
- Sequence-to-sequence grapheme-to-phoneme (G2P) model
- TensorFlow sequence-to-sequence character-to-diacritically-labelled-character model
- GRU language model for Wakirike language based on TensorFlow
- Bi-Directional LSTM-based end-to-end speech model
- ESP-Net experiments

In the following sections, the tools utilised for the systems developed and how they were utilised is discussed. Subsequently, the actual systems developed incrementally towards the final models are described.

3.2.1 CMUSphinx

The CMU Sphinx recogniser system is illustrated in Figure 3.1. In a speech application or experiment, the recogniser is called within the user application and is fed with input and other control parameters that determines the recogniser behaviour.

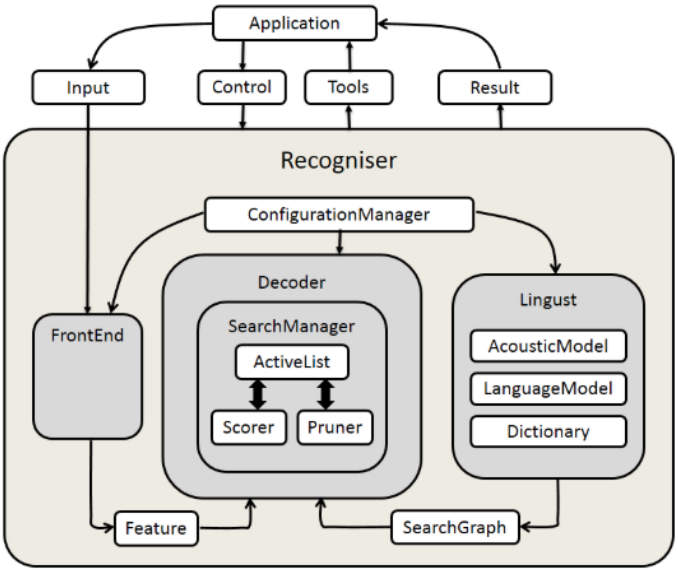

Figure 3.1: CMU Sphinx4 recogniser system

From the illustration, it is observed how the components of feature extraction, acoustic modelling, language modelling and decoding are linked within the CMU Sphinx system. Note that for identification and clarity classes/modules are capitalised in the following paragraphs.

In the CMU Sphinx realisation, the `FrontEnd` module implements feature extraction. The `Linguist` module implements the acoustic modelling and the language model component. Finally, the `Decoder` module implements a decoder. The `ConfigurationManager` class is used to determine the behaviour of the recogniser by specifying the parameters of the other modules.

From this implementation, the `FrontEnd` processor is the signal processing unit of Sphinx-4 parameterising signals using various implementations into a final sequence of `Features`. The `Linguist` is in charge of language and pronunciation modelling. This includes phonetic information from the `Dictionary` and structural information from one or more sets of `LanguageModels` and `AcousticModels`. The output of the `Linguist` is a `SearchGraph`. The `Feature`'s output from the `FrontEnd` and the `SearchGraph` output from the `Linguist` become the input for the `SearchManager` in the `Decoder`. The output of the decoder are `Results` objects. At any time prior to or during the recognition process, the researcher can via his application application issue Controls through the `ConfigurationManager` to each of the modules, and become a partner in the recognition process. The following subsections summarise

the submodules (Walker et al., 2004).

FrontEnd Module

Being consistent with having a "pluggable" framework, CMU Sphinx4 has the ability that most of its components can be replaced and at run-time. This flexibility allows various implementations of the comprising components of the recogniser. Accordingly, the front end supports but is not limited to Mel Frequency Cepstral Coefficients (MFCC), Perceptual Linear Prediction (PLP) and Linear Predictive Coding (LPC) implementations. In addition, comprising modules within the various implementations include support for various signal processing utilities such as Hamming Windows, Discrete Cosine Transform (DCT), Bark Frequency Warping, Mel Frequency Filtering, Cepstral Mean Normalisation (CMN) etc. All the tasks therefore required by the feature extraction process are implemented in this module.

Linguist

The job of the `Linguist` is to model the higher order and lower order grammar content of the audio input. This particular module caters for the acoustic model and the language model. The various `Linguist` implementations allow CMU Sphinx-4 to support different tasks such as traditional Context Free Grammar (CFG), Finite-State Grammars (FSGs), finite-state transducers and small N-gram language models. This module has three pluggable modules representing the `Dictionary`, `LanguageModel` and `AcousticModel`. The `Dictionary` comprises the pronunciation of all the words to be used in the `Decoder`. Sphinx-4 Linguist provides primary support for the CMU Pronouncing Dictionary (Carnegie Mellon University, 2016). The `SearchGraph` produced by the Linguist is capable of sharing parameters such as Gaussian mixtures, transition matrices and mixture weights and Sphinx-4 provides a single Acoustic model supporting acoustic models generated by the Sphinx-3 trainer. Depending on the memory architecture various implementations of the Linguist include the `FlatLinguist`, `DynamicFlatLinguist` and `LexTreeLinguist`. These will either create the `SearchGraph` entirely in memory or on demand. Finally, the `LanguageModel` supports a variety of formats such as `SimpleWorldListGramar` which as the name implies supports a simple word list. The `JSGFGramar` is a BNF-

style platform-independent realisation of the Java Speech API Grammar format. `LMGrammar` produces a bigram model. `FSTGrammar` supports finite-state transducer ARPA FST grammar format. The `SimpleNGramModel` support N-gram model and the `LargeTriGramModel` is suited to optimise memory storage.

Decoder

Provides a pluggable `SearchManager` to simplify decoding. `Decoder` tells `SearchManager` to recognise a set of `Feature` frames. This creates a `Result` object that contains all the paths that have reached a final non-emitting state(i.e. Word endings). Applications can modify the search space and Result object between steps, permitting the application to become a partner in the recognition process. The `SearchManager` is not restricted on any particular implementation, examples include Frame synchronous Viterbi, Bushderby, A*, bi-directional and parallel searches.

Each `SearchManager` uses a token passing algorithm described by (Young, Russel Thornton, 1989). A sphinx-4 token is an object that is associated with a `SearchState` and contains the overall acoustic and language scores of the path at a given point, a reference to the `SearchState`, a reference to an input `Feature` frame, and other relevant information.

The `SearchManager` sub-framework generates `ActiveLists` from currently active tokens in the search trellis by pruning using a pluggable Pruner. These in turn can be modified by the application to perform both relative and absolute beam pruning.

The `SearchManager` sub-framework also communicates with the `Scorer`, a pluggable state probability estimation module that provides state output density values on demand.

Other modules

`ConfigurationManager` allows various module implementations to be combined in various ways. Finally, we illustrate how the `ConfigurationManager` creates Automatic Speech Recognition (ASR) experiments using the CMU-sphinx4 objects described above in the sample code from (Lamere et al., 2003)

```
package com.example;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

import edu.cmu.sphinx.api.Configuration;
import edu.cmu.sphinx.api.SpeechResult;
import edu.cmu.sphinx.api.StreamSpeechRecognizer;

public class TranscriberDemo {
    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        configuration
        .setAcousticModelPath("resource:en-us");
        configuration
                .setDictionaryPath("resource:cmudict-en-us.dict");
        configuration
        .setLanguageModelPath("resource:en-us.lm.bin");

        StreamSpeechRecognizer recognizer = new StreamSpeechRecognizer(
                configuration);
        InputStream stream = new FileInputStream(new File("test.wav"));

        recognizer.startRecognition(stream);
        SpeechResult result;
        while ((result = recognizer.getResult()) != null) {
            System.out.format("Hypothesis: %s\n", result.getHypothesis());
        }
        recognizer.stopRecognition();
    }
}
```

The above java code sample represents a user application. We see three classes being imported. The `Configuration`, `SpeechResult`, and `StreamSpeechRecognizer` class. The `Configuration` object holds resources for the acoustic model, language

model and phonetic dictionary. The `SpeechRecognizer` object has different implementations representing the source of the speech signal. In the above sample the `StreamSpeechRecogniser` class is used to load the speech signal from a wave (.wav) file. However other speech signal sources are available such as the `LiveSpeechRecogniser` which implements loading the speech sound signal from a microphone device if available. In addition, Walker et al. (2004) 4 asserts that the Sphinx-4 system provides additional tools and utilities that contain helper classes for computing recognition statistics such as Word Error Rate (WER), phoneme error rates (PER) etc.

3.2.2 Kaldi

CMU Sphinx provides an object-oriented approach to speech recognition. Kaldi Povey et al. (2011b) on the other hand is a highly modularised library written in C++. Kaldi is based on weighted finite state transducers (WFSTs) used for inference graphs and decoding. The Kaldi WFSTs utilises OpenFst, an open source library, at its core. Together with a collection of configuration scripts for building complete recognition systems, Kaldi supports modeling of a variety of speech model variations with vast support for linear and affine transforms of speech features of arbitrary phonetic-context sizes. Kaldi is specifically suited for acoustic modeling with subspace Gaussian Mixture Models (SGMM) in addition to the standard Gaussian Mixture Models (GMMs).

Architecture

The component architecture of Kaldi is illustrated in the figure below. Modules can be divided into those that utilise the linear algebra libraries and those that use OpenFST. The decodable class forms the link between these two scopes. The rest of the modules lower down the hierarchy are based on modules higher up hierarchy according to this divide.

Feature Extraction

Kaldi supports various speech feature outputs including the standard Mel Frequency Cepstral Coefficients (MFCC), Perceptual Linear Prediction (PLP), Vocal Tract

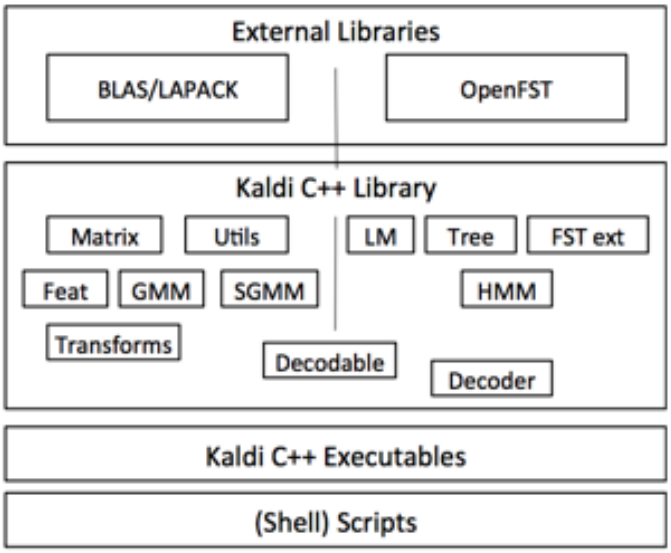

Figure 3.2: Kaldi Architecture(Povey et al., 2011b)

Length Normalisation (VTLN), Cepstral Mean Variance Normalisation (CMVN), Linear Discriminant Analysis (LDA), Semi-Tied Co-variance matrix (STC),Maximum Likelihood Linear Transformation (MLLT), Heteroscedastic Linear Discriminant Analysis (HLDA). These systems are made complete with various configuration parameters for fine tuning their individual models.

Acoustic Modelling

Full co-variance as well as diagonal co-variance GMM modelling is implemented in Kaldi. Efficient log-likelihoods are computed using simple dot products of mean times in-variance and in-variance co-variance. The `DiagGmm` class is responsible for storing co-variances of Gaussian densities. The Acoustic Modelling (AM) class represented by the `AMDiagGmm` class comprise a set of `DiagGmm` objects. These objects which represent Gaussian Mixture Models (GMMs) are in turn represented Probability Density Function (PDF) indices which are then mapped to Hidden Markov Model (HMM) states. There are classes to represent HMM topology as well as the overarching topology representing transition modelling. These two sets of classes provide information required for developing decoding graphs. Rather than using the conventional approach for HMM modelling using hand-made decision tree for left and right phones in a mono-phone model, tree-clustering algorithms automatically generate the decision tree.

Language Modelling and Decoding Graphs

Using the Finite-State Transducer (FST) back in addition to third party language modelling software, Kaldi is able infer sentence estimations using n-gram language models. During decoding, transition-ids are created and attached to corresponding pdf-IDs as a result of tied-state nature of phones where different phones are allowed to have share the same distribution. The transition-id therefore encapsulates the shared pdf-ID and the arc (transition) of phone-specific topology. This way transitions are fine-grained without adding complexity to the decoding graph

Core decoding algorithms are implemented using C++ classes one per decoder. Decoders implement an interface which accepts an acoustic model score for a particular input-symbol and frame. While single-pass decoding is achieved through C++ classes, multi-pass decoding is realised using the supporting configuration scripts.

3.2.3 Mozilla DeepSpeech

The DeepSpeech speech-to-text engine is an ASR speech model and model generator built by Mozilla is based on Baidu's Deep Speech research paper (Hannun et al., 2014a). The system comes in two forms; an installable speech-to-text engine based on the English language and the model trainer. These components were created and run effectively on Unix based systems and to a limited extent on Microsoft Windows systems. Various options for installing the speech to text engine includes either command line based or as an application programming interface (API) using python or NodeJS. In addition, the speech-to-text (STT) engine API also supports bindings for the Rust language, GoLang and GStreamer. This thesis however, did not rely on the STT engine nor API, but rather on the model trainer which was adapted in this research for scattering transform feature-based end-to-end speech recognition.

Runtime library dependencies of both the STT engine and the model trainer include libsox, 2 for sound processing of audio; libstdc++6, libgomp1 and libpthread are used to compile the Connectionist Temporal Classification (CTC) decoder implementation which incorporates the KenLM trained language model Heafield et al. (2013).

Graphics Processor Unit (GPU)-Enabled Speech Model Training

The model trainer of the Mozilla DeepSpeech platform is facilitated by the ability to train models on a highly parallel processing Graphics Processing Unit (GPU). This enables model training-time speed-ups over traditional CPU machines. The Mozilla DeepSpeech platform recommends Nvidia Graphics 10 series processor with a system requirement of 8GB of Random Access Memory (RAM). In section 3.2.5 we introduce TensorFlow python library. Mozilla DeepSpeech platform is able to utilise the GPU using the Nvidia GPU library, CUDA. This is achieved through the python TensorFlow library created by Google as discussed in section 3.2.5.

Common Voice training

The speech corpus used for training in this research was obtained from the Mozilla Common Voice Initiative speech corpora. This consists of over 250 hours of speech data that is subdivided into test, development and training data sets. In addition, the data was subdivided into clean data, that is, clean audio recording with accurate translation and a small subset containing skewed data, that is, audio recording which was either noisy or lacking accurate transcriptions. The skewed data subset consisted 15-25% of the training corpus and was incorporated so that the neural network speech model could simulate and learn real world noisy audio speech-to-text translation. The Mozilla DeepSpeech model trainer provided bash scripts for importing the Common Voice speech corpora as well as converting the files into the appropriate formats and provision of mapping files for the model trainer.

Mozilla DeepSpeech model parameters

The model trainer consists of a root python script "DeepSpeech.py" with various calls to other python scripts responsible for things like audio processing, distributed training, GPU configuration, training coordination. Other accessory bash scripts also present are responsible for downloading and training for different kinds of speech corpora including Mozilla Common Voice(moz, 2019a) and the Wall Street Journal (WSJ)(Paul and Baker, 1992). These sets of scripts are referred to as speech corpus importers.

In order to supply the model trainer with a set of hyper parameters for tuning various aspects of the Mozilla DeepSpeech platform, the following categories arguments passed to the root script ensue:

- Geometry - Defines the number of neurons in the hidden layers of the neural network.
- Cluster configuration - Parameters responsible for distributed training of the speech model across various nodes.
- Global constants - These include all other parameters to gain fine control of the training process. These parameters include how much of the training corpus will be used and which subset should be included; early stopping for pre-trained models that have already been trained to saturation, that is to a stopping condition; the dropout rate for neural network regularisation. This is a strategy to overcome over-fitting where instead of learning inference features the data, the neural network memorizes the training data.
- Global constants - These include all other parameters to gain fine control of the training process. These parameters include how much of the training corpus will be used and which subset should be included; early stopping for pre-trained models that have already been trained to saturation, that is to a stopping condition; the dropout rate for neural network regularisation. This is a strategy to overcome over-fitting where instead of learning inference features the data, the neural network memorizes the training data.
- Adam optimiser - parameters for the Adam optimiser
- Batching - set the number of batches during training.
- Weight Initialisation - standard deviation coefficients for initialising weights.
- Checkpointing - this includes the number of seconds before saving the current model parameter values to the disk. This enables resumption of training in instances where the training was interrupted. For training to resume successfully, the resuming training geometry parameter must be exactly the same as the interrupted geometry training parameter.

- Exporting - Includes parameters for saving a saturated model for inference.
- Reporting - Includes options for setting the log-level however reports are only sent to the standard console output.
- Decoder - These parameters include the path to the alphabet symbols and that of the custom CTC decoder used during decoding of the neural network output.
- Inference - It is possible to use a model trainer to either perform a one-shot inference or resume training from an already exported model. The parameters used for inference are responsible for performing these stated tasks.

In addition to the above configuration there are other accessory scripts that can be used for TensorFlow specific tasks such as conversion of the output model graph to several exportable formats.

3.2.4 Matlab and ScatNet toolbox

In this research, feature processing of audio files to obtain their deep scattering transforms was achieved using a MATLAB toolbox known as ScatNet (Andén et al., 2014). The ScatNet toolbox in general analyses time-series sampled analog signals and has been used successfully for music genre classification, texture and image classification (Andén and Mallat, 2011, Sifre and Mallat, 2013, 2014). In particular, the scattering transforms produced are signal processing layers of increasing width where each layer constitutes the convolution of a linear filter bank wavelet operator (Wop) with a non linear complex modulus.

$$||\text{complex signal}| \star \mathbf{Wop}| \star (\text{low-pass filter}) \tag{3.1}$$

It is the scattering transforms of the audio files that were fed into the DeepSpeech model trainer discussed in Section 3.2.3. The architecture of a scattering networks resembles a deep convolutional network in the sense that each subsequent layer is a mapping of all possible paths from the previous layer.

ScatNet provides default options for most of the parameters that require tuning in order to derive the scattering coefficients for an input signal. In particular, for

audio signals, the most important hyper-parameters set by the library is the number of scattering layers that captures the entire audio spectrum which is set at 2. In addition to this default, the only other parameter to set is the window period of the signal to be analysed per time. A suitable value for the window can be derived from the sampling rate of the input signal. The toolbox function `S=scat(x,Wop)` takes a an input signal, `x`, and an array of linear wavelet operators, `Wop`, in order to compute the scattering coefficients of the input signal. The resulting network, `S` is a cell array whose length $M + 1$ is equivalent to that of the linear filter operator.

Wavelet Factories

By providing optimal defaults for linear operators, ScatNet provides wavelet factories especially suited for efficient signal processing of images and sounds. Therefore, linear wavelet operators are built in a single command function through built-in "factories", which perform wavelet analysis tasks.

Further, the maximum number of wavelets J is automatically derived from the sampling from the sampling period T. The filter banks are formed by dilating the mother wavelet (ψ) by the dyadic factor ($2^{1/Q}$). In the Fourier domain this is expressed as

$$\hat{\psi}_j(\omega) = \hat{\psi}(2^{j/Q}\omega) \tag{3.2}$$

For audio application, to ensure optimized frequency coverage without frequency-redundancy or overlapping, the mother wavelet (ψ) is chosen so that ($Q_1 = 8$) and ($Q_2 = 1$) by default. This also means that the first order filter will be of a higher frequency resolution when compared to the second order filter.

Filter banks

In order to visualise the filters being used by the wavelet operations and referring to Sections (5.5 and 7.1) where it is shown that the first and second order scattering coefficients are respectively defined by the following forms

$$S_1 x(t, j_1) = |x \star \psi_{j_1}| \star \phi(t) \tag{3.3}$$

$$S_2x(t, j_1, j_2) = ||x \star \psi_{j_1}| \star \psi_{j_2}| \star \phi(t) \tag{3.4}$$

where (ψ_j) are band-pass filters and (ϕ) is a low-pass filter.

Furthermore, the wavelet transform operators(Wop) created by the wavelet_factory_1d function are only function handles and do not have any data in themselves. A second return value may be retrieved from the wavelet factory which contains the set of filters returned as a cell array by the wavelet factory.

```
[Wop,filters] = wavelet_factory_1d(N, filt_opt);
```

The filters return argument has a similar structure to the scattering network where each element in the cell array corresponds to the layer order in the scatter network hierarchy. Moreover, similar to the the scatter network, the filters cell array hierarchy has $M+1$ elements, where only M elements are utilised and no filters exist at $M = 0$. The non-zero coefficients of the band pass filters expressed in the Fourier domain, are held in `filtersm.psi.filter` field. When plotting these filters, they are first padded with zeros to the length of `N` which is the entire spectrum. Below is the sample plot made against default filters obtained by the `wavelet_factory_1d` filter.

The script below calculates filter banks at orders $(M = 1)$ and $(M = 2)$. The resulting plot is displayed in Figure 3.3.

```
figure;
for m = 1:2
    subplot(1,2,m);
    hold on;
    for k = 1:length(filters{m}.psi.filter)
        plot(realize_filter(filters{m}.psi.filter{k}, N));
    end
    hold off;
    ylim([0 1.5]);
    xlim([1 5*N/8]);
end
```

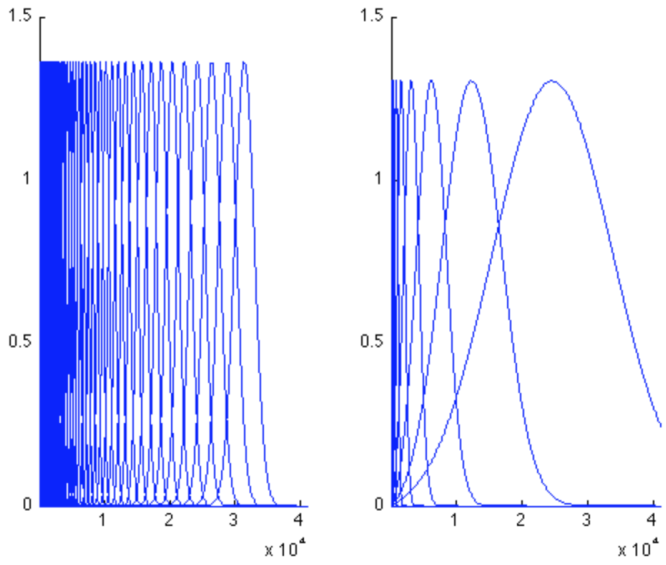

Figure 3.3: Scatter transform wavelet filters

Using the Matlab API

In the previous sections it was seen how the ScatNet toolbox calculates scatter coefficients based on wavelet theory. In this section, the scattering spectrum of an audio signal is implemented using only three command calls to ScatNet library.

The three steps taken in this section are as follows. First load the audio file and set the properties of the audio file required for audio processing of the signal within ScatNet. Note that to do this the sampling rate of the input signal is required. Here, a clip of Handel's Messiah, implemented in Matlab as a function is loaded into a "y" variable by default with the "load handel" command.

Given that the sampling rate of the loaded clip is , the parameters set are

i. N - the number of samples in the signal, and

ii. T - the window size. Here, T is set to 4096 which corresponds to about half a second.

```
load handel; % loads the signal into y
N = length(y);
T = 2^12;    % length of the averaging window
```

The second step is to create the filter operators for which the type of filter signal length, and the window length are passed in as parameters. It has been shown in

the preceding sections that two layers are sufficient to capture energy contained in an audio signal and by default the quality factors of the two layers are ($Q_1 = 8$) and ($Q_2 = 1$). These default_filter_options are automatically integrated with the 'audio', filter type option.

```
filt_opt = default_filter_options('audio', T);
Wop = wavelet_factory_1d(N, filt_opt);
```

Note that the wavelet_factory_ functions is an intensive operations because many filters are being built at once in batch processing of signals discussed in Section 3.2.4, we therefore only perform this as part of the initialisation and the returned wavelet operators (Wop) can be reused without having to recreate them.

Having all the parameters required, in the third and final step , call the scattering transform of y generic function scat, to derive the scatter coefficients.

```
S = scat(y, Wop);
```

Scatter Feature Enhancements and Batch Processing

Having obtained the scatter coefficients, further feature enhancement is achieved by taking the log of the normalised coefficients. This can be visualised using the built-in scattergram function which produces a translation-invariant, second-order, spectrogram-like visualization of the scattering transform a one-dimensional audio signal.

In the code snippet below, `j1` is the second-order coefficients arbitrarily chosen to equal `23`. The first parameter to scattergram are the first-order coefficients and the second wildcard [] parameters gathers all paths from the first order.

```
j1 = 23;
scattergram(S{2},[],S{3},j1);
```

The following functions in the code snippet below are applied to realise the log of the normalised scattergram.

```
S = renorm_scat(S);
S = log_scat(S);
scattergram(S{2},[],S{3},j1);
```

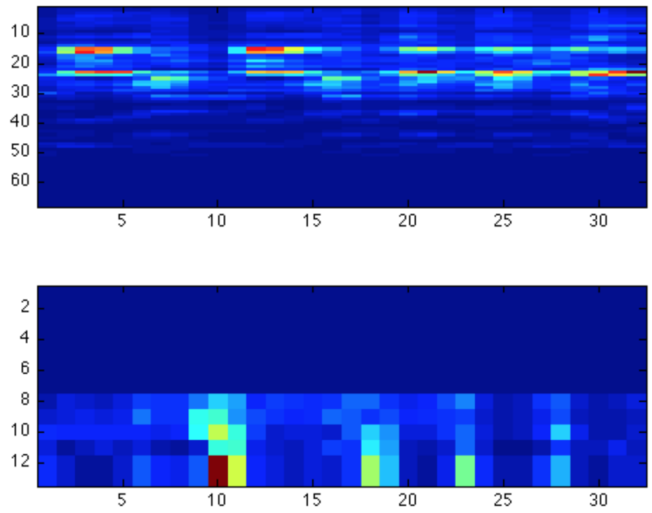

Figure 3.4: Unnormalised scattergram

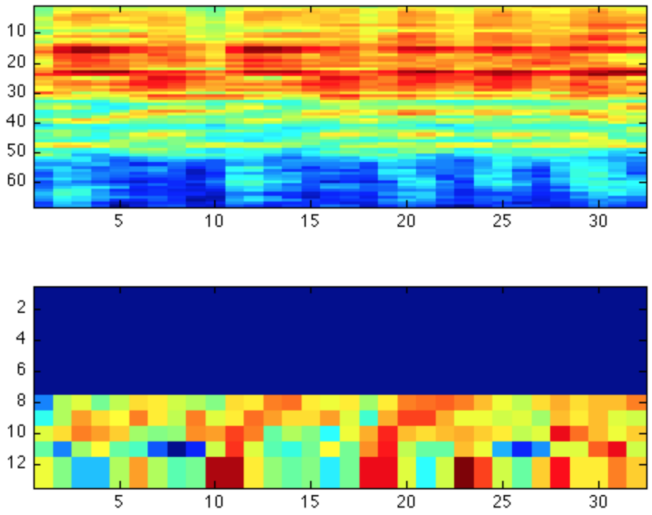

Figure 3.5: Log normalised scattergram

With the corresponding scattergram illustrated in figure 3.5.

Finally, to utilise the scattering coefficients as features for classification tasks, we extract the vector form using `format_scat` function.

```
[S_table, meta] = format_scat(S);
```

The `S_table` is a P-by-N table, where `P` is the flattened total of all the scattering combined coefficients within each layer of the Deep Scattering Network (DSN) and `N` is the number of time points. The network is now feature ready for classification tasks using an affine space classifier.

For batch processing ScatNet provides a database feature which can accept a collection of input vectors rather than a single input signal. The following commands show ScatNet commands for performing batch processing on the GTZAN dataset used for musical genre classification

First, specify the path to the audio target.

```
src = gtzan_src('/path/to/dataset');
```

Next all the defaults for ScatNet analysis and processing are set as explained in the previous Sections 3.2.4,3.2.4 and 3.2.4 above.

```
N = 5*2^17;
T = 8192;
filt_opt.Q = [8 1];
filt_opt.J = T_to_J(T, filt_opt);
scat_opt.M = 2;
Wop = wavelet_factory_1d(N, filt_opt, scat_opt);
feature_fun = @(x)(format_scat( ...
log_scat(renorm_scat(scat(x, Wop)))));
```

It is possible to optimise the training by sub-sampling each signal. The feature_sampling option is used to specify sub-sampling.

```
database_options.feature_sampling = 8;
```

Finally, a call is made to `prepare_database` function to compute all the scatter network features of the `src` database.

```
database = prepare_database(src, feature_fun, database_options);
```

In this research, further speed up was achieved by utilising Matlab's parallel processing on the for loop (see Appendix III) thus bypassing the batch processing utility of ScatNet.

3.2.5 TensorFlow

TensorFlow is a state-of-the-art high performance library by Google for Deep learning. Deep learning is a branch of artificial intelligence which acquires learning from deep neural network architectures. The paragraphs and subsections that follow under this topic give an overview of the TensorFlow library as outlined by the following authors Abadi et al. (2016), Goldsborough (2016) and Abadi et al. (2017).

Deep learning has significantly advanced in various application domains and by far out-performed traditional approaches. TensorFlow offers researchers and enthusiasts an open source software library for use in defining, training and deploying deep learning models.

TensorFlow works by defining data flow graphs with mutable state. A data flow graph represent complex functional node and edge architectures, where each node represents an operator instance applied to input values which constitutes the edges. The operators are implemented by abstract kernels for particular types of interchangeable devices (such as CPUs and GPUs)(Abadi et al., 2017).

There are three main concepts at TensorFlow's core. These concepts are tensors, operations and mutability. Tensors are arrays of arbitrary dimensions where the underlying data type is either specified or inferred at graph-construction time. Operations process data and constitute nodes within the compute graph. Basic operations invariably are mathematical functions such as vector dot products. However, some of the operations indeed may be associated with a read or state update. Such tensor which permit run-time updates in TensorFlow are referred to as variables. Finally, there may be edges for communicating and constrain the order of execution. These structures invariably affect the observable graph semantics and may also affect the computation performance.

Once a TensorFlow program constructs a graph using a client interface such the Python API, the TensorFlow program can send messages to the graph, by "feeding"

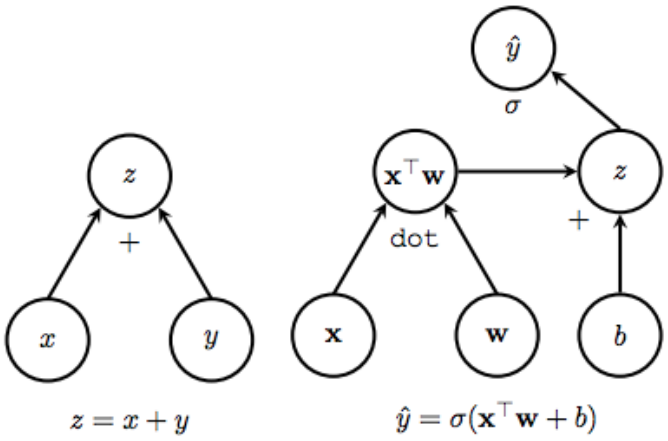

Figure 3.6: Sample TensorFlow computation graphs(Goldsborough, 2016)

it inputs and "fetching" outputs from it. TensorFlow then propagates the input values through the execution graph performing operations called by the client code, until all nodes instructed to run returned with their outputs.

Data dependencies and control edges, dictate the order of execution. Often, a graph is executed severally and tensors declared as placeholders or constants are used once. However, variable tensors have mutable state which allow persistence across multiple executions. The parameters of the model stored in variables are usually updated as part of running the graph.

Programming Model

In this section examples of execution data flow graphs are given; and in the following sections we highlight the other major special features of TensorFlow including automatic differentiation and back-prop algorithm implementation, control flow, check pointing, programming interface, sample implementation and graph visualisation.

In a TensorFlow computational data flow graph, vertices or nodes of the directed graph represent operations, while edges signify flow of data between these vertices or operations. Thus labels on nodes are representative of the actions taken at that node. Similarly, labels representing values flow in the direction of the edges. The inputs to a labeled operation are therefore the labels which have edges directed towards the operation. A computation or data flow graph is illustrated in Figure 3.6.

The left graph displays a basic compute graph consisting of an addition operator

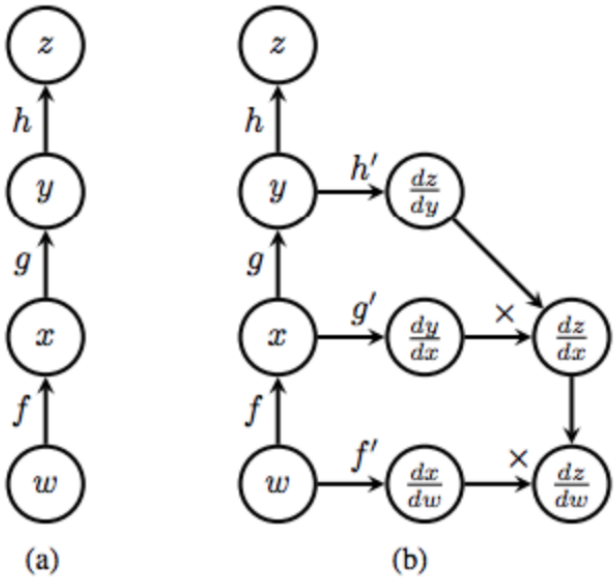

Figure 3.7: Tensorflow graph with backprop nodes (Goldsborough, 2016)

having two input variables x and y. The result, z is the output of the + operation. The right graph gives an example logistic regression function. $\hat{y}$ is the final output of the function for some sample vector $\mathbf{x}$, weight vector $\mathbf{w}$ and scalar bias b. As shown in the graph, $\hat{y}$ is the output of the sigmoid or logistic function σ

Backprop nodes

The Backprop algorithm Goodfellow et al. (2016) is an efficient method to compute error values for weights within a multilayer or deep neural network. The algorithm is summarised as follows. Assuming a neural network with two hidden layers. The two layers within the network respectively have output functions $f(x; w)$ and $g(x; w)$ such that $f(x; w) = f_x(w)$ and $g(x; w) = g_x(w)$. Where x is the input from the previous layer or from the input layer and are the weights. The error function , is an implicit function of all the previous layers. In the case of the 2-layer network $e = (f_x \circ g_x)(w) = f_x(g_x(w))$. The back prop algorithm uses the chain rule to correctly assign appropriate updates to each weight at every layer within the network. The updates which are the gradient or the error function with respect to the weights are de_x/dw. The backprop algorithm therefore uses the formula $[f_x(g_x(w))]' = f'_x(g_x(w)) \cdot g'x(w)$ as it traverses the graph in reverse to compute the updates.

Figure 3.7 illustrates how tensor implements backprop by adding two extra nodes at the appropriate layers within the network to satisfy the chain rule. For each

operation encountered, there is a corresponding gradient function that reverses the output as a function of the inputs. The output of this gradient function can then be propagated backwards to a previous operation which would represent a previous layer within a neural network. The gradient function propagated to the previous layer is then used to complete the parameters of the chain rule by multiplying with that previous layer's gradient. This output is ready to be propagated down the network to the subsequent layer to perform a similar function. This process continues until all the weights within the network have appropriate update values.

Control flow

TensorFlow also supports control-flow operations. For this reason TensorFlow is not a directed acyclic graph (DAG) but can support cyclic structures. If the number of loops required by the computation graph is known at graph construction. It is easy to maintain a DAG structure simply by unrolling the number of loops specified. However, this is not always the case. There are instances in which a variable number of loops is required at runtime. Hence, the computation graph becomes increasingly complex. This is particularly the case for back gradient descent and back propagation of errors (see section 3.2.5 for a walk through). The process of stepping back through a loop in reverse to compute gradients is known as back-propagation through time (Al-Rfou et al., 2016).

Checkpoints

One can add Save a node to a compute graph, connecting them to variables whose tensors can then be serialized. At another instance one may connect the same variable to a Restore operation. This operation deserializes the stored tensor at another point within the execution graph. This is especially useful over long periods of training to keep track of the model's variable parameters. These elements form part of distributed TensorFlow's fault tolerance ecosystem.

Programming Interface

TensorFlow implementation provides two developer interfaces which include the Python interface and the C++ interface. While the python interface offers a rich

feature set for creation and execution of computation graphs, the C++ interface is primarily a back end implementation with a much more limited API primarily used for executing graphs built with Python and serialised to Google's protocol buffer.

It is worth noting that unlike PyTorch (Ketkar, 2017), the Python API handshakes very well with NumPy(Oliphant, 2006–) numeric and scientific open source programming library. As such, TensorFlow tensors can be naturally substituted with NumPy ndarrays without any need for type-conversion seen in PyTorch tensors.

Tensorflow client model walk through

In this section, a sample client tensorflow model is examined. The model consists of a simple multi-layer perceptron (MLP) with one input and one output layer to classify hand-writtin digits in the MNIST(Krizhevsky et al., 2012) dataset. In this dataset, the examples are small images 28×28 pixels depicting handwritten digits from 0 to 9. The examples form a matrix having the shape $\mathbf{X} \in R^{n \times 784}$ where represents the number of images, and 784 represents the flattened 28 x 28 pixel image. The client code performs an affine transform operation, $\mathbf{X} \cdot \mathbf{W} + \mathbf{b}$, where $\mathbf{W}$ is the matrix of weights $\in R^{784 \times 10}$, and $\mathbf{b}$ is a vector of biases $\in R^{10}$. The result of the affine transform operator is the matrix $\mathbf{Y} \in R^{n \times 10}$. The resulting non-probabilistic logits gives an unnormalised distribution of digits.In order to obtain the valid probability distribution $Pr[x = i]$ where x-th example is classified as the digit i, the soft-max method is utilised.

$$softmax(\mathbf{x})_i = \frac{\exp(\mathbf{x}_i)}{\Sigma_j \exp(\mathbf{x}_j)} \tag{3.5}$$

Error loss values are then computed using an objective function and the model's current training parameters $\mathbf{W}$ and . This is obtained from the cross entropy calculation given by

$$H(\mathbf{L}, \mathbf{Y})_i = \Sigma_j \mathbf{L}_{i,j} \cdot \log(\mathbf{Y}_{i,j}) \tag{3.6}$$

Where $\mathbf{Y} = softmax(\mathbf{x})$ and $\mathbf{L}$ are the correct one-hot-encoded labels. More precisely, the batch-mean loss over all inputs $\mathbf{x}$.

Next, the Stochastic Gradient Descent (SGD) is run to update the weights of our model. A TensorFlow class is provided and will be initialised with a learning rate. The minimise function of this class takes the loss tensor as parameter used for

minimisation.

The operations run repeatedly within a `tf.Session` context manager. Refer to Appendix IV for the complete code listing.

Visualisation

TensorFlow interface offers the option of visualising computation graphs. Complex topologies consisting of various sub-layers can be presented in a lucid form, offering the user a congruent, organised picture of exactly how data is consumed in a compute graph. Sub-graphs may be grouped into visual blocks and referred to in name scopes. For example a single neural network layer may take up such a named scope. The name scopes are then interactively expanded on to give the detailed group visualisation.

Two types of metrics are obtainable from the TensorBoard. These are summary operations, when attached as nodes in the graph, permit the user to monitor individual tensor values over time. The first is the scalar summaries which capture tensor values and can be sampled at certain points within training epochs. One can now, for example, observe the trend of the accuracy loss of the training model over time.

The other summary operation offers the user the ability to track distributions, such as final soft-max densities or the distribution of neural network weights.

Lastly, sample images can be visualised on the TensorBoard graph. This way kernel filters of a convolutional neural network can also be visualised. In addition to all of these, one can perform zooming and panning actions directly on TensorBoard's web interface including expansion and collapsing of individual name scopes

3.2.6 Choregraphe

The Choregraphe software tool is a high-level language used for programming of Nao humanoid robots. This is built on top of the Naoqi/Gentoo Unix/Robot Operating System(ROS) **??**. Speech recognition and processing modules of the Choregraphe tool were explored and expanded at the initial stages of the research. However the Choregraphe software tool for the Nao robot was found to be unsuitable in speech recognition at the level of research that aligned with the research objectives and

therefore was not utilised in this work.

3.2.7 Alisa

Alisa tool is a lightly supervised sentence segmentation tool based on Voice Activity Detection (VAD) algorithms (Stan et al., 2016). It is so called lightly supervised because it requires small amounts of training data. Generally the tool was asserted to be optimised for sentence segmentation and offered assistance in the creation of new speech corpora in a language-independent fashion.

The Alisa tool researchers deploy a two-step method for aligning speech, and claim performance up to 70% imperfect transcriptions often found in online resources can be successfully aligned with a word error rate of less than 0.5%. This tool is therefore said to be suitable for development multilingual and under-resourced language aligned speech-corpora.

The motivation behind Alisa was to reduce the time and effort used to gather large amounts of quality data as well as actively eliminate the domain knowledge required to phonetically transcribe speech data. In addition, and as a bonus to achieving the first objective, is the ability to migrate speech technology fairly seamlessly from one language to another and therefore realise the rather tedious task of automatic transcription of a new language.

Alisa Architecture

The goal of automatic transcription of new language with low resource constraint is particularly valuable to this research and as such, it would be relevant to review the enhancements introduced to Alisa. The two step-method consists of a GMM-based sentence level segmenter and also an iterative grapheme acoustic model used for alignment. The sentence level GMM-based speech segmenter is used to automatically segment speech into utterances which as discussed earlier forms the basic unit of processing within any ASR system. This attempts to relieve the researcher off the manual process of segmenting the continuous audio file manually. This process included a GMM-based voice activity detector trained from about 10 minutes of manually labeled data. The second step grapheme based acoustic model is supplemented with a highly restricted word network they referred to as a skip network. To-

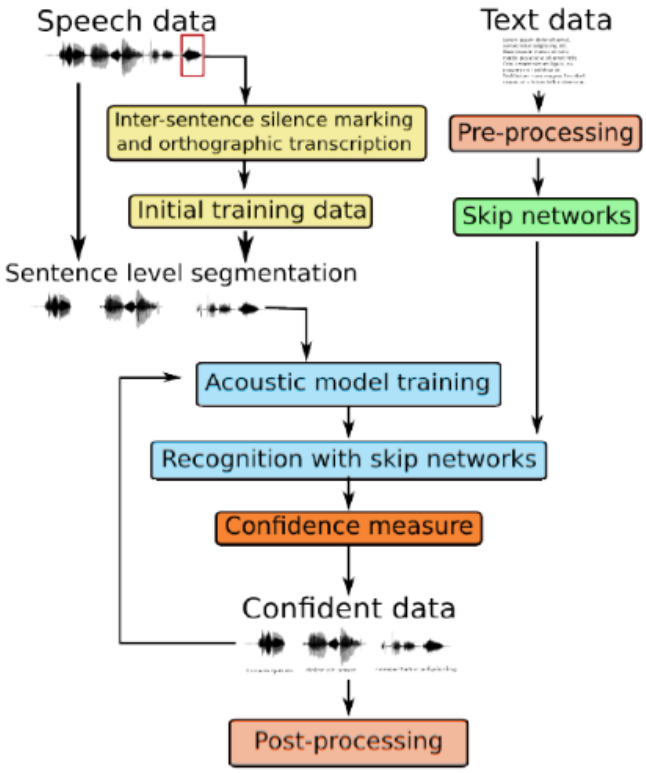

Figure 3.8: Alisa Architecture(Stan et al., 2016)

gether an iterative acoustic modelling training procedure is formulated. The method described required the initial training data and a minimal labelling procedure that involved simple letter to sound rules and inter-sentence silence segments to provide an orthographic transcript of the initial 10 minute recording data. Therefore, this process is resource-effective because non-experts can also provide this data. The actual alignment process made use of a grapheme level Viterbi decoder to drive the iteratively self-trained grapheme models. The model architecture is shown in the figure below.

Figure 3.8 shows a block diagram of the steps involved in the alignment. The method can be applied to any language with an alphabetic writing system, given the availability of speech resource and its corresponding approximate script.

There is an option of using a grapheme based acoustic model. This however increases the margin for error. Several steps were introduced in the Alisa tool to minimise this error margin. The chief being the introduction of a tri-grapheme acoustic model which is modeled after using context dependent triphones in traditional acoustic modelling. Other techniques deployed to crash the error margin

include the use of discriminative training with the Maximum Mutual Information (MMI) criterion (Schluter and Ney, 2001) and methods described in (Novotney and Schwartz, 2009). It was observed that Alisa provided good alignment but was not fully featured. For instance it had no way of adding insertions and substitutions in the audio data not provided in the transcription. Finally, Alisa was found to be restricted to only languages that can utilise the English alphabet.

3.3 Initial Experiments

The experiments in the following sections describe initial experiments based on the initial study of a language learning companion before the research was narrowed down to a low resource speech recognition. These preliminary experiments in addition to a preliminary Language Learning Survey helped to narrow down the Research to the specific speech processing task of Low Resource Automatic Speech Recognition (LR-ASR).

The following sections describe analysis of raw wave-forms using auto-correlation signal processing in Matlab and experiments made with the Nao robot speech processing engine and experiments with speech recognition toolkit and speech processing tasks. These tasks include digit recognition systems using CMUSphinx and Kaldi speech recognition toolkits and speech alignment tasks using Alisa tool.

3.3.1 Auto-correlation Experiments

Preliminary experiments were carried out on raw speech signals in an attempt to quickly segment individual phonemes based on a basic threshold algorithm. Further experiments designed an auto-correlation algorithm to attempt to discover a phoneme alphabet in a particular data set in a semi-supervised fashion.

This method had the goal of simulating posterior distributions of phonemes from auto-correlation estimate. This presents an unnormalised posterior distribution measurement of phoneme segments over the entire signal.

The correlation theory is based on the idea that when a signals is superimposed on itself in a time-shifted manner, the convolution over itself is highest when the two signals have zero time lag that is, perfectly overlapped in sync and the better

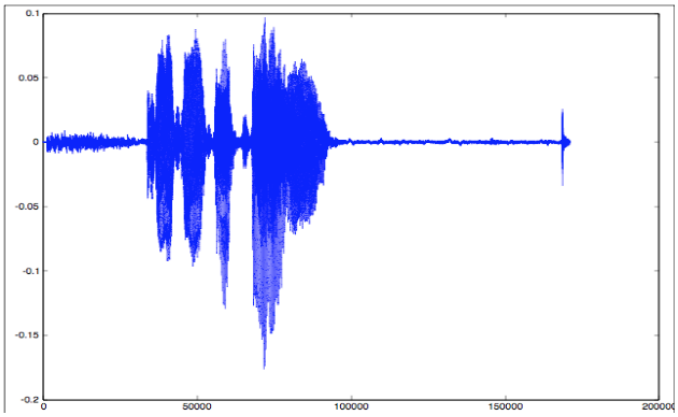

Figure 3.9: Original waveform input for auto-correlation

the overlapping the higher the value of the correlation and the lesser the signals are matched they tend to cancel out each other and hence a very low value of the correlation. The normalised auto-correlation value is obtained in Picone (1996) from a signal $x(n)$ in the following equation:

$$\Psi(i) = \frac{\sum_{n=0}^{N-1} x(n)x(n-i)}{\left(\sum_{n=0}^{N-1} x(n)^2\right)\left(\sum_{n=0}^{N-1} x(n-i)^2\right)} \tag{3.7}$$

Based on experimental procedure, estimated locations of similar wave-forms representing segmented phonemes are calculated. Although the procedure is subject to degrade in the face of most of the difficulties associated with dealing with raw audio waveform, it further emphasises the need for accurate speech features and pre-processing highlighted in the previous chapter.

This two stage procedure performs segmentation of phonemes and then discovery of phoneme clusters using a statistical auto-correlation algorithm. The process is described in the following sections.

Segmentation

Figure 3.10 describes the various steps of the segmentation phase while Figure 3.9 shows the original audio file. At the segmentation phase, we first of all adjust the scale of the original raw audio file to have only positive values rather than having it centred on zero (Figure 3.10a). At the next step a smoothing filter based on experimentation is used to perform both smoothing as well as determining the peaks and trough (Figure 3.10b). Then a threshold is applied to segment the waveform

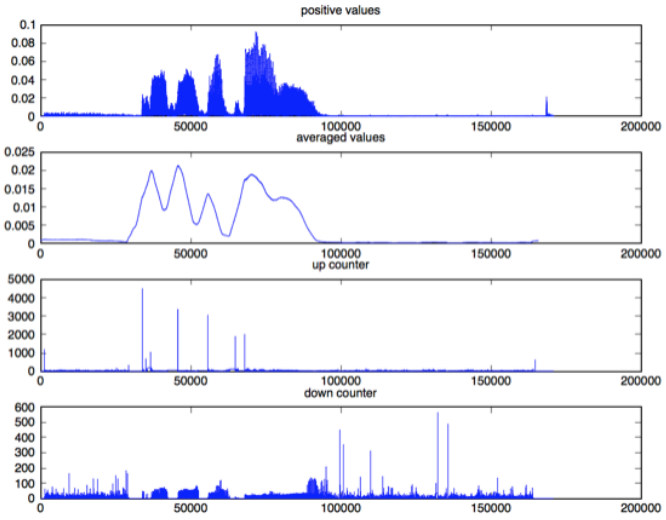

Figure 3.10: Original waveform input for auto-correlation

based on discovered inflection points (Figure 3.10c).

Auto-correlation

At the auto-correlation stage estimated phoneme segment boundaries are stored in an array and cross-correlated with the original signal. Even though at a top-level view, the entire signal is auto-correlated, at the individual segment level, the signals are cross correlated against one another. Furthermore, to achieve a 'fair' correlation estimate, individual segments representing estimated phonemes need to be re-sampled to eliminate mismatching of contour representations of the individual phonemes.

The proposed auto-correlation algorithm performs both top-down and bottom-top processing. In the first stage it does bottom-top segmentation, while in the second phase top-bottom auto-correlation. The major weakness is this auto-correlation method the segmentation algorithm, data filtering and the feature representation. The Bayesian method of segmentation (Kamper et al., 2016) which is related to this method also improved on these weaknesses was able to improve on these weaknesses by using ASR feature preprocessing and a combination of acoustic embedding Dynamic Time Warping (DTW) for clustering rather than auto-correlation. In effect using the extracted features for clustering is in theory a better speech estimate with less intrinsic noise for classification than using an only smoothed audio data.

3.3.2 Experiments with Nao robot

Nao is a humanoid robot developed mainly for deployment in environments for robotics education and development purposes. Nao comes with a speech recognition software that offers features such as language settings and recognition sensitivity. However it was understandably found to be limited because the Nao robot itself does not possess the processing power to perform CPU intensive training of acoustic models. The Nao robot did however offer a level of support for using the pocketsphinx system. The pocketsphinx system is the C-language equivalent of CMUSphinx speech recognizer system also by Carnegie Mellon University. Using the pocketsphinx method, acoustic models trained high performance systems can then be deployed to Nao for fast decoding within the Nao.

3.3.3 Digit Speech Recognition and Alignment Experiments

These experiments were performed using CMU Sphinx4 recognition system and Kaldi speech recognition software. While CMU Sphinx and pocketsphinx delivered standard interface for speech recognition using generative hybrid models, Kaldi speech in addition also offered advanced methods such as subspace Gaussian mixture model used to develop cross-lingual acoustic models and deep architectures for hybrid generative-discriminative models for speech recognition. The main challenge with Kaldi was that it was CPU intensive and required a reasonable amount of parallel processing to achieve good results within a reasonable time period.

Speech alignment experiments were performed using the Alisa Stan et al. (2016) tool which is a python based tool with calls made to the HMM toolkit Young et al. (2002). The Alisa tool alignment process undergoes a semi-supervised process and requires an error prone time-intensive manual pre-alignment procedure. The tool itself was found to be quite unstable and the output results were not very easily reproducible for further tests to be carried out on different data sets. In addition, the time-intensive pre-alignment procedure made the tool not very useful for this research. Had the tool been more successful, the tool, which utilises Voice Activity Detection (VAD) algorithms, would have been especially useful for sentence segmentation of long sequences of transcribed audio speech. This tool however still lacked in alignment at either a word-level or sub-word level of alignment required in ASR

pipelines.

3.4 End-to-end Research Experiments

A significant issue arises when using HMM-based toolkits such as Kaldi. This is the requirement for aligned speech. In more recent endeavours, there have been efforts towards automatic alignment of transcribed audio speech recordings through successive Baum-Welch estimation techniques Gales et al. (2014), Ragni and Gales (2018), Ragni et al. (2014). However, this technique is not particularly compatible with end-to-end goals adopted for this research as it would require preprocessing and successive pre-training of the data set.

The following section describes the post-alignment experiments and in a later Chapter, how these methods deal with the problem of automatic speech alignment in a fashion which was compatible with end-to-end speech processing. The end-to-end requirements were desirable for low-resource speech recognition as it introduces a simpler speech model design. The downside however to the end-to-end approach is the dependency on very deep recurrent neural network structures which require large volumes of data for successful training.

3.4.1 Tensor flow sequence-to-sequence character-to-diacritically-labelled-character model

Experiments performed in this and the next three sections are all based on sequence-to-sequence modelling using recurrent neural networks. While the this section and the next section represent precursor experiments centred around speech recognition tasks, the later two sections represent the final experiments reported in this work.

The character-to-diacritically labelled character model was a sequence-to-sequence diacritically labeled experiment to automatically infer diacritic transcriptions of the Wakirike language given the plain unmarked Wakirike language text as input. This is a task, when achieved successfully, then becomes a sub task towards developing a phonetic dictionary for the Wakirike Language and the phonetic dictionary in turn can be used in HMM speech recognition or equivalent end-to-end models. This experiment was a precursor experiment, the results of which were reserved for further

study.

3.4.2 Sequence-to-sequence Grapheme-to-Phoneme (G2P) model

This follow up experiment to the previous experiment in section 3.4.1, attempts to automatically generate a phonetic dictionary from graphemes in a text corpus. Grapheme-to-phoneme experiments come in two flavours, the first being a continuation of the previous experiment, that is, using diacritically marked symbols, and the second flavour using non-marked graphemes as input. The experiments we performed used the latter non-marked graphemes as input. As this experiment was also a subtask in HMM speech model building, the results of these experiments were reserved for further study.

What follows in the next three sections are sequence-to-sequence experiments actively developed in this research and are detailed in chapters (6,7 and 8). A brief summary of the experiments are highlighted in the following sections (3.4.3, 3.4.4 and 3.4.5). Note that these models all utilise TensorFlow deep learning library including the Bi-directional speech model (section 3.4.4) which is built on top of Mozilla DeepSpeech with the exception of section 3.4.5 which is based on PyTorch; a similar deep learning library.

3.4.3 GRU language model for Wakirike language based on TensorFlow

The language model developed in this research is a character-based sequence-to-sequence deep recurrent neural network that maps a sequence of characters to a sequence of words found in the training data set. This model met the objective of reducing the vocabulary size required for language models as well as the text corpus required as inferences could be made over the smaller-fixed character vocabulary rather than orders or magnitude larger word corpus with the possibility of out of vocabulary terms found in the training data. Though this may occur in the character sequence-model at the inference stage, it would not normally happen during training. The neural network model developed is described in Chapters 4 and 7, and consists

of Gated Recurrent Unit (GRU) Recurrent Neural Network (RNN). The GRU is a specialised type of Long Short-Term Memory (LSTM) cell RNN. The emphasis here is on the ability to model over particularly long sequences of the training data. In this case, over long character sequences. Thus, the network is able to learn long term dependencies as would be naturally required to construct grammatically correct sentences. In essence, the RNN is able to learn grammar rules inherently from the training data.

3.4.4 Bi-Directional LSTM-based end-to-end speech model

A similar LSTM sequence-to-sequence network based on Baidu Research's original research design (Hannun et al., 2014a) is developed in this research for end-to-end speech recognition. This model, as its name implies, attempts to establish long term relationships by adding a reinforcing LSTM layer learning information but this time from the opposite direction, hence the bi-directional architecture.

In addition, the model incorporates the Connectionist Temporal Classifier (CTC) decoder. This enables the model to make run-time inferences on both the character as well as estimate audio wave to character label alignment simultaneously. This makes this design accommodate end-to-end goals and ultimately simplifies the overall design and completely eliminates the need for either manual or semi-supervised alignments mentioned previously in sections (3.2.7, 3.3.3 and 3.4).

3.4.5 ESP-Net Experiments

The ESP-Net (End-to-end Speech Network) toolkit (Watanabe et al., 2018), is a speech processing toolkit that was of interest to this research because it offers end-to-end capabilities not only in Automatic Speech Recognition (ASR) but also in Text-To-Speech (TTS) or speech synthesis and other speech-sequence-processing related tasks. In addition, the toolkit offers multi-modal training combining both Attention networks Vaswani et al. (2017) with CTC Transformer networks as well as multi-channel feature representation that is, the fusing together of multiple feature representations of an audio signal. Only preliminary experiments were carried out using ESPNet and is discussed in Chapter 8 of this work.

3.5 Method of evaluation

System building methodology (Nunamaker Jr et al., 1990) for speech recognition systems requires models to be evaluated against speech recognition Machine Learning metrics. For language models, perplexity metric was used for evaluation. BiLingual Evaluation Understudy (BLEU)(Papineni et al., 2002) has also been used as a metric for evaluating language models.

Perplexity measures the complexity of a language that the language model is designed to represent (Jelinek, 1976). In practice, the entropy of a language with an N-gram language model $P_N(W)$ is measured from a set of sentences and is defined as

$$H = \sum_{\mathbf{W} \in \Omega} P_N(\mathbf{W}) \tag{3.8}$$

where Ω is a set of sentences of the language. The perplexity, which is interpreted as the average word-branching factor, is defined as

$$PP(W) = 2^H \tag{3.9}$$

where H is the average entropy of the system or the average log probability defined as

$$H = -\frac{1}{N} \sum_{i=1}^{N} [log_2 P(w_1, w_2 \dots w_N)] \tag{3.10}$$

For a bi gram model therefore, equation (3.10) becomes

$$PP(W) = 2^H = 2^{-\frac{1}{N} \sum_{i=1}^{N} [log_2 P(w_1, w_2 \dots w_N)]} \tag{3.11}$$

After simplifying we have

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}} \tag{3.12}$$

Full speech recognition pipelines are usually evaluated against the Word Error Rate (WER). WER is computed as follows:

$$WER = \frac{I + D + R}{WC} \times 100 \tag{3.13}$$

Here I, D, and R are wrong insertions, deletions and replacements respectively and WC is the word count.

Metrics used for low speech recognition in the zero speech challenge (Versteegh et al., 2015) include the ABX metric. Other common speech recognition error metrics following a similar definition as the Word Error Rate (WER) are Character Error Rate (CER), Phoneme Error Rate (PER) and Syllabic Error Rate (SyER) and sentence error rate (SER).

3.6 Chapter Summary

In this chapter we outline how this research set out to achieve its objectives. The main claim of this research is that by building a speech model that combines knowledge of end-to-end processing along with state of the art signal processing, the overall training complexity and build time for new ASR systems can be improved. This research aims to deliver this through by the unique combination of a CTC-based deep recurrent bi-directional neural network with high performance feature processing of Deep Scattering Networks (DSNs).

This chapter also reviews the technologies utilised by this research in order to arrive at the research outputs and briefly describes the experiments performed. Within this space we describe CMUSphinx, Kaldi, Mozilla DeepSpeech, TensorFlow, Matlab and ScatNet as major libraries used. The first two of these are Hidden Markov Model (HMM)-based libraries and the rest are signal processing systems used to build Deep Recurrent Neural Network (RNN) models. Finally, we mention metrics for the evaluation of the models built in this research.

Chapter 4

Background 1: Recurrent Neural Networks in Speech Recognition

The HMM model described in Chapter 2 uses a divide and conquer strategy which has also been described as a generative Machine Learning algorithm in which we use the smaller components' representations as modelled by the HMM to learn the entire speech process. In previous chapters, this was referred to as the bottom-top strategy. The discriminative method however uses the opposite mechanism. Instead of using the building blocks of speech to determine speech parameters of a HMM, the discriminative strategy determines the posterior probability directly using the joint probability distribution of the parameters involved in the discriminative process. The discriminative approach, discussed in this chapter focuses in on Neural network architectures.

4.1 Neural network architecture

The building block of a neural network simulates a combination of two consecutive linear and non-linear operations having many inputs interconnected with the linear portion of the network. This rudimentary structure is described by McCullough and Pitts (1942) and in Cowan (1990) as the Perceptron in Figure 4.1

The linear operation is the sum of the products of the input feature and a weight vector set. This vector sum of products is referred to as an affine transformation or operation. The non linear operation is given by any one of a selection of nonlinear

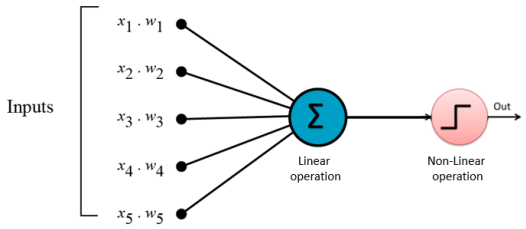

Figure 4.1: Perceptron

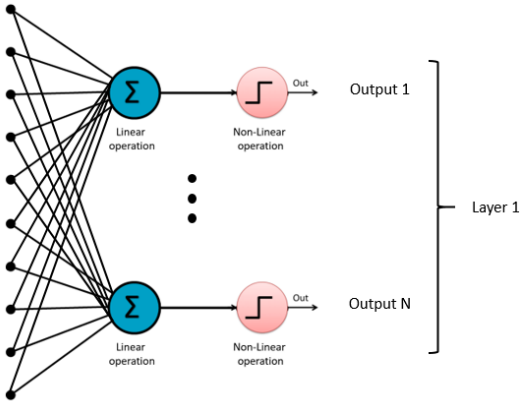

Figure 4.2: Neural network

functions. In Figure 4.2 this is shown as a step function. The step function is activated (becomes 1) whenever the output of the linear function is above a certain threshold, otherwise remains at 0. A simple neural network of perceptrons is formed by stacking the perceptrons into an interconnected layer as shown in the Figure 4.2 :

From the preceding paragraph, each combination of linear operation followed by a non linear operation is called a neuron and the total number of neurons in the layer formed is termed as M-number of neurons in the layer.

4.1.1 Multi-layer Perceptron (MLP)

The multilayer Perceptron or MLP extends the basic Perceptron structure by adding one or more hidden layers. These hidden layers comprise the outputs of one layer becoming the input of the next layer. In the simplest case having one hidden layer, the output of layer 1 becomes the input of the final output layer. In comparison, the Perceptron is a one dimensional structure having one or more linear and non linear combination outputs, while the multilayer Perceptron is a 2-dimensional structure having one or more hidden layers of N linear and non-linear combination outputs.

Mathematically speaking the output of each layer of an MLP having N inputs and M neurons is given by

$$z_j = h(b_j) = \frac{1}{1 + e^{-b_j}} \tag{4.1}$$

is the non-linear function while is the linear function given by:

$$b_j = \sum_{i=0}^{N} w_{ji}^{(1)} \qquad j = 1, 2, \ldots, M \tag{4.2}$$

For each layer in the MLP, the zeroth input value x_0 is 1 indicating a bias term. This bias term is used in the neural network to ensure regularised and expected behaviour of the neural network. In this example the non-linear step function is given by a more complex exponential. In the next section the nonlinear functions for a multilayer Perceptron is derived.

4.1.2 Sigmoid and soft-max Activation Function

The combination of the linear function and the non linear function in the neural network could be said to be transformation of an algebraic problem to a probabilistic function. In this case the "step" function is a squashing sigmoid-shaped function that converts the inputs into a Naive Bayes function evaluating the probability that an output belongs to any of the output classes (C_y) given the data ($\mathbf{x}$).

$$p(C_1|\mathbf{x}) = f(a) = f(\mathbf{w}^\top \mathbf{x} + w_0) \tag{4.3}$$

In a two class problem with classes C_1 and C_2, the posterior probability of class C_1 is expressed using Bayes's theorem

$$p(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(\mathbf{x}|C_2)p(C_2)} \tag{4.4}$$

Dividing through by $p(\mathbf{x}|C_1)p(C_1)$ gives us

$$p(C_1|(x) = \frac{1}{1 + \frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_2)p(C_2)}} \tag{4.5}$$

If we define the ratio of the log posterior probabilities as

$$a = \ln \frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_2)p(C_2)} \tag{4.6}$$

If we substitute back into (4) we have:

$$p(C_1|\mathbf{x}) = f(a) = \frac{1}{1+e^{-a}} \tag{4.7}$$

Here $a = \mathbf{w}^\top \mathbf{x} = w_0$. Thus the activation for the non-linear function is driven by the probability of the data to give the output class. The probabilistic function here is called a sigmoid function due to the s-shaped graph that is plotted by the function.

Rather than using the sigmoid function for multi-class classification a similar soft max function is derived by using the log probability of classes. If $a_k = \ln(p(\mathbf{x}|C_k)p(C_k))$ then:

$$y_k = p(C_k|\mathbf{x}) = \frac{e^{a_k}}{\Sigma_{\ell=1}^{K} e^{a_\ell}} \tag{4.8}$$

$$a_k = \sum_{i=0}^{d} w_{ki} x_i \tag{4.9}$$

Recall that in the generative classification method the problem is divided into sub problems by using the conditional probability, while in the discriminative approach the joint probability is determined by looking at the data directly. This is what $p(C_k|\mathbf{x})$ represents. However also, recall that we still need to determine the correct probability distribution represented by the data. This is achieved by determining the values of the weights of the linear operation. In the next section a method known as back propagation is discussed. Back propagation is the training algorithm used to determine the weight vector of all the layers in the neural network. Back propagation is an extension of the Gradient descent algorithm.

4.1.3 Back propagation algorithm (backprop)

In the previous section, the neural network architecture has been described as having N inputs M neurons and L layers. Each layer comprises M neurons of a maximum of N inputs times M neurons interconnections which embodies the inner product

of the inputs and unknown set of weights. The output of this inner product is then passed to a logistic squashing function that results in the output probabilities. The discriminative process is used here to determine the correct combination of weight vectors that accurately describe the training data. For neural networks, the weight vectors at each layer are determined through propagating the errors back through each preceding layer and adjusting the weights according to the errors propagated each time a batch of the data is processed. This process of continuously adjusting weights from back propagation continues until all the data is processed and a steady state has been reached. The steady state refers to the fact that the error has reached a steady and/or acceptable negligible value. This is often referred to in Machine Learning as convergence (Boden, 2002).

4.1.4 Gradient Descent

The last section ended stating that the back-propagation algorithm is an extension of the gradient descent algorithm. It has also been seen that back propagation works by propagating the error and making adjustments on the weights. In this section, the Gradient Descent algorithm is reviewed and how it is used in back propagation is examined.

The concept behind the Gradient descent algorithm is the fact that a function is optimized when the gradient of the function is equal to 0. Gradient descent algorithm is significant in Machine Learning applications because a cost function is easily defined for a particular Machine Learning application that is able to determine the error between the predicted value and the actual value. Then, the parameters of the problem can be adjusted until the derivative of the cost function using gradient descent is zero. Thus the Machine Learning algorithm adjusts its parameters until the error is minimised or removed.

A common error function or cost function for neural networks is the sum-of-squares error cost function. This is obtained by summing the difference between the actual value and the Machine Learning model value over the training set N.

$$E^n = \frac{1}{2}\sum_{k=1}^{K}(y_k^n - t_k^n)^2 \tag{4.10}$$

In a neural network having a weight matrix $\mathbf{W}$ of M neurons times N inputs, the resulting gradient is a vector of partial derivatives of E with respect to each element.

$$\nabla_{\mathbf{W}} E = \left(\frac{\partial E}{\partial w_{10}}, \ldots, \frac{\partial E}{\partial w_{ki}}, \ldots, \frac{\partial E}{\partial w_{Kd}} \right) \tag{4.11}$$

The adjustment on each weight therefore on each iteration is:

$$w_{kj}^{\tau+1} = w_{kj}^{\tau} - \eta \frac{\partial E}{\partial w_{kj}} \tag{4.12}$$

Where τ is the iteration and η is a constant learning rate which is a factor to speed up or slow down the rate of learning of the Machine Learning algorithm which in this case is the neural network.

4.2 RNN, LSTM and GRU Networks

Neural networks have become increasingly popular due to their ability to model non-linear system dynamics. Since their inception, there have been many modifications made to the original design of having linear affine transformations terminated with a nonlinear functions as the means to capture both linear and non-linear features of the target system. In particular, one of such neural network modifications, namely the recurrent neural network, has been shown to overcome the limitation of varying lengths in the inputs and outputs of the classic feed-forward neural network. In addition the RNN is not only able to learn non-linear features of a system but has also been shown to be effective at capturing the patterns in sequential data. This section develops recurrent neural networks (RNNs) from a specialised multi-layer Perceptron (MLP) or the deep neural network (DNN).

4.2.1 Deep Neural Networks (DNNs)

Deep neural networks have been accepted to be networks having multiple layers and capable of hierarchical knowledge representation (Yu and Deng, 2016). This will therefore include multi-layer Perceptrons (MLPs) having more than one hidden layer (Dahl et al., 2012) as well as deep belief networks (DBNs)(Mohamed et al., 2009, Yu et al., 2010) having a similar structure. Therefore, following the MLP architecture,

A DNN uses multiple hidden layers and generates distribution function, $p(c|x_t)$ on the output layer when an input vector $\mathbf{x}_t$ is applied. At the first hidden layer, activations are vectors evaluated using

$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)T}\mathbf{x}_t + \mathbf{b}^{(1)}) \tag{4.13}$$

The matrix $\mathbf{W}^{(1)}$ is the weight matrix and vector $b^{(1)}$, the bias vector for the layer. The function $\sigma(\cdot)$ is the point-wise non-linear function. DNNs activations $h^{(i)}$ at layer i, at arbitrarily many hidden layers after the first hidden layer, are subsequently hidden activations are determined from

$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)T}\mathbf{h}^{(i-1)} + \mathbf{b}^{(1)}) \tag{4.14}$$

The distribution over all the possible set of characters c is obtained in the final layer of the network in the exact way of a multi-layer Perceptron, that is, using soft max activation at the output layer of the form,

$$p(c = c_k|x_t) = \frac{exp(-(\mathbf{W}_k^{(s)T}h^{(i-1)} + b_k^{(1)}))}{\sum_j exp(-(\mathbf{W}_k^{(s)T}h^{(i-1)} + b_k^{(1)}))} \tag{4.15}$$

$W_k^{(s)}$ and $b_k^{(k)}$ respectively are the output weight matrix and the scalar bias term of the k-th neuron. Accordingly, sub gradients for all parameters in the DNN are utilised to back propagate errors in weights during training for gradient-based optimisation techniques. In DNN-HMM speech models, DNNs are trained to predict probability distributions over senones. However, in the model neural network described in section 4.3.1, of this thesis, predicts per character conditional distributions. Combining equations (4.12, 4.13, 4.14 and 4.15) the following simplified

algorithm ensues

Result: Optimal weights

initialise weights randomly;

while *error is significant or epochs less than maximum* **do**

    forward computation in equation (4.13 and 4.14);

    determine layer wise error for weights and biases $\Delta_{\mathbf{W}}E$ and $\Delta_{\mathbf{b}}E$;

    update weights and biases according to gradient descent. Equation (4.12);

end

Algorithm 1: DNN training algorithm

4.2.2 Recurrent Neural Networks

One of the two advantages RNNs have over regular DNNs is the ability to capture varying lengths of outputs to inputs. That is for tasks such as language translation where there is no one to one correspondence of number of words in a sentence for example from the source language to the output destination language. At the same time the sentence length appearing at the input and that appearing at the output differ for different sentences. This is the first problem of varying lengths for input and output sequences.

The second issue that RNNs effectively contain as opposed to DNNs is capturing temporal relationships between the input sequences. As was realised for hidden Markov models, it was seen that the HMM modeled not just observation likelihoods but also transition state likelihoods which were latent or hidden variables. By tying the output of previous neuron activations to present neuron activations, a DNN inherits a cyclic architecture becoming a recurrent neural network (RNN). As a result, an RNN is to able capture previous hidden states and in the process derive memory-like capabilities (Yu and Deng, 2016).

In speech processing, it is observed that for a given utterance, there are various temporal dependencies which may not be sufficiently captured by DNN-based systems because DNN systems ignore previous hidden representations and output distributions at each time step t. The DNN derives its output using only the feature inputs x_t. The architecture of RNN to enable better modelling of temporal dependencies present in a speech is given in (Hannun et al., 2014b, Yu and Deng,

2016).

$$h_t^{(j)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(j)T} h_{t-1}^{(j)} + b^{(j)})) \tag{4.16}$$

It can be seen in equation (4.16) above that given a selected RNN hidden layer j, a temporally recurrent weight matrix $W^{(f)}$ is computed for output activations $h_{t-1}^{(j)}$ for the hidden activation vector of layer j at time step $t-1$ such that the output contributes to the standard DNN output of $\mathbf{W}^{(j)T} h_t^{(i-1)}$. It can also be seen from equation (4.16) that the temporal recurrent weight matrix computation is a modified version of the standard DNN weight matrix computation and that the overall output is a superposition of the two.

Since computations for a RNN are the same as those described in standard DNN evaluations, it is possible to compute the sub gradient for RNN architecture using the back propagation algorithm. The modified algorithm appropriately called back propagation through time (BPTT) (Boden, 2002, Jaeger, 2002) is derived in section 4.2.3 below.

4.2.3 Back propagation through time (BPTT) algorithm

First we define an arbitrary but carefully chosen number of time steps $t = 1, 2, \ldots, T$ such that at each time step the states of the neuron activations $j = 1, 2, \ldots, J$ are captured. Using the sum-squared error as the cost function

$$E = c \sum_{t=1}^{T} ||\mathbf{l}_t - \mathbf{y}_t||^2 = c \sum_{t=1}^{T} \sum_{j=1}^{L} (l_t(j) - y_t(j))^2 \tag{4.17}$$

Where c is a gradient descent convenience factor in Equation (4.17). $||\mathbf{l}_t - \mathbf{y}_t||$ is the modulus of the difference between the actual output $\mathbf{y}_t$ and the label vector $\mathbf{y}_t$ at time t. The two-step BPTT algorithm described in Yu and Deng (2016) is involves the recursive computation of the cost function and updating of the network weights.

For each of these steps recall from equation (4.16) the activation of a hidden layer is a result of the composition of the regular DNN activation and an activation generated from weights from the previous time step.

The error term at final time t=T is

$$\delta_T^y(j) = -\frac{\delta E}{\delta y_T(j)}\frac{\delta y_T(j)}{\delta v_T(j)} = (l_T(j) - y_T(j))g'(v_T(j)) \text{ for } j = 1, 2, \ldots, L \tag{4.18}$$

or

$$\delta_T^y = (\mathbf{l}_T - \mathbf{y}_T) \bullet g'(\mathbf{v}_T) \tag{4.19}$$

The error at the hidden layer is given as

$$\delta_T^h(j) = -\left(\sum_{i=1}^{L}\frac{\partial E}{\partial v_T(i)}\frac{\partial v_T(i)}{\partial h_T(j)}\frac{\partial h_T(j)}{\partial u_t(j)}\right) = \sum_{i=1}^{L}\delta_T^y(i)w_{hy}(i,j)f'(u_T(j)) \text{ for } j = 1, 2, ..., N \tag{4.20}$$

or $\delta_T^h = \mathbf{W}_{hy}^T\delta_T^y \bullet f'(\mathbf{u}_T)$ where $\bullet$ is element-wise multiplication.

The recursive component for other time frames, $t = T-1, T-2, \ldots, 1$, the error term is determined as

$$\delta_t^y(j) = (l_t(j) - y_t(j))g'(v_t(j)) \text{ for } j = 1, 2, \ldots, L \tag{4.21}$$

or

$$\delta_t^y = (\mathbf{l}_t - \mathbf{y}_t) \bullet g'(\mathbf{v}_t) \tag{4.22}$$

Therefore the output units are

$$\begin{aligned}\delta_t^h(j) &= -\left[\sum_{i=1}^{N}\frac{\partial E}{\partial \mathbf{u}_{t+1}(i)}\frac{\partial \mathbf{u}_{t+1}(i)}{\partial h_t(j)} + \sum_{i=1}^{L}\frac{\partial E}{\partial v_t(i)}\frac{\partial v_t(i)}{\partial h_t(j)}\right]\frac{\partial h_t(j)}{\partial u_t(j)} \\ &= \left[\sum_{i=1}^{N}\delta_{t+1}^h(i)w_{hh}(i,j) + \sum_{i=1}^{L}\delta_t^y(i)w_{hy}(i,j)\right]f'(u_t(j)) \text{ for } j = 1, \ldots, N \\ \text{or } \delta_t^h &= \left[\mathbf{W}_{hh}^\top\delta_{t+1}^h + \mathbf{W}_{hy}^\top\delta_t^y\right] \bullet f'(\mathbf{u}_t)\end{aligned} \tag{4.23}$$

Note that the error terms are propagated back from hidden layer at time frame $t+1$ to the output at time frame t.

Update of RNN Weights

The weights are updated using the error terms determined in the previous section. For the output weight matrices, we have

$$\begin{aligned} w_{hy}^{new}(i,j) &= w_{hy}(i,j) - \gamma \sum_{t=1}^{T} \frac{\partial E}{\partial v_t(i)} \frac{\partial v_t(i)}{\partial w_{hy}(i,j)} = w_{hy}(i,j) - \gamma \sum_{i=1}^{T} \delta_t^y(i) h_t(j) \\ \text{or } \mathbf{W}_{hy}^{new} &= \mathbf{W}_{hy} + \gamma \sum_{t=1}^{T} \delta_y^t \mathbf{h}_t^\top \end{aligned} \tag{4.24}$$

For the input weight matrices, we get

$$w_{xh}^{new}(i,j) = w_{xh}(i,j) - \gamma \sum_{t=1}^{T} \frac{\partial E}{\partial u_t(i)} \frac{\partial u_t(i)}{\partial w_{xh}(i,j)} = w_{xh}(i,j) - \gamma \sum_{t=1}^{T} \delta_t^h(i) x_t(j) \tag{4.25}$$

or

$$\mathbf{W}_{xh}^{new} = \mathbf{W}_{xh} + \gamma \sum_{t=1}^{T} \delta_h^t \mathbf{x}_t^\top \tag{4.26}$$

For the recurrent weight matrices we have

$$\begin{aligned} w_{hh}^{new}(i,j) &= w_{hh}(i,j) - \gamma \sum_{t=1}^{T} \frac{\partial E}{\partial u_t(i)} \frac{\partial u_t(i)}{\partial w_{hh}(i,j)} \\ &= w_{hh}(i,j) - \gamma \sum_{t=1}^{T} \delta_h^t(i) h_{t-1}(j) \\ \text{or } &= \mathbf{W}_{hh}^{new} = \mathbf{W}_{hh} + \gamma \sum_{t=1}^{T} \delta_h^t \mathbf{h}_{t-1}^\top \end{aligned} \tag{4.27}$$

In the BPTT algorithm the sub gradients are summed over all time frames. The algorithm is summarised below:

Data: $\{\mathbf{x}_t, \mathbf{I}_t\} 1 \leq t \leq T$

Result: Optimal weights

//$\mathbf{x}_t$ is the input feature sequence //$\mathbf{I}_t$ is the label sequence;

initialise weights randomly;

for *error is significant or epochs less than maximum* **do**

  for $t \leftarrow 1; t <= T; t \leftarrow t+1$ **do**

    //forward computation ;

    $\mathbf{u}_t \leftarrow \mathbf{W}_{xh} + \mathbf{W}_{hh}\mathbf{h}_{t-1}$;

    $\mathbf{h}_t \leftarrow f(\mathbf{u}_t)$;

    $\mathbf{v}_t \leftarrow \mathbf{W}_{hy}\mathbf{h}_t$;

    $\mathbf{y}_t \leftarrow g(\mathbf{v}_t)$

  end

  begin

    //backprop through time ;

    $\delta_T^y = (\mathbf{l}_T - \mathbf{y}_T) \bullet g'(\mathbf{v}_T)$;

    $\delta_T^h = \mathbf{W}_{hh}^\top \delta_{t+1}^h + \mathbf{W}_{hy}^\top \delta_T^y \bullet f'(\mathbf{u}_T)$;

    for $t \leftarrow T-t; t >= 1; t \leftarrow t-1$ **do**

      $\delta_t^y = (\mathbf{l}_t - \mathbf{y}_t) \bullet g'(\mathbf{v}_t)$;

      $\delta_t^h = \left[\mathbf{W}_{hh}^\top \delta_{t+1}^h + \mathbf{W}_{hy}^\top \delta_t^y\right] \bullet f'(\mathbf{u}_t)$;

    end

  end

  update weights and biases according to gradient descent;

  begin

    $\mathbf{W}_{hy}^{new} = \mathbf{W}_{hy} + \gamma \sum_{t=1}^{T} \delta_y^t \mathbf{h}_t^\top$;

    $\mathbf{W}_{hh}^{new} = \mathbf{W}_{hh} + \gamma \sum_{t=1}^{T} \delta_h^t \mathbf{h}_{t-1}^\top$;

  end

end

Algorithm 2: RNN training algorithm

4.2.4 LSTMs and GRUs

A special implementation of the RNN called the Long Short Term Memory (LSTM) has been designed to capture patterns over particularly long sequences of data and thus is an ideal candidate for generating character sequences while preserving syntactic language rules learned from the training data.

The internal structure and working of the LSTM cell is documented by its creators in Sak et al. (2014). The ability to recall information over extended sequences results from the internal gated structure which performs a series of element wise multiplications on the inputs and internal state of the LSTM cell at each time step. In addition to the output neurons which in this text we refer to as the write gate and denote as the current cell state, $\mathbf{c}_t$, three additional gates (comprising a neural network sub-layer) located within the LSTM cell are the input gate, the forget gate and the output gate. Together with the initial current state cell, these gates along with the current-state cell itself enable the LSTM cell architecture to store information, forward information, delete information and receive information. Generally however, the LSTM cell looks like a regular feed-forward network having a set of neurons capped with a nonlinear function. The recurrent nature of the network arises, however due to the fact that the internal state of the RNN cell is rerouted back as an input to the RNN cell or input to the next cell in the time-series giving rise to sequence memory within the LSTM architecture. Mathematically, these gates are formulated as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}^{(xi)}\mathbf{x}_t + \mathbf{W}^{(hi)}\mathbf{h}_{t-1} + \mathbf{W}^{(ci)}\mathbf{c}_{t-1} + \mathbf{b}^{(i)}) \tag{4.28}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^{(xf)}\mathbf{x}_t + \mathbf{W}^{(hf)}\mathbf{h}_{t-1} + \mathbf{W}^{(cf)}\mathbf{c}_{t-1} + \mathbf{b}^{(f)}) \tag{4.29}$$

$$\mathbf{c}_t = \mathbf{f}_t \bullet \mathbf{c}_{t-1} + \mathbf{i}_t \bullet \tanh(\mathbf{W}^{(xc)}\mathbf{x}_t + \mathbf{W}^{(hc)}\mathbf{h}_{t-1} + \mathbf{b}^{(c)}) \tag{4.30}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^{(xo)}\mathbf{x}_t + \mathbf{W}^{(ho)}\mathbf{h}_{t-1} + \mathbf{W}^{(co)}\mathbf{c}_{t-1} + \mathbf{b}^{(o)}) \tag{4.31}$$

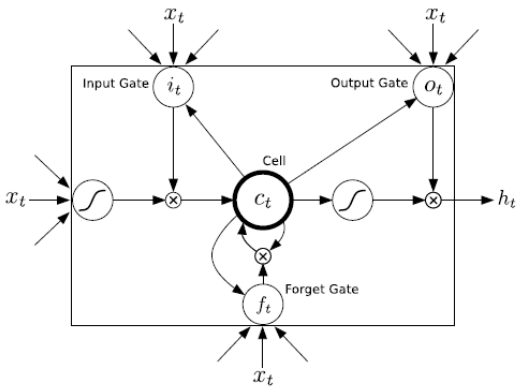

Figure 4.3: An LSTM Cell (Graves et al., 2013)

$$\mathbf{h}_t = \mathbf{o}_t \bullet \tanh(\mathbf{c}_t) \tag{4.32}$$

The gates in the above formula are illustrated in Figure 4.3. $\mathbf{i}_t$ represents the input gate, $\mathbf{f}_t$ is the forget gate and $\mathbf{o}_t$ represents the output gate. At each of these gates therefore, the inputs consisting of hidden states in addition to the regular inputs are multiplied by a set of weights and passed through a soft-max function. These weights during training learn whether the gate will, during inference, open or not. In summary, the input gate tells the LSTM whether or not to receive new information, the forget gate determines whether the current information it already has from the previous step should be kept or dropped and the output gate determines what should be forwarded to the next LSTM cell. Note also that the LSTM has two sigmoid ($tanh$) activation functions utilised at the input and output of the current cell $\mathbf{c}_t$.

One particular variant of the original LSTM model is the GRU cell. Though simpler than an LSTM cell the GRU cell performs equally efficiently. The GRU cell is a subset implementation of the LSTM cell. Rather than using the output gate of the LSTM, this gate is omitted in the GRU and the output result of the other internal gates are always forwarded. The second simplification is a merge of the internal gate state vectors into a single vector $\mathbf{h}_{(t)}$. This merged gate here referred to as $\mathbf{z}(t)$, controls both the forget gate an the input gate and acts as follows. Whenever a value is retained by the cell the previous value is erased first. That is, if the gate controller outputs a 1, in the LSTM this corresponds to the input gate is open and the forget gate is closed. Therefore if $\mathbf{z}(t)$ it outputs a 0, the reverse

happens for the input gate and the forget gate in the LSTM. There is, however, a new gate controller, $\mathbf{r}(t)$, which determines which portion of the previous state will be shown at the output (Cho et al., 2014).

The architecture of a GRU is formulated as follows:

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}^T \cdot \mathbf{x}_{(t-1)}) \tag{4.33}$$

$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}^T \cdot \mathbf{x}_{(t-1)}) \tag{4.34}$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)})) \tag{4.35}$$

$$\mathbf{h}_{(t)} = (1 - \mathbf{z}_{(t)}) \otimes (\mathbf{h}_{(t-1)}) + \mathbf{z}_{(t)} \otimes \mathbf{g}_t \tag{4.36}$$

Due to the light-weight nature of the GRU cell, it is common practice to use GRU cells in place of LSTM cells. This precedence achieves the much desired lighter computation load on the actual hardware performing the RNN training. As each of the gates required in an LSTM cell comprises high density matrix multiplication operations in themselves, the condensation of two gates into one and the omission of the output gate within GRU cells pushes towards halving the architectural complexity and coupled with the equally efficient performance of the GRU when compared to the LSTM cell ultimately serves as an overall improvement on the LSTM architecture. For these reasons, GRUs have highly appealing features when compared to LSTMs and was the RNN cell of choice used for the study in this report.

4.3 Deep speech architecture

This work makes use of an enhanced RNN architecture called the Bi-directional Recurrent Neural Network (BiRNN). While Hannun et al. (2014b) assert that forward recurrent connections does reflect the sequential relationships of an audio waveform, perhaps the BiRNN model achieves a more robust sequence model.

The BiRNN is a preferred end to end mechanism due to the length of sequence

over which temporal relationships can be captured. This implies that BiRNNs will be suited for capturing temporal relationships over much longer sequences than a forward only RNN, because hidden state information is preserved in both forwards and backwards direction.

In addition, such a model has a notion of complete sentence or utterance integration, having information over the entire temporal extent of the input features when making each prediction.

The formulation of the BiRNN is derived by starting off with the basic RNN architecture which is referred to as the forward architecture. From the forward architecture we derive the backward architecture. If we choose a temporally recurrent layer j, the BiRNN forward and backward intermediate hidden representation $h_t^{(f)}$ and $h_t^{(b)}$ is given as.

$$h_t^{(f)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(f)T} h_{t-1}^{(j)} + b^{(j)})) \tag{4.37}$$

$$h_t^{(b)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(b)T} h_{t+1}^{(b)} + b^{(j)})) \tag{4.38}$$

Temporal weight matrices $W^{(f)}$ and $W^{(b)}$ propagate $h_t^{(f)}$ and $h_t^{(b)}$ forward and backward in time respectively.

Hannun et al. (2014b) points out that the recurrent forward and backward components are evaluated entirely independent of each other and for optimal training, a modified non linearity function $\sigma(z) = min(max(z, 0), 20)$ is recommended.

The final BiRNN representation $h_t^{(j)}$ for the layer is now the superposition of the two RNN components,

$$h_t^{(j)} = h_t^{(f)} + h_t^{(b)} \tag{4.39}$$

Also note that back propagation through time (BPTT) sub gradient evaluations are computed from the combined BiRNN structure directly during training.

4.3.1 Connectionist Temporal Classification (CTC)

The term CTC stands for Connectionist Temporal classification. This algorithm was designed to solve the problem of fuzzy alignment between the source input

data and the output classification desired from the Machine Learning system. This type of fuzzy alignment is observed in speech recognition systems since the same speech in either the same individual or different individuals will have different signal forms. This is a many to one relationship between the input signal and the output classification that is also dependent on the style of speaking at the moment when the utterance is said. Unlike hybrid DNN-HMM networks the CTC algorithm deploys an end-to-end framework that models all aspects of the input sequence in a single neural network, therefore discarding the need for an HMM interpretation of the input sequence. In addition, the CTC method does not require pre-segmented training data at the same time output classification is made independent of post-processing.

CTC works by making predictions at any point in the input sequence. For the case of speech modelling, CTC makes a character prediction for every time step of the raw audio input speech signal. Although this initially seems counter intuitive, this method models the many to one relationship seen in the fuzzy audio speech to text alignment.

For hybrid DNN-HMM systems, speech or more accurately, acoustic models, require separate training of targets for every time-slice in the input sequence. Secondly, and as a consequence of this, it becomes necessary to segment the audio sequence, in order to provide targets for every time-slice. A third consequence is the limitation of DNNs previously discussed. As the DNN network only outputs local classifications, global aspects such as the likelihood of two consecutive labels appearing together cannot be directly modelled. Without an external model, usually in the form of a language model, the hybrid speech model will significantly degrade performance.

In the CTC case, so long as the overall sequence of labels is correct the network can be optimised to correct the temporal or fuzzy alignments. Since this many to one fuzzy alignment is simultaneously modelled in CTC, then there is no need for pre-segmented data. At the same time, CTC computes probabilities of complete label sequences, hence external post-processing required by hybrid models is eliminated.

Similar to the HMM sequence model, the CTC algorithm is a sequence model that predicts the next label in a sequence as a cumulative of previous sequences. This section develops the CTC loss function borrowing concepts used in HMM models such as the forward backward algorithm as outlined in (Graves et al., 2006). In

the following paragraph we introduce terminology associated with the CTC loss function.

Given two symbols A and $\mathcal{B}$ such that A has a many to one relationship with $\mathcal{B}$, signifying the temporal nature of the classification. The symbol A represents an alphabet from which a sequence of the output classifications are drawn from. This CTC output consists of a soft-max layer in a BiRNN (bidirectional recurrent neural network).

This output models the probability distribution of a complete sequence of arbitrary length $|A|$ over all possible labels in A from activations within $|A|$. An extra activation is given to represent the probability of outputting a *blank*, or no label. At each time-step leading up to the final step, the probability distribution estimated as distribution over all possible label sequences of length leading up to that of the input sequence.

It is now possible to define the extended alphabet $A' = A \cup \{blank\}$, also, $y_{t,p}$ as the activation of network output p at time t. Therefore $y_{t,p}$ is the probability that the network will output element $p \in A'$ at time t given that x is the input sequence of length T. The distribution sought after $Pr(\pi|x)$, is the conditionally-independent distribution over the subset A'^T where A'^T denotes the set of length T sequences in A'.

$$\Pr(\pi \mid x) = \prod_{t=1}^{T} y_{t,\pi_t} \tag{4.40}$$

From Equation (4.40), it is now possible to define the many-to-one mapping $\mathcal{B} : A^{\prime T} \rightarrow A^{\leq T}$. This is the mapping of a set A'^T, which indicates the paths, onto another set $A^{\leq T}$, that of possible labels in x. $\mathcal{B}$ then becomes a sequence of symbols with length less than or equal to T over A. Note that $\mathcal{B}$ is a set containing sequential symbols belonging to the set A and not. A' because there is no blank symbol in $\mathcal{B}$. This is achieved when first take out all repeated labels and then take out all the blanks from the sequence A'^T. For instance,

$$\begin{aligned} \mathcal{B}(a - ab-) &= aab \\ \mathcal{B}(-aa - -abb) &= aab. \end{aligned} \tag{4.41}$$

The mapping obtained by $\mathcal{B}$ is equivalent to when the output switches from not predicting a new symbol to predicting a symbol or from predicting one symbol to

another symbol assuming this was also possible. Intuitively, the probability of $\mathcal{B}$ which is the labelling of $l \in A^{\leq T}$ being a many to one of $A^{`T}$ is determined by summing over all the paths in $A^{\prime T}$ mapped onto it by $\mathcal{B}$. Thus:

$$\Pr(l \,|\, x) = \sum_{\pi \in \mathcal{B}^{-1}(l)} \Pr(\pi \,|\, x) \tag{4.42}$$

This mapping makes CTC robust to unsegmented data as it predicts all the labels where they occur and later the 'collapsed' sequence will be extended over the approximate period where the previous extended sequence occurred thus aligning labels to input sequences on-the-fly without knowing in advance where label to input sequence alignments occur.

4.3.2 Forward-backward algorithm

The forward-backward algorithm is used to estimate the probability of a point in the sequence as the product of all point leading up to that point from the initial state, the forward variable (α), multiplied by the probability of all the points from that state to the end of the sequence, the backward variable (β).

The difference between this estimation and that determined from equation (4.42) is the fact that the forward-backward algorithm converts equation (4.42) into a form that is both recursive as well as reduces the computational complexity from an otherwise intractable computation to one that is readily computable.

With CTC, consider a modified "label sequence" l', that caters for blank characters in between regular ones l, as defined in A. Thus, if U is defined as the length of l. Then U' is of length $2U + 1$. CTC therefore integrates probability distributions of transitions between blank and non-blank labels at the same time CTC calculates those transition occurring between pairs of distinct non-blank labels. The forward variable, $\alpha(t, u)$ now becomes the summed probability of all length t paths that are mapped by $\mathcal{B}$ onto the length $\lfloor u/2 \rfloor$ prefix of l. (Note, $\lfloor u/2 \rfloor$ is the floor of $u/2$, the greatest integer less than or equal to $u/2$.) For some sequence s, let $s_{p:q}$ denote the sub-sequence $s_p, s_{p+1}, \ldots, s_{q-1}, s_q$, and define the set

$V(t,u) \equiv \{\pi \in A'^t : \mathcal{B}(\pi) = l_{1:\lfloor u/2 \rfloor} \text{ and } \pi_t = l'_u\}$. $\alpha(t,u)$ then becomes

$$\alpha(t,u) \equiv \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y_{i,\pi_i} \tag{4.43}$$

The forward variables at time t is calculated recursively from the preceding values at time $t-1$ and expressed as the sum of the forward variables with and without the final blank at time T.

$$\Pr(l \,|\, x) = \alpha(T, U') + \alpha(T, U' - 1) \tag{4.44}$$

For the initial conditions, correct paths begin with a blank symbol (b) and the first symbol l (l_1):

$$\begin{aligned} \alpha(1,1) &= y_{1,b} \\ \alpha(1,2) &= y_{1,l_1} \\ \alpha(1,u) &= 0, \, \forall u > 2 \end{aligned} \tag{4.45}$$

The forward variable then takes the following recursive form:

$$\alpha(t,u) = y_{t,l'_u} \sum_{i=f(u)}^{u} \alpha(t-1,i) \tag{4.46}$$

where

$$f(u) = \begin{cases} u-1, & \text{if } l'_u = \textit{blank} \text{ or } l'_{u-2} = l'_u \\ u-2, & \text{otherwise} \end{cases} \tag{4.47}$$

Figure 4.4 expresses the recurrence relation for $\alpha(t,u)$. While t is expressed on the x axis, u is illustrated on the y axis. The CTC algorithm assumes that outputs of the network potentially alternate between blank symbols indicated as black circles and non-blank elements, the white circles, all in l'. The sequential graph constructed from this 2-dimensional matrix show computational dependencies between sequential pairs of the recurrence relation for $\alpha(t,u)$. Therefore, the value $\alpha(2,3)$, formed from $\alpha(1,2)$, corresponds to the *blank* symbol at $t = 2$ and $u = 3$, is . Also, $\alpha(2,2)$, equivalent to the symbol c at $t = 2$ and $u = 2$, is gotten from $\alpha(1,2)$ and $\alpha(1,1)$. Note that there are not enough time steps when $u < U'2(Tt)1$, therefore, $\alpha(t,u) = 0$.

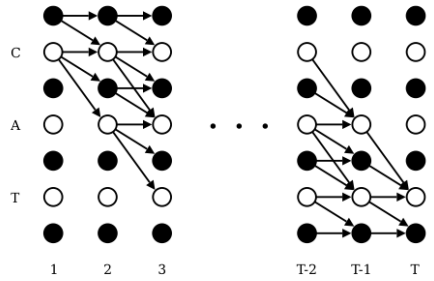

Figure 4.4: Beam Search Lattice Structure (Graves et al., 2006)

Also note the boundary condition;

$$\alpha(t,0) = 0 \quad \forall t \tag{4.48}$$

The backward variable $\beta(t,u)$ ais built similarly to the forward variable. Rather than moving from the start of the sequence to we define the path starting at $t+1$ that completes the sequence at T when appended to any path $\hat{\pi}$ that generates $\alpha(t,u)$. Then, assuming $W(t,u) \equiv \{\pi \in A'^{T-t} : \mathcal{B}(\hat{\pi}+\pi) = l \ \forall \hat{\pi} \in V(t,u)\}$, therefore

$$\beta(t,u) \equiv \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-t} y_{t+i,\pi_i} \tag{4.49}$$

The backward variable is therefore equivalently initialised as thus

$$\begin{aligned} \beta(T,U') &= 1 \\ \beta(T,U'-1) &= 1 \\ \beta(T,u) &= 0, \ \forall u < U'-1 \end{aligned} \tag{4.50}$$

The recursion rule is defined as follows:

$$\beta(t,u) = \sum_{i=u}^{g(u)} \beta(t+1,i) y_{t+1,l'_i} \tag{4.51}$$

similarly,

$$g(u) = \begin{cases} u+1, & \text{if } l'_u = blank \text{ or } l'_{u+2} = l'_u \\ u+2, & \text{otherwise} \end{cases} \tag{4.52}$$

4.3.3 CTC Loss function

The cross entropy error is a loss function used to measure accuracy of probabilistic measures. It is calculated as the negative log probability of a likelihood measure. The CTC loss function $\mathcal{L}(S)$ uses the cross entropy loss function of and is defined as the cross entropy error of correctly labelling all the training samples in some training set S:

$$\mathcal{L}(S) = -\ln \prod_{(x,z)\in S} \Pr(z \,|\, x) = -\sum_{(x,z)\in S} \ln \Pr(z \,|\, x) \tag{4.53}$$

where z is the output label and x is the input sequence. Since $\mathcal{L}(S)$ in equation 4.53 is differentiable, this loss function can be back propagated to the softmax layer in the BiRNN configuration discussed in section 4.3.

$$\mathcal{L}(x, z) \equiv -\ln \Pr(z \,|\, x) \tag{4.54}$$

and therefore

$$\mathcal{L}(S) = \sum_{(x,z)\in S} \mathcal{L}(x, z) \tag{4.55}$$

From the definition of the forward and backward variables ($\alpha(t, u)$ and $\beta(t, u)$), we also establish that $X(t, u) \equiv \{\pi \in A'^T : \mathcal{B}(\pi) = z,\ \pi_t = z'_u\}$, such that

$$\alpha(t, u)\beta(t, u) = \sum_{\pi\in X(t,u)} \prod_{t=1}^{T} y_{t,\pi_t} \tag{4.56}$$

then substituting $\Pr(\pi \,|\, x)$ from the expression in equation **??**, we have

$$\alpha(t, u)\beta(t, u) = \sum_{\pi\in X(t,u)} \Pr(\pi \,|\, x) \tag{4.57}$$

Also observe that $\Pr(l \,|\, x)$ is equivalent to the total probability $\Pr(z \,|\, x)$. Paths going through z'_u at time t can be obtained as summed over all u to get

$$\Pr(z \,|\, x) = \sum_{u=1}^{|z'|} \alpha(t, u)\beta(t, u) \tag{4.58}$$

Thus a sample loss is determined by

$$\mathcal{L}(x,z) = -\ln \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{4.59}$$

and therefore the overall loss is given by

$$\mathcal{L}(S) = -\sum_{(x,z)\in S} \ln \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{4.60}$$

In the model described in this work, the gradient $\mathcal{L}(x,z)$is computed using TensorFlow's automatic differentiation capabilities. In practice, computations soon lead to underflow. However, the log scale, being used in the above loss function calculations avoids this situation and another useful equation in this context is

$$\ln(a+b) = \ln(a) + \ln(1+e^{\ln b - \ln a}) \tag{4.61}$$

4.4 Chapter Summary

Deep Neural Networks (DNNs) are at the centre of the models developed within this research. They are able to overcome the challenge of complex modelling of latent information when discriminating directly from the data. To this extent, they tend to be data intensive in nature. In this chapter, neural network architectures and algorithms were considered. The Chapter begins with the rudimentary perceptron algorithm which is the precursor to the logistic regression algorithm. The Neural Network and Multi-Layer Peceptron (MLP), uses logistic regression concept and adds an extra layer of neurons and back propagation algorithm to optimise classifications.

The Deep Neural Networks used within this research are special DNNs which are able to identify patterns in data having sequential patterns. These are the deep Recurrent Neural Networks (RNNs). It is shown in this Chapter that RNNs are able to learn the recurrent relationship information by modifying their architecture such that the data paths among the neurons are modified so that hidden states are also used as inputs. In addition the back propagation algorithm is also modified in

terms of datapaths of the algorithm output to reflect the sequential structure of the neural network.

Special categories of RNNs used to build speech and language models developed in this thesis are the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) RNN cells. In this research were utilised for development of a character-based language model and the Bidirectional RNN (Bi-RNN) used for development of the speech model.

A special algorithm, the Connectionist Temporal Classification (CTC) algorithm, also employed by the BiRNN is described in this Chapter. The CTC algorithm overcomes the challenge of a character-based speech model when considering misaligned nature of audio data between specific points in the audio that correspond to equivalent characters in the transcription. In addition, it is also discussed how the CTC algorithm utilises the forward-backward algorithm to perform classification. In a later Chapter 7, the prefix beam search algorithm is described, and, in combination with output probabilities obtained from the CTC algorithm and a language model, performs decoding of the output into the actual speech-to-text translations.

Chapter 5

Background 2: Deep Scattering network

Curve fitting is a very common theme in pattern recognition. The concept of invariant functions convey mapping functions that approximate a discriminating function when a parent function is reduced from a high dimensional space to a low dimensional space Mallat (2016). In this chapter an invariance function called a scattering transform enables invariance of groups of deformations that could apply to speech signals thereby preserving higher level characterisations useful for classifying speech sounds. Works done by (Andén and Mallat, 2011, Peddinti et al., 2014, Sainath et al., 2014, Zeghidour et al., 2016) have shown that when the scattering spectrum are applied to speech signals and used as input to speech systems have state of the art performance. In particular Sainath et al. (2014) shows 4-7% relative improvement in word error rates (WER) over Mel frequencies cepstral coefficients (MFCCs) for 50 and 430 hours of English Broadcast News speech corpus. While experiments have been performed with hybrid HMM-DNN systems in the past, this thesis focuses on the use of scatter transforms in end-to-end RNN speech models.

This chapter iterates the use of the Fourier transform as the starting analysis function for building invariant functions and then discusses the Mel filter bank solution and then establishes why the scattering transform through the wavelet modulus operator provides better invariance features over the Mel filters.

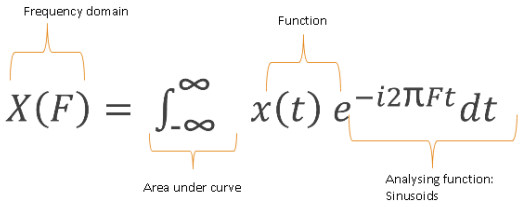

Figure 5.1: Fourier Equation

5.1 Fourier transform

The Fourier transform often referred to as the power spectrum, allows us to discover frequencies contained within a signal. The Fourier transform is a convolution between a signal and a complex sinusoid from $-\infty$ to $+\infty$ (Figure 5.1).

From the orthogonal property of complex exponential function, two functions are orthogonal if $\int f(x)g(x) = 0$ where f(x) and g(x) are complementary functions, one being referred to as the analysis equation and the other referred to as the synthesis function.

If the discrete form of the Fourier transform analysis equation is given by

$$a_k = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{\left(-j\frac{2\pi kt}{T}\right)} \tag{5.1}$$

Then, the corresponding synthesis equation is given by

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{\left(j\frac{2\pi kt}{T}\right)} \tag{5.2}$$

Recall that $x(t)$ is the original signal while a_k is the Fourier Series coefficient. This coefficient indicates the amplitude and phase of the original signal's higher order harmonics indexed by k such that higher values of k correspond to higher frequency components. In a typical spectrogram (Figure 5.2), it can be seen that the energy of the signal is concentrated about a central region and then harmonic spikes of energy content exponentially decrease and taper off. Therefore in Figure 5.2, the energies are concentrated at frequencies of about 100, 150 and 400 hertz.

The Fourier transform discussed in the preceding paragraph constitutes a valuable tool for the analysis of the frequency component of a signal. However is not able to determine when in time a frequency occurs hence is not able to analyse time

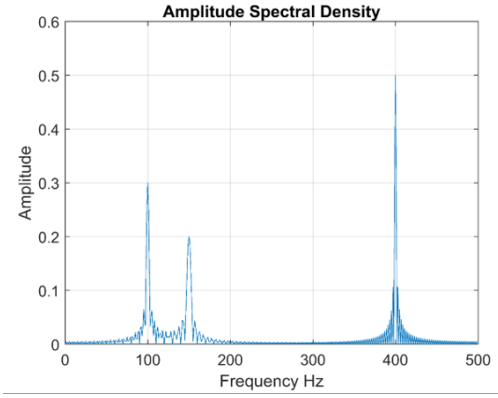

Figure 5.2: Sample Spectrogram
cwt (2015)

related signal deformations. The Short-time Fourier Transform (STFT) attempts to salvage this by windowing the signal in time signal and performing Fourier transforms over sliding windows sections of the original signal rather than the whole signal. There is however, a resolution trade off that ensues from this operation such that, the higher the resolution in time accuracy, the lower the frequency accuracy and vice versa. In the next section on the continuous wavelet transform, how the wavelet transform improves on the weaknesses of the Fourier Transform and the STFT is reviewed.

5.2 Wavelet transform

The continuous wavelet transform can be defined as a signal multiplied by scaled and shifted version of a wavelet function $\psi(t)$ referred to as the mother wavelet. The time-frequency tile-allocation of the three basic transforms examined in the first part of this chapter is illustrated in Figure 5.3

It can be seen here that for the Fourier transform there is no time information obtained. In the STFT, as there is no way of telling where in time the frequencies are contained, the STFT makes a blanket range of the resolution of the window and is therefore equally tiled potentially losing information based on this setup. For the case of the wavelet, because it is a scaled and shifted convolution, it takes care of the this problem providing a good resolution in both time and frequency. The

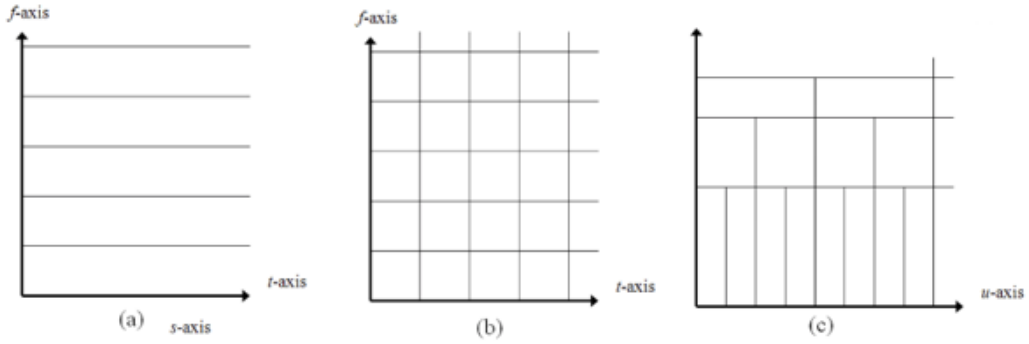

Figure 5.3: Time frequency tiling for (a) Fourier Transform (b) Short-time Fourier Transform (STFT) (c) Wavelet transform

fundamental representation of the continuous wavelet function is:

$$C(a,b) = \int f(t) \frac{1}{\sqrt{a}} \psi \left(\frac{t-b}{a} \right) dt \tag{5.3}$$

In this equation, a and b respectively represent the scaling and shifting resolution variables of the wavelet function. This is referred to as a mother wavelet. A few other mother wavelet functions discussed later in this chapter. Generally a mother wavelet is identified as being energy spikes in an infinite signal whose accumulative energy sums to zero.

5.3 Discrete and Fast wavelet transform

Synthesis and analysis equations (5.2 and 5.1) can be formulated as a linear combination of the basis $\phi_k(t)$ such that the basis, $\phi_k(t) = e^{j2\pi kt}$, and it's conjugate or orthonormal basis, $\tilde{\phi}_k(t) = e^{-j2\pi kt}$, equations (5.2 and 5.1) now become

$$x(t) = \sum_k a_k \phi_k \tag{5.4}$$

$$a_k = \int x(t) \tilde{\phi}_k(t) \tag{5.5}$$

With respect to scaling and shifting variables of continuous wavelet transforms in equation (5.3), a similar linear combination transformation can be applied by constructing orthonormal bases parameters, referred to as scaling (ϕ) and translating

(ψ) functions. For example, a simple Haar mother wavelet transform associated with a delta function, it is seen that:

$$\phi_{j,k}(t) = 2^{j/2}\phi(2^j t - k) \tag{5.6}$$

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k) \tag{5.7}$$

where j is associated with the dilation (scaling) parameter and k is associated with the position (shifting) parameter. If the Haar coefficients $h_{(.)}[n] = \{1/\sqrt{2}, 1/\sqrt{2}\}$ are extracted we have the following dilation and position parameters.

$$\phi(t) = h_\phi[n]\sqrt{2}\phi(2t - n) \tag{5.8}$$

$$\psi(t) = h_\phi[n]\sqrt{2}\psi(2t - n) \tag{5.9}$$

For any signal, a discrete wavelet transform in $l^2(Z)^1$ can be approximated by

$$f[n] = \frac{1}{\sqrt{M}}\sum_k W_\phi[j_0, k]\phi_{j_0,k}[n] + \frac{1}{\sqrt{M}}\sum_{j=j_0}^{\infty}\sum_k W_\psi[j, k]\psi_{j,k}[n] \tag{5.10}$$

Here $f[n], \phi_{j_0,k}[n]$ and $\psi_{j,k}[n]$ are discrete functions defined in [0,M - 1], having a total of M points. Because the sets $\{\phi_{j_0,k}[n]\}_{k\in\mathbf{Z}}$ and $\{\psi_{(j,k)\in\mathbf{Z}^2, j\geq j_0}\}$ are orthogonal to each other. We can simply take the inner product to obtain the wavelet coefficients.

$$W_\phi[j_0, k] = \frac{1}{\sqrt{M}}\sum_n f[n]\phi_{j_0,k}[n] \tag{5.11}$$

$$W_\psi[j, k] = \frac{1}{\sqrt{M}}\sum_n f[n]\psi_{j,k}[n] \quad j \geq j_0 \tag{5.12}$$

Equation (5.11) is called approximation coefficient while (5.12) is called detailed coefficients.

These two components show that the approximation coefficient, $W_\phi[j_0, k]$, models a low pass filter and the detailed coefficient,$W_\psi[j_0, k]$, models a high pass filter. It is possible to determine the approximation and detailed coefficients without the

scaling and dilating parameters. The resulting coefficients, called the fast wavelet transform, are a convolution between the wavelet coefficients and a down-sampled version of the next order coefficients. The fast wavelet transform was first postulated in (Mallat, 1989).

$$W_{\phi}[j,k] = h_{\phi}[-n] * W_{\phi}[j+1,n]|_{n=2k,k\geq 0} \tag{5.13}$$

$$W_{\psi}[j_0,k] = h_{\psi}[-n] * W_{\phi}[j+1,n]|_{n=2k,k\geq 0} \tag{5.14}$$

For analysis of the Haar wavelet and the derivation of equations (5.13 and 5.14) see Appendix I.

5.4 Mel filter banks

The Fourier and wavelet transform are general means of extracting information from continuous signals using the frequency domain and in the case of the Wavelet transform using both time and frequency domain. The objective in Machine Learning, however, is to extract patterns from the derived information. In this chapter, in particular, the Mel filter bank and the scatter transform are elaborated on as speech feature extractors. They process high dimensional information obtained from the Fourier and wavelet transform signal processing techniques and reducing the information obtained as lower dimension features. All this aimed towards loss-less encoding of speech signals relevant for speech recognition.

The Mel filter banks form the basis of the Mel Frequency Cepstral Coefficients (MFCCs) described by (Davis and Mermelstein, 1980). MFCCs are state-of-the-art speech feature engineering drivers behind automatic speech recognition acoustic models. Other common speech features used in speech recognition include, Linear Prediction Coefficients (LPCs) and Linear Prediction Cepstral Coefficients (LPCCs),Perceptual Linear Prediction coefficients (PLP), (Dines et al., 2010, McLoughlin, 2009). The following paragraphs describe how the Mel filters are derived.

The Mel scale as described by Stevens et al. (1937) is a perceptual scale which

measures sound frequencies as perceived by human subjects equidistant from a sound source as compared to the actual frequency. This scale is non-linear as the human ear processes sound non-linearly both in frequency as well as amplitude.

For the case of frequency, the human ear can discriminate lower frequencies more accurately than the higher frequencies. The Mel scale model this behaviour by utilising frequency bins. The frequency bin ranges are narrow at low frequencies and become wide in higher frequencies. In the case of the speech signal amplitude, a similar process is observed where the ear discriminates softer sounds better than louder sounds. Generally, sound will be required to be 8 times as loud for significant perception by the ear. While the Mel scales handle the frequency non-linearity in the speech signal, the signal amplitude is linearised during feature extraction by taking the log of the power spectrum of the signal, also known as the cepstral values. Furthermore, using a log scale also allows for a channel normalisation technique that employs cepstral mean subtraction. (Becchetti, 1999).

The minimum frequency number of bins used for the Mel scale is 26 bins. In order to determine the frequency ranges we use the following formula to convert between the Mel scale and the regular frequency scale:

$$M(f) = 1125 \ln(1 + f/700) \tag{5.15}$$

$$M^1(m) = 700 \exp(m/1125)1 \tag{5.16}$$

A simple approximation for the Mel scale is obtained by applying linear scale for the first ten filters and for the first 1kHz of the speech frequency range then applying the following formula for the rest (Becchetti, 1999)(Becchetti, 1999):

$$\Delta_m = 1.2 \times \Delta_{m-1} \tag{5.17}$$

where m is the frequency bin index and Δ_m is the frequency range between the start and end frequencies for the m-th bin. The resulting filters are overlapping filters shown in Figure 5.4. For speech recognition, we compute a statistical value or coefficient for each Mel frequency bin from the inverse discrete fourier transform (IDFT) of the Mel filters. The coefficients are also concatenated with their delta

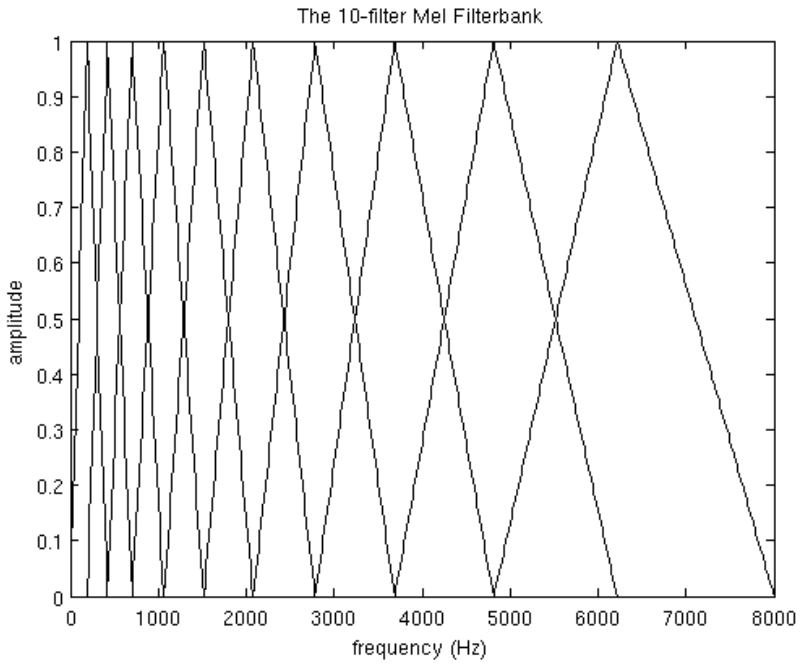

Figure 5.4: Mel filter plot (Lyons, 2012)

and delta-delta values. The delta and delta-delta values are determined from the following equation:

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n}c_{tn})}{2\sum_{n=1}^{N} n^2} \tag{5.18}$$

where c_x is the x-th coefficient and $2n$ is the delta range which is usually $2-4$. The delta values are first order derived coefficients obtained from the original Mel filter coefficients while the delta-delta values are second-order derived coefficients obtained from the first-order derived delta coefficients.

There are two reasons for obtaining the IDFT from the filter banks. The first is that since the bins use overlapping windows, the filter bin outputs tend to be correlated and obtaining the IDFT helps to decorrelate the outputs. Secondly, decorrelated signals optimise algorithm computation efficiency involving matrix operations such that rather than using full co variance matrix, it is much simpler to compute the matrix operations from the matrix diagonal. Also note that for cepstral values obtained from taking the log of the power power spectrum, the discrete cosine transform (DCT) is used to obtain the IDFT. This is as a result of the cepstral values being real and symmetric(Gales et al., 2008).

As an attempt for MFCCs to incorporate dynamic frequency changes of the signal, the deltas and the delta-deltas are obtained from the coefficient computation

in equation 5.18. However, it is worthy to note that only the first 13 of the coefficients and the resulting dynamic coefficients are used as speech features as it is observed that higher frequency dynamic coefficients rather degrade ASR performance (Gales et al., 2008).

5.5 Deep scattering spectrum

Scattering wavelets are interpreted from Mel Frequency Spectral Coefficients (MFCC) which are the Mel filters without applying the IDFT or the DCT.In this section reference is made to (Andén and Mallat, 2011, 2014, Zeghidour et al., 2016) for the scattering wavelet definition.

The Mel scale can be interpreted as dilations of a Wavelet. In the case of the MFSC it lacks the ability to capture non-stationary structures when outside the frame window. Such a filter is constructed by dilating one filter having an octave bandwidth of $1/Q$ as follows

$$\psi_j(t) = a^{-j}\psi(a^{-j}t) \quad | \quad a = 2^{1/Q} \text{ and } j \leq J \tag{5.19}$$

The transfer function of the constructed filter approximately ranges between $2Q\pi - \pi$ and $Q\pi + \pi$. For low frequencies below 2^{-J} a simple low pass filter is employed. Non-linear invariance is induced by applying a contracting modulus operation and the equivalent result is similar to deriving extracting a contracted envelope at different resolutions while filtering out the complex phase information. Therefore, for a signal x we define the zeroth order Scattering transform as follows:

$$|W|x = (x \star \phi(t), |x \star \psi_{j_1}(t)|)_{t \in R, j_1 \in J_1} \tag{5.20}$$

However, the difference between the MFCC and the Deep Scattering Network (DSN) is that the MFCC is a shallow architecture while the DSN is a deep architecture. Therefore, it is observed that time-averaging of the low-pass filter loses information contained in the high frequencies. Nevertheless, since the wavelet transform is invertible, the DSN is able to go deeper by applying the next level scattering operation. The transform is therefore reapplied over $S_0x = x \star \phi(t)$ on the residue

wavelet coefficients $x \star \psi_{\lambda_1}|$ and the process of time averaging to obtain invariant contracted envelopes reapplied. The resulting first order scattering coefficient were obtained from

$$S_1 x(t, \psi_{j_1}) = |x \star \psi_{j_1}| \star \phi(t)) \tag{5.21}$$

It is shown in Andén and Mallat (2014) that if the wavelets ψ_{j_1} have the same frequency resolution as the standard Mel-filters, then the $S_1 x$ coefficients approximate the Mel-filter coefficients. Unlike the Mel-filter banks however, there is a means of regaining discriminating feature information, lost in high frequencies in a DSN. This can be observed when first order wavelets ψ_{j_2} are applied to DSN coefficients $|x \star \phi_{j_1}|$:

$$|W_2||x \star \phi_{j_1}| = (|x \star \psi_{j_1}| \star \phi, ||x \star \psi_{j_1}| \star \psi_{j_2}|)_{j_2 \in J_2}) \tag{5.22}$$

The second order DSN coefficients also requires time averaging to derive local invariance stability, hence the resulting coefficients are averaged again with a low-pass filter ϕ and the final second order DSN scattering parameters are obtained.

$$S_2 x(t, j_1, j_2) = ||x \star \psi_{j_1}| \star \psi_{j_2}| \star \phi(t) \tag{5.23}$$

Figure 5.5 shows how the process of obtaining scatter coefficients can be successively made deeper computing higher-order invariance by retrieving the lost characteristics and thus culminating a deep scattering spectrum. Andén and Mallat (2014) shows that speech signals can be analysed using the first two DSN layers and the coefficients obtained are generally stable to deformation and translation invariant while possessing better discriminating features than MFCCs.

5.6 Chapter Summary

This chapter highlights the characteristics of the Deep Scattering Network that enable it as a candidate rich in pattern recognition discriminators for speech recognition. It is shown also that features required for speech feature invariance is recovered through successive layers of the deep scattering network.

The development of this algorithm in this chapter introduces the Fourier transform as a means of determining frequency contents in a speech wave. While the

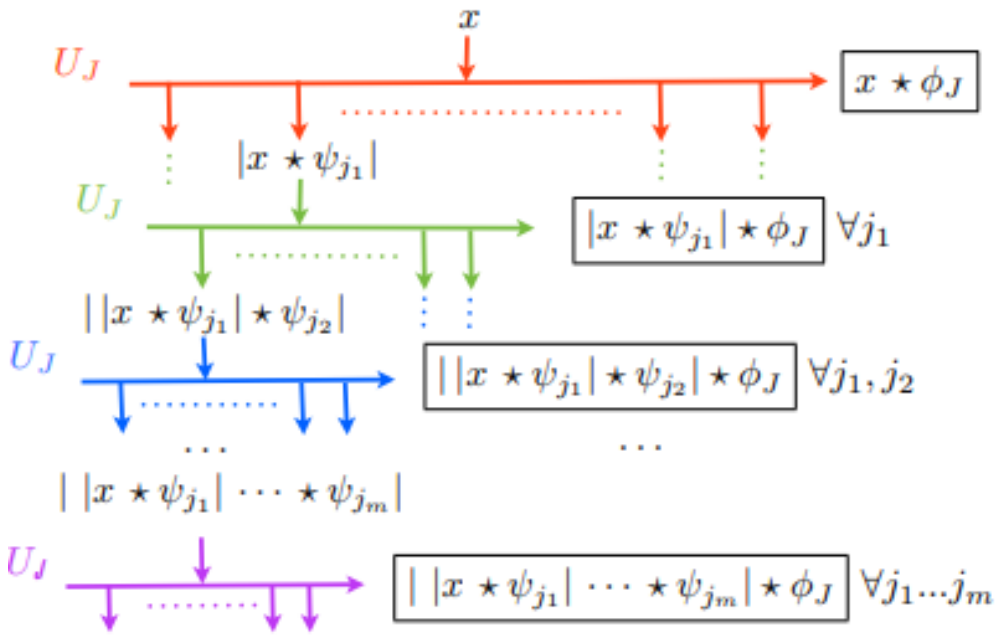

Figure 5.5: Scattering network - 2 layers deep
Andén and Mallat (2011)

Fourier transform has high resolution for frequency, it has no temporal resolution and temporal frequencies or the instantaneous frequency within speech signals therefore cannot be resolved in the Fourier Transform. The Short-time Fourier Transform (STFT) attempts to solve this but there is a trade-off to be made between the temporal and frequency resolution. Better resolutions of time and frequencies however, can be obtained using the Wavelet transform. This chapter also discussed the characteristics of the Wavelet transform that enable better time and frequency filtering.

Finally, MFCCs and Deep Scattering Networks (DSNs) are discussed and compared. It was shown here that through wavelet operations employed by the Deep Scattering Networks frequency resolution lost in MFCC is gained by the DSN and these frequencies contain information relevant for speech invariance. In turn, invariance information is highly useful for better speech discrimination.

Chapter 6

Empirical Analysis 1: Wakirike Language Model

Neural networks have become increasingly popular due to their ability to model non-linear system dynamics. Since their inception, there have been many modifications made to the original design of having linear affine transformations terminated with a nonlinear functions as the means to capture both linear and non-linear features of the target system. In particular, one of such neural network modifications, namely the recurrent neural network, has been shown to overcome the limitation of varying lengths in the inputs and outputs of the classic feed-forward neural network. In addition the RNN is not only able to learn non-linear features of a system but has also been shown to be effective at capturing the patterns in sequential data.

A language model for the Wakirike language is developed in this chapter. This model draws upon the premise that the grammar of a language is expressed in the character sequence pattern ultimately revealed in the words rendered by the character sequences. Therefore, abstract grammar rules can be extracted and learned by a character-based RNN neural network. Specialised implementations of the RNN called the Long Short Term Memory (LSTM) and also the Gated Recurrent Unit (GRU), discussed in chapter 4, are designed to capture patterns over particularly long data sequences and are thus, ideal candidates for generating character sequences while preserving syntactic language rules in the words formed from generated character sequences. These long-term relationships and patterns are learned by the neural network model from the training data it ingests.

6.1 Data Preparation

The Wakirike New Testament Bible served as the source of data for the deep neural network training. There is no readily available soft or on-line copy of the Wakirike new testament bible. As such, the Wakirike New Testament Bible was typed to form a text corpus, giving rise to a complete corpus word size of 668,522 words and a character count of 6,539,176 characters. The data set was then divided into 11 parts. Two parts dedicated for testing and validation and the remaining nine parts were used for training.

The Unicode representations of the character set consisting of letters and punctuation marks are one-hot encoded and batched for sequential input, each batch having a character sequence length of 30 characters.

6.2 GRU Training

The modified LSTM RNN known as the Gated Recurrent Unit (GRU) is employed for the neural network model built in this work, in order to optimise network performance while conserving computation resources. GRUs have been shown to give similar performance to regular LSTMs however, with a lighter system resource footprint (Cho et al., 2014). The GRU RNN used to train the Wakirike text corpus comprised an internal network size of 512 nodes for each layer and was 3 layers deep. Externally, 30 GRUs represented the number of recurrent connections each connection representing a time step bearing contextual for the recurrent input sequence.

To mitigate for over-fitting, due to the multi-layered high-dimensional depth of this neural network, a small learning rate of 0.001 was used. To further marginalise over-fitting the popular and effective dropout method (Srivastava et al., 2014) for regularising deep neural networks was kept at 20% such that only 80% of neural network activations are propagated from one layer to the next, whereas the remaining 20% were randomly zeroed out.

6.3 Output Language Generation

The neural network was trained for 10 epochs and achieved a prediction accuracy of 85% on held-out data. With this GRU character-based language model, it is possible to seed this network with an input character and select from the top-N candidates thus causing the Neural network to generate its own sentences. In this scenario, the network is said to perform language generation by immanently constructing its own sentences. The generated language output from the GRU language model was found to be intelligible and a reflection of the overall context of the training data.

A clever use of this new corpus generated by the GRU language model of this work was to determine a word-based perplexity metric for the GRU neural language model. In this work, the word-based perplexity metric was determined from the output language generated by first estimating the word based perplexity on the training data. The same perplexity calculation was then used on the generated neural language model corpus. The corpus size of the neural language model was made to be equivalent to that of the training data, that is containing 6,539,176 characters. The perplexity calculation was based on a modified Kneser-Key 5-gram model with smoothing (Heafield et al., 2013). The results discussed below showed that the GRU-based RNN model generated a superior model compared to the n-gram model that better matched the training data.

The evaluation of the GRU language model of the Wakirike language was performed using a perplexity measurement metric. The Perplexity metric applies the language model to a test data-set and measures how probable the test data-set is. Perplexity is a relative measure given by the formula:

$$PP(W) = P(w_1, w_2 \dots w_N)^{\frac{1}{N}} \tag{6.1}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{6.2}$$

Where $w_1, \dots, w_N$ are the sequence of words. The language model with the lower relative perplexity score is therefore expected to yield better approximation of the data when applied to unseen data generally.

Table 6.1: Perplexity Calculation results

Language Model	Perplexity	
Held-out data size (characters)	998	99
LSTM RNN	1.6398	1.7622
3-gram with Keysner Soothing and interpolation	1.8046	1.9461

The result of the training of the GRU-Cell Recurrent Neural Network on low-resourced Wakirike Language gave impressive and intelligible results and showed better results when measured with standard n-gram language models. The results showed that it is indeed possible to derive a language model using a GRU-cell RNN on a low resource character sequence corpus for the Wakirike language.

A character based perplexity metric is possible using the negative log likelihood of the character sequence.

$$PP(X) = exp\left\{\frac{\sum_{t=1}^{T} \log P(x_t|x_{1:t1})}{T}\right\} \tag{6.3}$$

However, our base-line language model is a 5-gram word-based language model. Therefore, comparing a word based model to a character based model requires a conversion step. In this work, the conversion step involved using the GRU language model generated a corpus which was rescored by re-estimating with a 5-gram word-based language model

Table 6.1 shows the Results of the Perplexity model of the LSTM Wakirike Language model and an equivalent 5-gram Language model with interpolation and Keysner smoothing (Heafield et al., 2013) for various lengths of the held-out data.

6.4 Chapter Summary

This chapter shows the application of a character-based Gated Recurrent Unit RNN on the low resource language of Wakirike to generate a language model for the Wakirike language. The data-set and preparation and the details of the network were discussed. The output of this model was used to hallucinate the Wakirike language which was then scored against word-based perplexity to obtain a metric against the baseline language model.

It can be inferred that the GRU character-model developed has an improved language model and because it is based on a character-model, which is fine-grained when compared to a word model, it is likely to generalise data better when used in practice and is less biased than a word-based model. This can be observed from the fact that the output corpus produced a larger vocabulary size.

Chapter 7

Empirical Analysis 2: Deep Recurrent Speech Recognition models

This work explores the prospects of deep recurrent end-to-end architectures applied to speech recognition. Complementary aspects of developing speech recognition systems are eliminated by focusing on end-to-end speech units as a two-step process requiring a Connectionist Temporal Character Classification (CTC)(Graves et al., 2006) model and Language Model (LM) rather than a three-step process requiring an Acoustic model(AM), LM and phonetic dictionary. Employing a two-step process rather than a three-step process is particularly desirable for low resource languages as less effort is required developing fewer simplified models.

Earlier in chapter one, deep learning was defined as a type of representational learning whereby different levels of complexity are captured in internal layer-wise encapsulations. It has also been noted that layer-wise stacking of neural and neural network type architectures such as the Restricted Boltzmann Machine (RBM) deep belief networks (DBMs) were used to implement such representations. In this chapter, the end-to-end Bi-directional Recurrent Neural Network model is described. Here, the development of the features using the deep scattering convolution network is first elaborated on. The model parameters and architecture are described and the decoding algorithm are all detailed in sections contained within this chapter.

7.1 Deep Scattering Features

The fast wavelet transform is derived in Chapter 5 from a low pass filter and a high pass filter. The speech features used in this research using a deep scattering network 2 layers deep was created using the wavelet modulus operator comprising a low pass filter and a band pass filter. Hyper parameters of the system included the window period for each sampled sub section, T; The Q-band value for the band pass filter and the number of wavelets J at each scattering layer for the total number of layers, $M = 2$.

The matlab scatnet toolbox (Andén et al., 2014), used to determine the scatter coefficient features for this research, provides optimal values for hyper parameters for audio signal processing into scatter features. In this regime the value for the hyper parameter $T = 512$ samples per window. This corresponds to a window of $50 milliseconds$ for the audio signals sampled at $8000Hz$. For the first scattering layer the Q-band parameter was $Q = 8$ and the second scattering layer took the value $Q = 1$. Finally J is pre-calculated based on the value of T. These after Scat-Net processing produce a feature-vector having 165 dimensions. These feature vectors in turn are used as inputs to the bi-direction neural network model whose architecture is described in succeeding sections.

7.2 CTC-BiRNN Architecture

The core of the system is a bidirectional recurrent neural network (BiRNN) trained to ingest scatter coefficients described in the previous section, in order to generate English text transcriptions. An end-to-end system therefore specifies that utterances x and the corresponding label y be sampled from a training set such that the sample $S = (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ...$ In our end-to-end model, each utterance, $x^{(i)}$ is a processed feature vector consisting of 165 dimensions. Recall, every window passes through a scattering transform to yield an input of vector of $p = 165$ features; consequently, $x^{(i)}_{t,p}$ denotes the p-th feature in a scatter transform at time t.

GPU training of the speech model architecture developed above was conducted using Mozilla deepspeech (moz, 2019a) CTC bi-directional RNN implementation along with the accompanying Mozilla Common voice dataset (moz, 2019b). The

Common Voice Dataset project consists of voice samples in short recordings approximately 4 seconds each. The complete dataset is about 250 hours of recording divided into training, test and development subsets. The BiRNN, given the input sequence, x, outputs a sequence of probabilities $y_t = P(c_t|x)$, where $c_t \in a, b, c, \ldots, z, space, apostrophe, blank$.

The actual architecture of our core Bi-RNN is similar to the deepspeech system described in Hannun et al. (2014a). This structure constitutes 5 hidden layers and one output layer. The first three layers are regular DNNs followed by a bi-directional recurrent layer. As such, the output of the first three layers are computed by:

$$h_t^{(l)} = g(W^{(l)}h_t^{(l1)} + b^{(l)}) \tag{7.1}$$

$g(\cdot) = min\{max\{0, z\}, 20\}$ is the clipped rectified linear unit and $W^{(l)}, b^{(l)}$ are weight matrix and bias parameters for layer as described in sections 4.2.1 and 4.3 respectively.

It was shown in chapter 4 the recurrent layer comprise a forward and backward RNNs whose equations are repeated here for reference

$$h_t^{(f)} = g(W^{(4)}h_t^{(3)} + W_r^{(f)}h_{t1}^{(f)} + b^{(4)}) \tag{7.2}$$

$$h_t^{(b)} = g(W^{(4)}h_t^{(3)} + W_r^{(b)}h_{t+1}^{(b)} + b^{(4)}) \tag{7.3}$$

Consequently, $h^{(f)}$ is the sequential computation from $t = 1$ to $t = T^{(i)}$ for the i-th utterance and $h^{(b)}$ is the reverse computation from $t = T^{(i)}$ to $t = 1$. In addition the output from layer five is summarily given as the combined outputs from the recurrent layer:

$$h^{(5)} = g(W^{(5)}h^{(4)} + b^{(5)}) \tag{7.4}$$

where $h^{(4)} = h^{(f)} + h^{(b)}$. The output of the Bi-RNN on layer 6 is a standard soft-max layer that outputs a predicted character over probabilities for each time slice t and

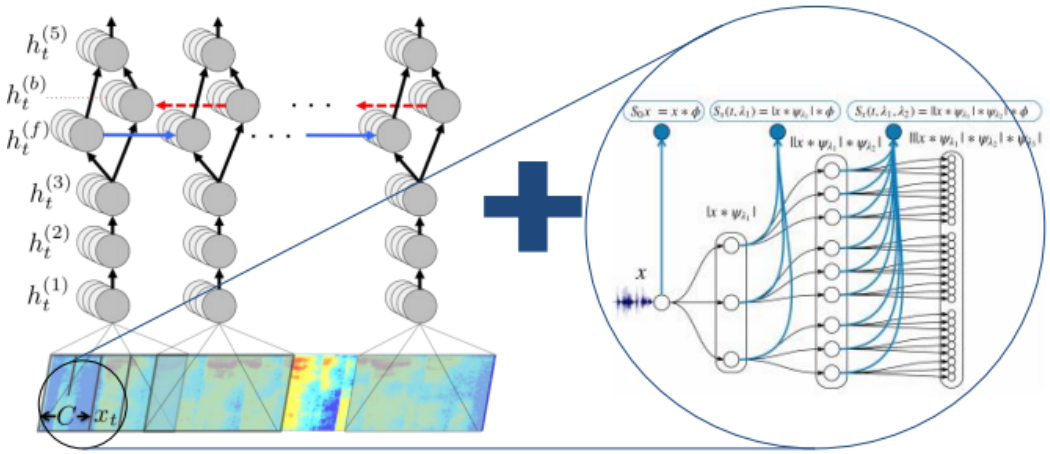

Figure 7.1: Deep scattering Bi-RNN Model

character k in the alphabet:

$$h_{t,k}^{(6)} = \hat{y}_{t,k} \equiv P(c_t = k \mid x) = \frac{\exp\left((W^{(6)}h_t^{(5)})_k + b_k^{(6)}\right)}{\sum_j \exp\left((W^{(6)}h_t^{(5)})_j + b_j^{(6)}\right)}) \tag{7.5}$$

$b_k^{(6)}$ takes on the -th bias and $(W^{(6)}h_t^{(5)})_k$ is the matrix product of the k-th element. The error of the outputs are then computed using the CTC loss function Graves (2014) as described in chapter 4. A summary of our model is illustrated in Figure 7.1.

7.3 CTC Decoding

In chapter three the CTC loss function algorithm was established as being able to maximise the probability of two cases. The first case of transiting to a blank and the second case of transiting to a non blank. In this section, this concept is used to enable decoding of the network output from posterior distribution output to character sequences which can be measured against a reference transcription using either character error rate (CER) or word error rate (WER).

Recall, all the output symbols are in the alphabet Σ and augmented with the blank symbol. The posterior output of the CTC network is the probability of the symbol given the speech feature input $p(c|x_t)$ at time t for $t = 1, \ldots, T$ and T is the length of the input sequence. Also recall two further sets of probabilities also being

maintained by the model are the probability of a blank character p_b and that of a non blank character p_{nb}.

Several strategies have been employed to obtain a translation string from the output of the deep neural network. The prefix beam search employed by the CTC decoder of this research is derived from an initial greedy approximation, where at each time step determine the argument that maximises the probability $p(c|x_t)$ at each time step. Let $C = (c_1, \ldots, c_T$ be the character string then, the greedy approach has

$$c_t = arg \max_{c \in \Sigma} p(c|x_t) \tag{7.6}$$

However, this simple approximation is unable to collapse repeating sequences and remove blank symbols. In addition, the approximation is unable to include the constraint of a lexicon or language model.

The prefix beam search algorithm Hannun et al. (2014b) adopted in this work incorporates a language model derived from a lexicon in addition to keeping track of the various likelihoods used for decoding. For the language model constraint, the transcription W is recovered from acoustic input X at time t by choosing the word which maximising the posterior probability:

$$W_i = arg \max_{W_i \in \Sigma_W} p_{net}(W; X) p_{lm}(W) \tag{7.7}$$

In equation 7.7, the Bayes product of language model prior p_{lm} and the network output p_{net} are utilised to maximise the probability of a particular character-word sequence in the lexicon given by Σ_W. The overall calculation used to derive the final posterior distribution includes word insertion factors (α and β) used to balance the highly constrained n-gram language model.

The second strategy adopted by the prefix beam search which improves the decoding algorithm is the beam search strategy. With this approach, the search maintains all possible paths; however, it retains only k number paths which maximise the output sequence probability. Improvements gained with this method are seen when certain maximal paths are made obsolete owing to new information derived from the multiple paths in being maintained in memory.

The recursive prefix beam search algorithm illustrated in Figure 7.2 attempts to

find the string formulated in equation 7.7. Two sets prefixes A_{prev} and A_{nxet} are initialised, such that at A_{nxet} maintains the prefixes in the current time-step while A_{prev} maintains only k-prefixes from the previous time-step. Note that at the end of each time step A_{prev} is updated with only -most probable prefixes from A_{nxet}. Therefore while, A_{nxet} contains all the possible new paths from based on A_{prev} as a Cartesian product of $A_{prev} \times \Sigma \in \mathcal{Z}^k \times \mathcal{Z}^{|\Sigma|}$ where $|\Sigma|$ is the length of Σ. The probabilities of each prefix obtained at each time step are the sum of the probability of non-blank plus the probability of a blank symbol.

At every time step and for every prefix ℓ currently in A_{prev}, a character from the alphabet Σ is presented to the prefix. The prefix is only extended only when the presented symbol is not a blank or a space. A_{nxet} and A_{prev} maintain a list of active prefixes at the previous time step and proposed prefixes at the next time step respectively, The prefix probability is given by multiplying the word insertion term by the sum of the blank and non-blank symbol probabilities.

$$p(\ell|x_{1:t}) = (p_{nb}(\ell|x_{1:t}) + p_b(\ell|x_{1:t}))|W(\ell)|^{\beta} \tag{7.8}$$

$W(\cdot)$ is obtained by segmenting all the characters in the sequence with the space-character symbol and truncating any characters trailing the set of words in the sequence. The prefix distribution however varies slightly depending on network output character being presented.

ℓ_{end} is the variable representing the last symbol in the prefix sequence in A_{prev}. If the symbol presented is the same as ℓ_{end} then the probability of a non-blank symbol,$p_{nb} = 0$. If the symbol being presented is blank then we do not extend the prefix. Finally, if the symbol being presented is a space then we invoke the language model as follows

$$p(\ell^+|x_{1:t}) = p(W(\ell^+)|W(\ell))^{\alpha}(p_{nb}(\ell|x_{1:t}) + p_b(\ell|x_{1:t}))|W(\ell)|^{\beta} \tag{7.9}$$

Note that $p(W(\ell^+)|W(\ell))$ is set to 0 if the current word $W(\ell^+)$ is not in the lexicon. This becomes a constraint to enforce all character strings to consist only of words in the lexicon. Furthermore, $p(W(\ell^+)|W(\ell))$ is extended to include all the character sequences representing number of words considered by the n-gram

language model by constituting the last $n-1$ words in character sequence $W(\ell)$.

7.4 Model Hyper parameters

The hidden layer matrix for each layer comprised 1024 hidden units (6.6M free parameters). The weights are initialised from a uniform random distribution having a standard deviation of 0.046875. The Adam optimisation algorithm (Kingma and Ba, 2014) was used with initial learning rate of, and a momentum of 0.95 was deployed to optimise the learning rate.

The network was trained for a total of five to fifty epochs over the training set for experiments conducted. The training time for Python GPU implementation is shown in Table 7.1. For decoding with prefix search we use a beam size of 200 and cross-validated with a held-out set to find optimal settings for the parameters α and β. Figure 7.4 shows word error rates for various GPU configurations and audio data-set sizes.

7.5 Model Baseline

The study by Hannun et al. (2014b) reported successful character error rate (CER) using deep neural network (DNN), recurrent deep neural network with only forward temporal connections (RDNN), and also bi-directional recurrent neural networks (BRDNN). The models used in this their study had 5 hidden layers having either 1,824 or 2,048 hidden units in each hidden layer. For a baseline, the model produced by the Mozilla DeepSpeech team was adopted. This model had a similar architecture with 5 hidden units and 2048 hidden units and was trained on the Librespeech corpus and the common voice data corpora (moz, 2019a, Panayotov et al., 2015).

Word Error Rates by this model were optimised after 75 epochs, learning rate of 0.0001 and a dropout rate of 15%. In addition, the language model hyper parameters for alpha and beta were 0.75 and 1.85 respectively. This achieved 8% WER. This model was developed using MFCC features of the training corpus.

Table 7.1: GPU Experiments

Experiment	Hours of speech	Total training time	Estimated training
1. 2xGPU 10GB RAM	1	7 days	Completed
2. 2xGPU 10GB RAM	10	355 days	Completed
3. 5xGPU 15GB RAM	10	17 hours	Completed
4. 5xGPU 15GB RAM	40	12 days	Completed
5. 1xCPU 16GB RAM	20	4+ days	70 days
6. 1xGPU 2GB RAM	20	17+ days	100 days
7. 1xCPU 16GB RAM	20	4+ days	70 days
8. 1xCPU 16GB RAM	20	4+ days	70 days
9. ESPNet 1xGPU 2GB	1	1 hour	Completed

7.6 Results

Experiments were carried out on different GPU configurations. A set of experiments was performed a GPU configuration consisting of 2 GPUs having a total of 10 gigabytes of memory. The second set of experiments was carried out on a GPU configuration comprising 5 GPUs having a total of 15 gigabytes of memory. Experiments were also performed using single GPUs having 2GB and another single GPU having 8GB. For each GPU configuration experiments were carried out on varying-size subsets of the common voice corpus being utilised. The various GPU configurations along with the training times are shown in Table 7.1.

In addition to the GPU configuration, experiments involving CPU and multi-node training were carried out. Although quite a number of configuration did not reach a stopping condition, the multi node configurations which made use of Tensorflow distributed feature in particularly required regular user-intervention, and as such, was short-lived. In table 7.1, the first four configurations trained to saturation. For these results, the training loss reduced significantly once the data was increased to ten hours of training. However word error rates (WER) only showed improvement on the 40 hours data set. This seems to indicate that a threshold of about 40 hours is required for the model to begin to converge for a Large Scale Vocabulary Continuous Speech Recognition (LSVCSR) system

Table 7.2: Summary of GPU Experiments

Experiment	Hours of speech	Corpus	Metric	Score
1. 2xGPU 10GB RAM	1	CV LVCSR	WER(%)	100+
2. 2xGPU 10GB RAM	10	CV LVCSR	WER(%)	100+
3. 5xGPU 15GB RAM	10	CV LVCSR	WER(%)	100
4. 5xGPU 15GB RAM	40	CV LVCSR	WER(%)	87
5. ESPNet 1xGPU 2GB	1	AN4 MVCSR	CER(%)	9.5

7.7 Preliminary ESPNet Experiment

Preliminary experiments were carried out using the ESPNet (Watanabe et al., 2018) an overview of which is described in Chapter 3 and detailed some more in this section and Chapter **??**. A much smaller audio corpus guaranteed to converge however was used for these experiments. The AN4 (alphanumeric) corpus by Carnegie Mellon University (Acero, 1990), is a small vocabulary speech corpus having only 948 training utterances and 140 test utterances.

The corpus utterances are 16-bit linearly sampled at 16kHz, each recording made with near-field microphone quality. The compressed tar file comes with a variety of audio formats including raw wav format, the NIST sphere format and those already encoded as Mel cepstral coefficients.

Experiments were carried out using ESPNet default parameters which included those for character based-Recurrent Neural Network language model RNN-LM, multi-channel feature input and multi-objective learning using both CTC-Transformer and Attention-Transducer networks.

7.7.1 ESPNet Speech model architecture, parameters and results

The end-to-end architecture at the core of ESPNet is the CTC-Transformer+Attention Transducer model. Together these two architectures achieve joint multi-objective speech training and decoding. The CTC-Transformer model is based on a Bi-RNN similar to what is obtainable in the DeepSpeech model. The attention transducer model is further explained in Chapter 8. There are up to 11 variants of Attention networks implemented in ESPNet, however, the results of the ESPNet experiment

performed was determined from the model described in Chorowski et al. (2015). Moreover, the multi-objective training was performed with equal weights on both the CTC-transformer and the Attention-Transducer. Finally the system was trained for 20 epochs only.

With this minimal default setting, the test set had a final recognition score of 9.5% character error rate (CER). The next Chapter discusses how the baseline can be scaled and remodelled for integrating scattering features.

7.8 Chapter Summary

In this chapter the details of the novel structure having the end-to-end deep bi-RNN architecture and deep scattering features were elaborated on. The architecture which follows a five-layer structure consisting of a feed-forward neural network in the first three layers and the last two consisting of recurrent structures flowing in two different directions. The network is then fed in with a 165-dimension feature vector containing deep-scattering encoding derived from a sampled raw audio file.

The results showed that the training of the model was moving towards a very slow convergence as indicated by the slow decrements in training loss. However, we speculate that on the complete data-set, the model will not only converge but show improvements in word error rates. Already this is seen from results from preliminary baseline experiments with ESPNet. With an advanced architecture, yet having the CTC-Transformer Bi-RNN at its core, the ESPNet baseline model yielded a competitive 9.5% CER using multi-channel features derived from features integrated with 83 log MFCC feature vector.

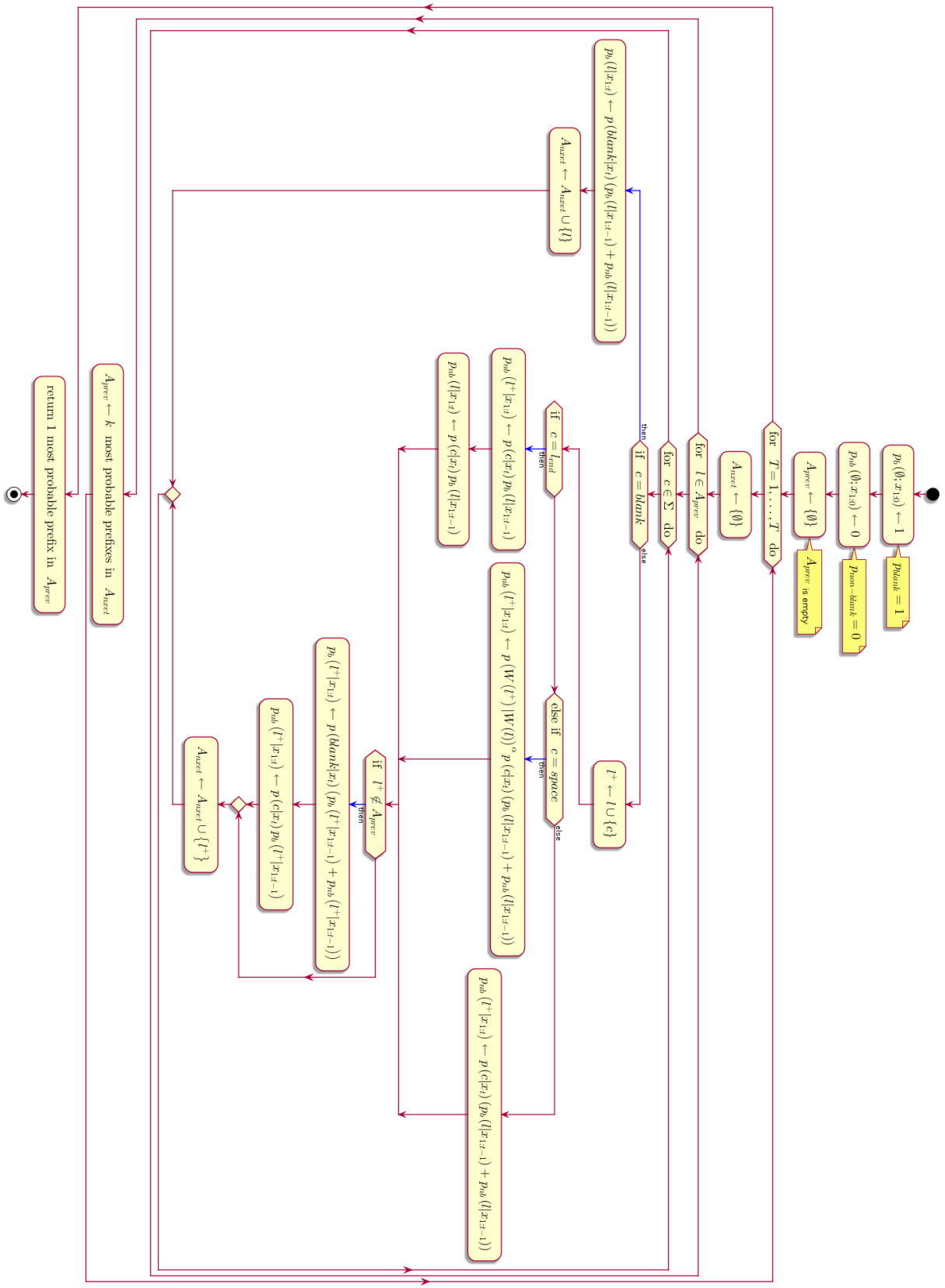

Figure 7.2: Prefix beam search algorithm

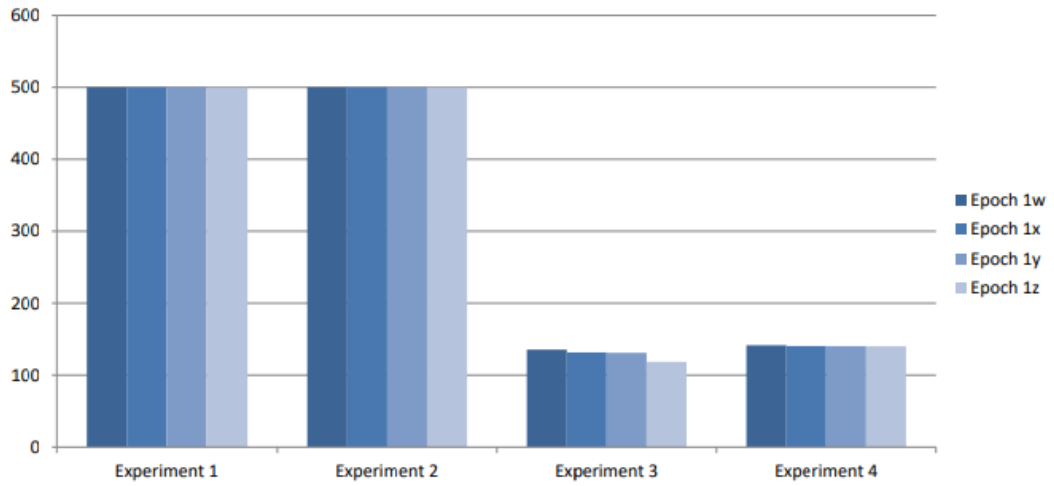

Figure 7.3: Training Loss, where $w < x < y < z$ are taken arbitrarily across the total number of epochs

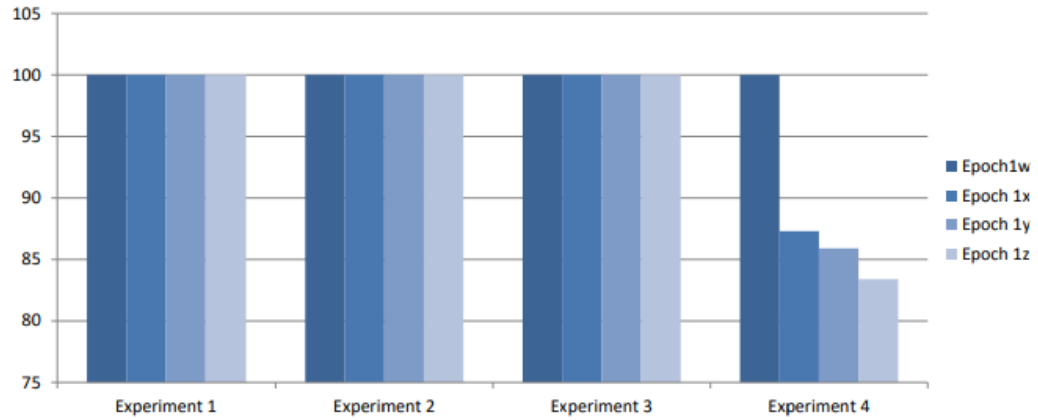

Figure 7.4: WER, where wxyz are taken arbitrarily across the total number of epochs

Chapter 8

Conclusion and Future Work

The advancement of Machine Learning has a direct impact on the development of more efficient speech recognition algorithms and at the same time the advancement of speech recognition helps with the improvement of Machine Learning algorithms, as in general, the methods used in Machine Learning usually are directly transferable to speech processing and vice-versa. This mutual relationship implies that speech recognition is a blossoming research field because there is a tremendous amount of work being done in the Machine Learning community. Particularly in the area of deep learning and neural networks, there is quite a vast array of neural network solutions that have been applied or are yet to be applied to speech recognition. Two models worthy of mentioning are Generative Adversarial Networks (GANs) and Attention-based models.

8.1 Generative adversarial networks (GAN)

GANs consists of two Networks working as adversaries to one another. The first, being a generative network, generates content. The second network is a discriminative network intended to determine the accuracy of the first generative network. Hence the generative network is generating output less distinguishable for the discriminator while the discriminator uses output from the generator to improve the ability to discriminate output from the generator with the original source data.

GAN networks can have applications where the generative network consists of a speech synthesis network and the discriminating network is a speech recogniser.

However successive training of these two networks from a data-resource perspective would require an immense amount of data resources for expected performances.

8.2 Attention-based Models

The objective of attention-based networks highlighted by Vaswani et al. (2017) is to reduce sequential computation while attaining hidden representation across arbitrary lengths of sequential input. Mechanisms which have been deployed to achieve this includes a combination of convolutional and recurrent schemes (Gehring et al., 2017, Kaiser and Bengio, 2016, Kalchbrenner et al., 2016). Vaswani et al. (2017) introduces a transduction model known as a Transformer based on self attention network with the ability to compute long term dependencies while eliminating sequence aligned RNN and convolutional architectures.

Self attention is a network that intrinsically reduces the need for intensive resource training. Vaswani et al. (2017) reports that state of the art BLEU score of 41.0 having used a small fraction of training resources. While GANs might not be attractive for low resource speech recognition, they still remain an important tool for verification of the output of other networks. At the same time self attention networks can help to reduce the resource requirements of GANs when used within the context of a GAN.

As a study to further this thesis, these networks are likely candidates for network training using scatter features as input discriminatory functions. Attention based networks as a means reduce training resources required, while GANs can be used as a means to generate training data.

8.3 Joint Training with ESPNet

Chapter 7 has already seen promising results based on preliminary experiments using the ESPNet model. This model does fulfil major objectives outlined by this research. The inclusion of Attention-based models and the warp-CTC decoder implemented in ESPNet ensures faster convergence and ultimately faster time to train, in addition to the end-to-end architecture presented by ESPNet. The next step required for a further study is an integration of scattering features into ESPNet. This scattering

features implementation is the subject of a paper publication immediately sought after based on this work. The features of ESPNet include state-of-the-art end-to-end architectures including variants of Attention-Transducers and CTC-decoders. Using a weighting function one can control how much bias either the CTC-Transform or the Attention-Transducer will get during training. The joint training helps to improve robustness as well as achieve fast convergence.

$$\mathcal{L} = \alpha\mathcal{L}^{ctc} + (1-\alpha)L^{att} \tag{8.1}$$

At the same time joint decoding of labels is integrated with the character based RNN-language modelling. The log probability of the RNNLM-integrated decoding of character labels is as follows

$$\log p(y_n|y_{1:n1}, \mathbf{h}_{1:T^{\text{`}}}) = \log p^{hyp}(y_n|y_{1:n1}, \mathbf{h}_{1:T^{\text{`}}}) + \beta \log p^{lm}(y_n|y_{1:n1}) \tag{8.2}$$

Where joint decoding, $\log p^{hyp}(y_n|y_{1:n1}, \mathbf{h}_{1:T^{\text{`}}})$ is given by

$$\log p^{hyp}(y_n|y_{1:n1}, \mathbf{h}_{1:T^{\text{`}}}) = \alpha \log p^{ctc}(y_n|y_{1:n-1}, \mathbf{h}_{1:T'}) + (1-\alpha) \log p^{att}(y_n|y_{1:n-1}, \mathbf{h}_{1:T'}) \tag{8.3}$$

Furthermore, multi-channel training integrates noise robust and far-field speech recognition tasks which can accommodate joint 83-dimension MFCC and scatter transform training in addition to speech enhancement features such as beam forming and STFT masking Ochiai et al. (2017). Both single and multi-channel scatter transform features are proposed as subjects in the future paper publication.

8.4 Model Pretraining

The major setback this work suffered was from a reasonable time for training. This work therefore recommends that speech models or indeed artificial intelligence models should be trainable within a maximum of 30 days and shouldn't generally exceed 20 days training.

An area of neural network training optimisation not developed in this report is that of layer-wise greedy pre-training. In this process, rather than train the deep

neural network structure all in one stage, the network layers are successively added and trained layer by layer, one layer at a time (Goodfellow et al., 2016). The intuition behind this is that this makes the layers saturate much faster as the previous layer has already been saturated before the new layer is being added.

This layer-wise pretraining procedure is thought to speed up training, than training when done with the fully connected network and there have been a few different approaches to pretraining in for deep neural network architectures for speech recognition.

Hendrycks et al. (2019) introduces an advanced pretraining method in which existing models are retrained in a Generative Adversarial Network (GAN) fashion in order to optimise performance and model robustness. This is an instance where GANs are being deployed in speech recognition. This method however is not likely going to help improve model training convergence time.

Another method described in (Ramachandran et al., 2016, **?**) uses a knowledge transfer mechanism where hidden layers in an already existing related network is re-trained with new extended layers to complete training in the new domain. The effectiveness of such a transfer method will be measured of the how the two domains correlate with one another. It therefore, would be logical to conclude that the more the domains correlate the faster the pretraining model is likely to converge faster.

Finally, Wang et al. (2019) proposes a Tandem Connectionist Encoding Network (TCEN) for bridging the gap for fine tuning CTC-Transformer networks along with pretraining of Attention-transducers.

As a further study, amongst other techniques, the most viable method for this study is to investigate the knowledge transfer mechanism by approaching the feature engineering problem as a latent space analysis problem. Given that during the process of mapping acoustic speech sequences to the MFCC reduced the latent space from a high dimension to a low dimension. It is reasonable therefore to hypothesize that training from hidden layers of an MFCC deep RNN would converge faster than weights initialised through generic means.

8.5 Conclusion

End-to-end discriminative neural network speech models have now become a well established method in Automatic Speech Recognition.

Bi-directional Recurrent neural network (Bi-RNN) end-to-end system, is augmented by features derived from a deep scattering network as opposed to the standard Mel Frequency Cepstral Coefficients(MFCC) features used in state of the art acoustic models. These specialised deep scattering features, consumed by the Bi-RNN, model a light-weight convolution network. This work shows that it is possible to build a speech model from a combination of deep scattering features and a Bi-RNN. There has been no record of deep scattering features being used in end-to-end bi-RNN speech models as far as we are aware.

This thesis shows that Deep Scattering features derived from wavelet filter operations on audio data produce viable candidates for end-to-end training of Automatic speech recognition models.

The objective of this research to facilitate fast and efficient speech recognition is achieved using the ESPNet system. The major advantage of this system is that in addition to robust end-to-end models is the intrinsic integration of a character-based language model enabling it to satisfy both low-resource challenge criteria of top level word and sentence modelling through the character-based language model and the sub-word and acoustic modelling of input features.

Appendix I - Haar Wavelet

A fundamental purpose of analysing functions such as the Fourier and wavelet functions are the reconstruction of signals from it's decomposition. Certain criteria or properties are therefore required for analysis functions.

In Chapter 5.1, the orthogonal properties of the Fourier transform equations was introduced. In the case of wavelets the following properties ensue. In addition to orthogonal properties, wavelets are required to perform localised analysis of a function. Hence, unlike their Fourier counterparts, they need to be bounded in time. It is also seen that when the energy contained within the wavelet bases sum to zero (sometimes normalised to 1)i.e.

$$E = \int_{-\infty}^{\infty} |x|^2 dt = ||x(t)||^2 = 0 \tag{4}$$

Then, such wavelet bases are orthonormal and the fundamental or scaling equation forms a recurrence relation which is a solution to the dilation equation as follows:

$$\phi(t) = \sum_k c_k \phi(2t - k) \tag{5}$$

Where $\phi(2t - k)$ is a contracted version of $\phi(t)$ shifted along the time axis by an integer step k and factored by an associated coefficient c_k. At the same time it is also observed that it is possible to setup this recurrence relation to becoming dyadic such that the sum of coefficients, c_k equals 2, i.e. $\Sigma_k c_k = 2$. Haar, wavelets constitute the simplest of this family of wavelets.

The mother wavelet of the Haar wavelet has only two coefficients $c_0 = c_1 = 1$ and is given by:

$$\psi(t) = \phi(2t) + \phi(2t - 1) \tag{6}$$

Observe here that $c_0 + c_1 = 2$, i.e. dyadic. The solution to this recurrence equation and the resulting plot is given in Figure 1.

Through multi-resolution analysis, the following reconstruction of the Haar wavelet is derived:

$$\phi_{j,k}[n] = 2^{j/2} \phi[2^j n - k] \tag{7}$$

The parameter j, controls the resolution of the signal reconstruction and the following wavelets and function representation are given in Figure 2

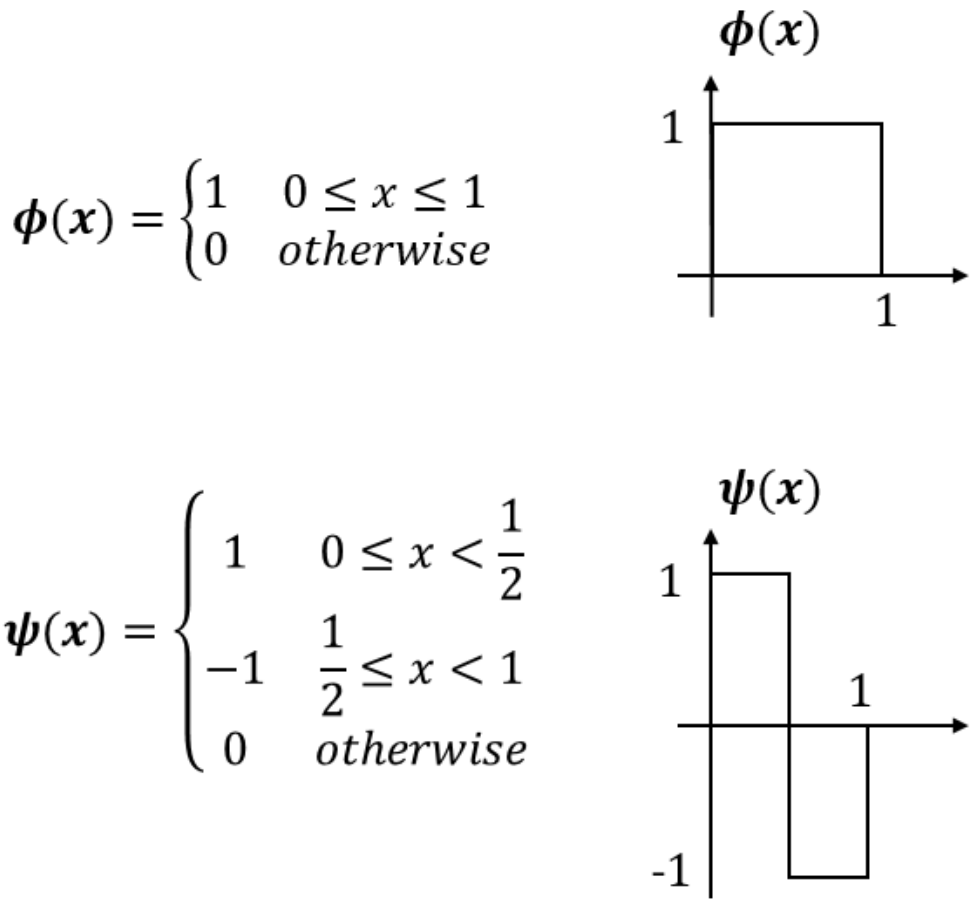

Figure 1: Haar wavelet

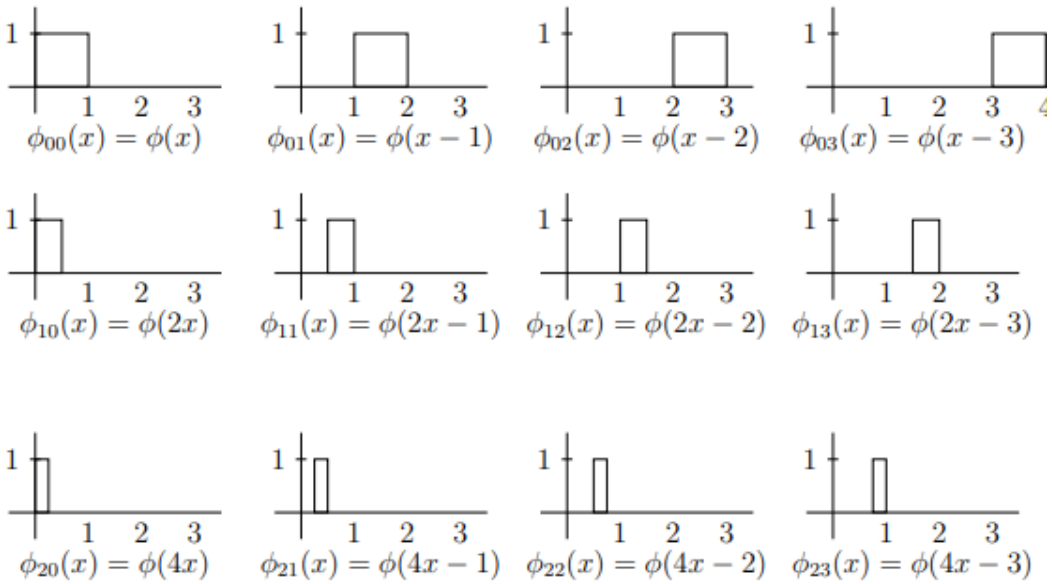

Figure 2: Multi resolution analysis of Haar wavelets

Appendix II - Gabor and Morlet Wavelets

Gabor Wavelet filter

The 1D-Gabor wavelet is defined spatially by

$$\psi(x) = exp\left(-\frac{x^2}{2\sigma^2} + i\xi x\right) \tag{8}$$

The Fourier-transform is

$$\hat{\psi}(\omega) = exp\left(-\frac{\sigma^2(\omega-\xi)^2}{2}\right) \tag{9}$$

It's value bounded at 0 is $\hat{\psi(0)} = exp(-\sigma^2\xi^2/2)$, so we have

$$\xi\sigma = \sqrt{-2\log(\hat{\psi}(0))} \tag{10}$$

The wavelets are therefore computed as

$$\psi_j = \frac{1}{a^j\sigma}\left(\frac{x}{a^j}\right) \tag{11}$$

Where a is the scale factor. if we call τ, the value of the two Gabor where the plots intersect, we have, as in figure 3.

$$\frac{\xi}{a} + \frac{g^{-1}(\tau)}{a\sigma} = \xi - \frac{g^{-1}(\tau)}{\sigma} \tag{12}$$

where $g(x) = exp(-x^2/2)$ i.e. $g^{-1}(\tau) = \sqrt{-2\log(\tau)}$ So we have

$$\xi\sigma = \sqrt{-2\log(\tau)}\frac{a+1}{a-1} \tag{13}$$

The value of ξ is fixed by the fact that we need the frequency information so we set

$$\xi = 3\pi/4 \tag{14}$$

Equations (10 and 13) show that the choice of σ is trade off between two antagonist requirement on the wavelet

i. a zero-mean: σ should be large.

ii. *tau* should be large therefore σ should be small

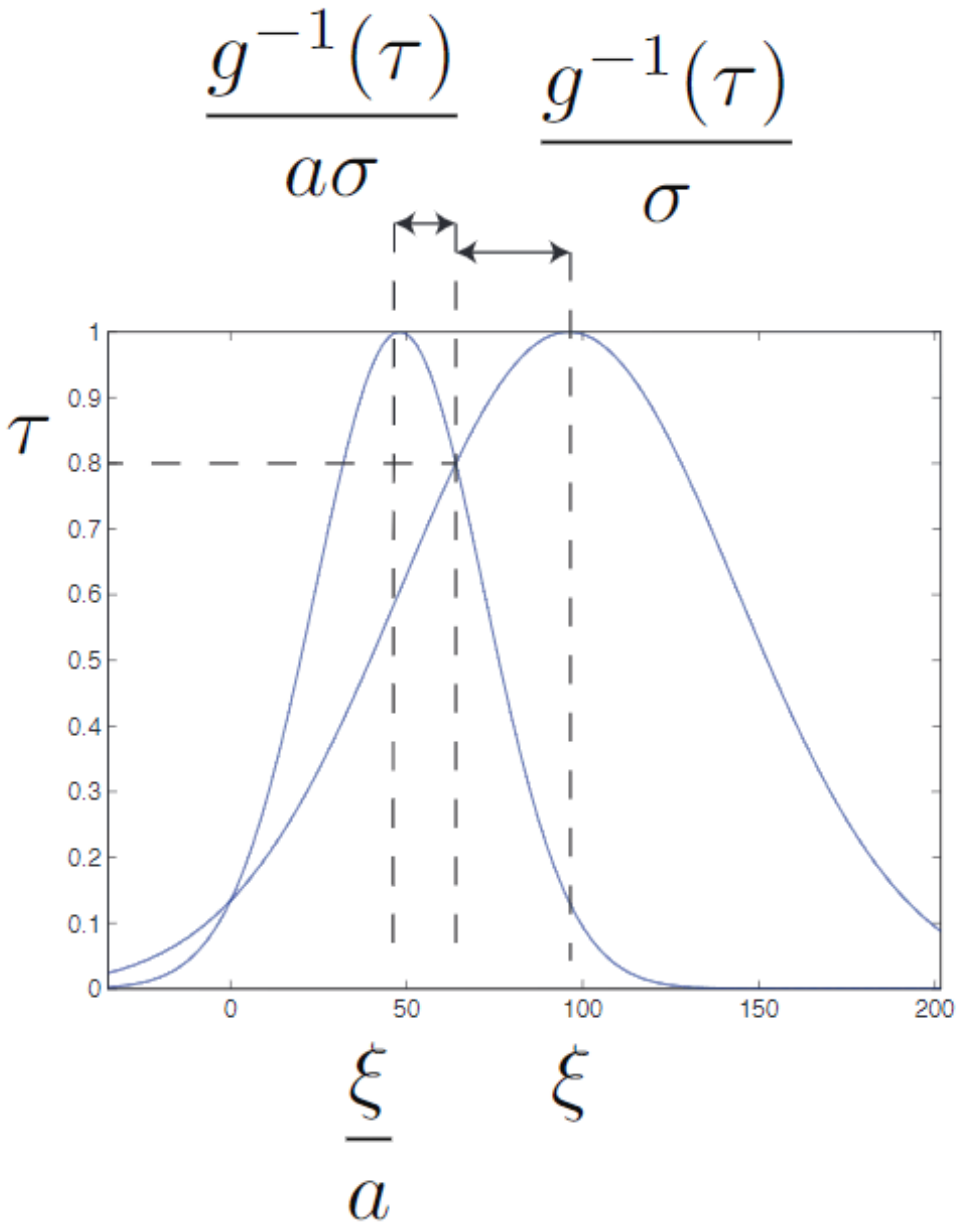

Figure 3: Fourier transform of adjacent scale Gabor wavelet. τ has been set to 0.8

With these requirements met we see that

$$\hat{\psi}(0) = \tau^{\frac{(a+1)^2}{(a-1)^2}} \tag{15}$$

So we can see that these two requirements are compatible as we take more and more bands per octave.

In the implementation, the only parameter about the wavelet that the user can set is $\tau_{as} = \tau$, where 'as' stands for adjacent scales. The implementation of parameters is the following

i. The value of ξ is set to $3\pi/4$.

ii. The user chooses values for τ, a and j.

iii. The value of σ is computed with

$$\sigma = \frac{\sqrt{-2\log(\tau)}}{\xi}\frac{a+1}{a-1} \tag{16}$$

There is another parameter called τ_{lc}. 'lc' stands for low-coarse. It controls the value crossing between the low pass filters ψ_j (a Gaussian) and the coarsest scale wavelet ψ_{J-1} and this parameter determines the value of the bandwidth of the low pas filter.

Morlet wavelet filter

Morlet filters are modified Gabor filters that have zero-mean: The idea is to subtract a Gabor, its envelop times a constant so that the results has zero mean:

$$\phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)(\exp(i\xi x) - exp(-\sigma^{22}/2)) \tag{17}$$

It's Fourier transform is

$$\hat{\psi}(\omega) = \omega \exp\left(-\frac{\sigma^2(\omega-\xi)^2}{2}\right) - \exp\left(-\frac{\sigma^2(\omega+\xi)^2}{2}\right)\Big) \tag{18}$$

And we can see it has zero $\hat{\psi}(0) = 0$

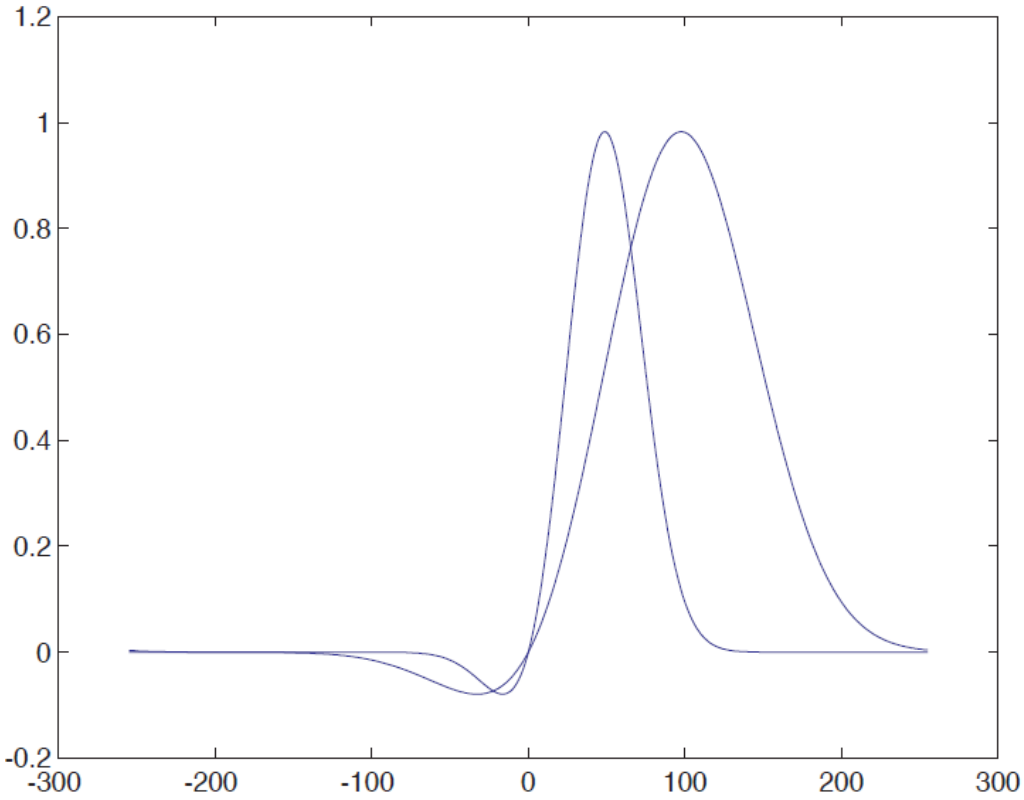

Figure 4: Fourier transform of adjacent scale Gabor wavelet. τ has been set to 0.8

Appendix III - Matlab listing for scattering network

```
function scatter()
    %UNTITLED Summary of this function goes here
    %   Detailed explanation goes here
    T = readtable('data/cv-valid-dev.xlsx','ReadRowNames',true);
    F=table2cell(T(:,{'scatterc','wav_filename'}));
    all_files=size(F,1);
    ofm='hh:mm:ss';
    ifm='dd-mmm-yy HH:MM:SS.FFF';
    tic;
    for i = 1:all_files
      wav_file=strjoin(F(i,2));
      dss_file=strjoin(F(i,1));
      if exist(wav_file,'file')>0
        if exist(dss_file,'file')==0
                st = transpose(scatter_audio(wav_file));
                csvwrite(dss_file,st);
        end
      else
        fprintf('\nNot found:%s',wav_file);
      end

      pg=i/all_files*100;
      ts=datestr(now,ifm);
      tv=toc;
      d=duration(seconds(tv),'Format',ofm);
      pc=(all_files/i*tv)-tv;
      eta=duration(seconds(pc),'Format',ofm);

      if mod(i,500)==0 || i==1 || i==10 || i==100
        fileID = fopen('log/dss180625.log','w+');
        s=sprintf('\n%s: processing file  %s',ts,wav_file);
        fprintf(fileID,'%s',s);
        fprintf('%s',s);
        s=sprintf('\n%s : processing %d of %d files  %3.2f%% complete.. ti
        fprintf(fileID,'%s',s);
        fprintf('%s',s);
        fclose(fileID);
```

```
        end
    end
end

function st= scatter_audio(inputArg1)
    y=audioread(inputArg1);
    N=length(y);
    T=2^9;
    filt_opt=default_filter_options('audio',T);
    Wop=wavelet_factory_1d(N,filt_opt);
    S=scat(y,Wop);
    S=renorm_scat(S);
    S=log_scat(S);
    st=format_scat(S);
end
```

Appendix IV - Code listing for Section 3.2.5 - Sample TensorFlow client code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" A one-hidden-layer-MLP MNIST-classifier. """
from __future__ import absolute_import from __future__ import division
from __future__ import print_function
# Import the training data (MNIST)
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

# Possibly download and extract the MNIST data set.
# Retrieve the labels as one-hot-encoded vectors.
mnist = input_data.read_data_sets("/tmp/mnist",one_hot=True)

# Create a new graph
graph = tf.Graph()

# Set our graph as the one to add nodes to
with graph.as_default():

  # Placeholder for input examples (None = variable dimension)
  examples = tf.placeholder(shape=[None, 784], dtype=tf.float32)
  # Placeholder for labels
  labels = tf.placeholder(shape=[None, 10], dtype=tf.float32)

  weights = tf.Variable(tf.truncated_normal(shape=[784, 10],
    stddev=0.1))
  bias = tf.Variable(tf.constant(0.1, shape=[10]))

  # Apply an affine transformation to the input features
  logits = tf.matmul(examples, weights)
    + bias estimates = tf.nn.softmax(logits)

  # Compute the cross-entropy
  cross_entropy = -tf.reduce_sum(labels * tf.log(estimates),
    reduction_indices=[1])
  # And finally the loss
  loss = tf.reduce_mean(cross_entropy)
```

```
  # Create a gradient-descent optimizer that minimizes the loss.
  # We choose a learning rate of 0.01
  optimizer = tf.train.GradientDescentOptimizer(0.5).minimize(loss)

  # Find the indices where the predictions were correct
  correct_predictions = tf.equal(
    tf.argmax(estimates, dimension=1),
    tf.argmax(labels, dimension=1))
  accuracy = tf.reduce_mean(
    tf.cast(correct_predictions, tf.float32))

with tf.Session(graph=graph) as session:
  tf.initialize_all_variables().run()
  for step in range(1001):
    example_batch, label_batch = mnist.train.next_batch(100)
    feed_dict = {examples: example_batch, labels: label_batch}
    if step % 100 == 0:
      _, loss_value, accuracy_value =
        session.run([optimizer, loss, accuracy], feed_dict=feed_dict)
      print("Loss at time {0}: {1}".format(step, loss_value))
      print("Accuracy at time {0}:{1}".format(step, accuracy_value))
    else:
      optimizer.run(feed_dict)
```

Bibliography

Continuous wavelet transform, 2015.

2019a. URL `https://github.com/mozilla/DeepSpeech#common-voice-training-data`.

Mozilla deepspeech, 2019b. URL `https://voice.mozilla.org/en`.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Martín Abadi, Michael Isard, and Derek G Murray. A computational model for tensorflow: an introduction. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 1–7. ACM, 2017.

Alejandro Acero. Acoustical and environmental robustness in automatic speech recognition. In *Proc. of ICASSP*, 1990.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

J Andén, L Sifre, S Mallat, M Kapoko, V Lostanlen, and E Oyallon. Scatnet (v0. 2). *Computer Software. Available: http://www.di.ens.fr/data/software/scatnet/. [Accessed: December 10, 2013]*, 0.2, 2014.

Joakim Andén and Stéphane Mallat. Multiscale scattering for audio classification. In *ISMIR*, pages 657–662. Miami, FL, 2011.

Joakim Andén and Stéphane Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62(16):4114–4128, 2014.

Claudio Becchetti and Lucio P. Ricotti. *Speech recognition: theory and C++ implementation*. Wiley, New York, 1998. ISBN 0471977306;9780471977308;.

L Becchetti. The behaviour of financial time series: stylised features, theoretical interpretations and proposals for hidden markov model applications. *Speech recognition. Theory and C++ implementation*, 1999.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

Laurent Besacier, Etienne Barnard, Alexey Karpov, and Tanja Schultz. Automatic speech recognition for under-resourced languages: A survey. *Speech Communication*, 56:85–100, 2014a.

Laurent Besacier, Etienne Barnard, Alexey Karpov, and Tanja Schultz. Introduction to the special issue on processing under-resourced languages. 2014b.

Mikael Boden. A guide to recurrent neural networks and backpropagation. *the Dallas project*, 2002.

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.9919&rep=rep1&type=pdf`.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. Wav2letter: an end-to-end convnet-based speech recognition system. *arXiv preprint arXiv:1609.03193*, 2016.

Jack D Cowan. Discussion: Mcculloch-pitts and related neural nets from 1943 to 1989. *Bulletin of mathematical biology*, 52(1-2):73–97, 1990.

George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2012.

Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

Li Deng and Xiao Li. Machine learning paradigms for speech recognition: An overview. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(5): 1060–1089, 2013.

Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

John Dines, Junichi Yamagishi, and Simon King. Measuring the gap between hmm-based asr and tts. *IEEE Journal of Selected Topics in Signal Processing*, 4(6): 1046–1058, 2010.

Sadaoki Furui. Speaker-independent isolated word recognition using dynamic features of speech spectrum. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(1):52–59, 1986.

Mark Gales, Steve Young, et al. The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304, 2008.

Mark JF Gales, Kate M Knill, Anton Ragni, and Shakti P Rath. Speech recognition and keyword spotting for low-resource languages: Babel project research at cued. In *Spoken Language Technologies for Under-Resourced Languages*, 2014.

Mark John Francis Gales, Shinji Watanabe, and Eric Fosler-Lussier. Structured discriminative models for speech recognition: An overview. *IEEE Signal Processing Magazine*, 29(6):70–81, 2012.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR.org, 2017.

Arnab Ghoshal, Pawel Swietojanski, and Steve Renals. Multilingual training of deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7319–7323. IEEE, 2013.

Jean-Philippe Goldman. Easyalign: an automatic phonetic alignment tool under praat. 2011.

Peter Goldsborough. A tour of tensorflow. *arXiv preprint arXiv:1610.01178*, 2016.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2014.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772, 2014.

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

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

Frantisek Grezl and Petr Fousek. Optimizing bottle-neck features for lvcsr. In *ICASSP*, volume 8, pages 4729–4732, 2008.

Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014a.

Awni Y Hannun, Andrew L Maas, Daniel Jurafsky, and Andrew Y Ng. First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *arXiv preprint arXiv:1408.2873*, 2014b.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 690–696, Sofia, Bulgaria, August 2013. URL `https://kheafield.com/papers/edinburgh/estimate_paper.pdf`.

Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. *arXiv preprint arXiv:1901.09960*, 2019.

Hynek Hermansky. Perceptual linear predictive (plp) analysis of speech. *The Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

Hynek Hermansky and Nelson Morgan. Rasta processing of speech. *IEEE transactions on speech and audio processing*, 2(4):578–589, 1994.

Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976. doi: 10.1109/PROC.1976.10159.

Bing-Hwang Juang and S. Furui. Automatic recognition and understanding of spoken language - a first step toward natural human-machine communication. *Proceedings of the IEEE*, 88(8):1142–1165, 2000. doi: 10.1109/5.880077. URL `http://ieeexplore.ieee.org/document/880077`.

Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In *Advances in Neural Information Processing Systems*, pages 3781–3789, 2016.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Herman Kamper, Aren Jansen, and Sharon Goldwater. Unsupervised word segmentation and lexicon discovery using acoustic word embeddings. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):669–679, 2016.

Nikhil Ketkar. Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer, 2017.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Julius Kunze, Louis Kirsch, Ilia Kurenkov, Andreas Krug, Jens Johannsmeier, and Sebastian Stober. Transfer learning for speech recognition on a budget. *arXiv preprint arXiv:1706.00290*, 2017.

Paul Lamere, Philip Kwok, Evandro Gouvêa, Bhiksha Raj, Rita Singh, William Walker, Manfred Warmuth, and Peter Wolf. The cmu sphinx-4 speech recognition system, 2003.

Julia A Lasserre, Christopher M Bishop, and Thomas P Minka. Principled hybrids of generative and discriminative models. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 87–94. IEEE, 2006.

Thang Luong, Richard Socher, and Christopher D Manning. Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113, 2013.

J Lyons. Mel frequency cepstral coefficient (mfcc) tutorial, 2012. URL `http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/`.

Stéphane Mallat. Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, 374(2065):20150203, 2016.

Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693, 1989.

Ian McLoughlin. *Applied speech and audio processing: with Matlab examples*. Cambridge University Press, 2009.

Tomáš Mikolov, Anoop Deoras, Stefan Kombrink, Lukáš Burget, and Jan Černockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

Abdel-rahman Mohamed, George Dahl, and Geoffrey Hinton. Deep belief networks for phone recognition. In *Nips workshop on deep learning for speech recognition and related applications*, volume 1, page 39. Vancouver, Canada, 2009.

Abdel-rahman Mohamed, George E Dahl, Geoffrey Hinton, et al. Acoustic modeling using deep belief networks. *IEEE Trans. Audio, Speech & Language Processing*, 20(1):14–22, 2012.

Scott Novotney and Richard Schwartz. Analysis of low-resource acoustic model self-training. In *Tenth Annual Conference of the International Speech Communication Association*, 2009.

Jay F Nunamaker Jr, Minder Chen, and Titus DM Purdin. Systems development in information systems research. *Journal of management information systems*, 7 (3):89–106, 1990.

Tsubasa Ochiai, Shinji Watanabe, Takaaki Hori, and John R Hershey. Multichannel end-to-end speech recognition. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2632–2641. JMLR. org, 2017.

Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. URL `http://www.numpy.org/`. [Online; accessed ¡today¿].

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

Douglas B Paul and Janet M Baker. The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics, 1992.

Vijayaditya Peddinti, TaraN Sainath, Shay Maymon, Bhuvana Ramabhadran, David Nahamoo, and Vaibhava Goel. Deep scattering spectrum with deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 210–214. IEEE, 2014.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162.

Joseph Picone. Fundamentals of speech recognition: A short course. *Institute for Signal and Information Processing, Mississippi State University*, 1996.

Daniel Povey, Lukáš Burget, Mohit Agarwal, Pinar Akyazi, Feng Kai, Arnab Ghoshal, Ondřej Glembek, Nagendra Goel, Martin Karafiát, Ariya Rastrow, et al. The subspace gaussian mixture model—a structured model for speech recognition. *Computer Speech & Language*, 25(2):404–439, 2011a.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number CONF. IEEE Signal Processing Society, 2011b.

Anton Ragni and Mark JF Gales. Automatic speech recognition system development in the" wild". In *Interspeech*, pages 2217–2221, 2018.

Anton Ragni, Katherine Mary Knill, Shakti P Rath, and Mark John Gales. Data augmentation for low resource languages. 2014.

Prajit Ramachandran, Peter J Liu, and Quoc V Le. Unsupervised pretraining for sequence to sequence learning. *arXiv preprint arXiv:1611.02683*, 2016.

A. Rosenberg, K. Audhkhasi, A. Sethy, B. Ramabhadran, and M. Picheny. End-to-end speech recognition and keyword search on low-resource languages. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5280–5284, March 2017. doi: 10.1109/ICASSP.2017.7953164.

Charles Ogan D. S. *Okrika: A kingdom of the Niger Delta*. Onyoma Research Publications, Port Harcourt, Rivers State, Nigeria, 1 edition, 2008.

Tara N Sainath, Vijayaditya Peddinti, Brian Kingsbury, Petr Fousek, Bhuvana Ramabhadran, and David Nahamoo. Deep scattering spectra with deep neural networks for lvcsr tasks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL `http://arxiv.org/abs/1402.1128`.

George Saon, Hong-Kwang J Kuo, Steven Rennie, and Michael Picheny. The ibm 2015 english conversational telephone speech recognition system. *arXiv preprint arXiv:1505.05899*, 2015.

Ralf Schluter and Hermann Ney. Model-based mce bound to the true bayes' error. *IEEE Signal Processing Letters*, 8(5):131–133, 2001.

Laurent Sifre and Stéphane Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1233–1240, 2013.

Laurent Sifre and Stéphane Mallat. Rigid-motion scattering for image classification. *Ph. D. dissertation*, 2014.

Gary F. Simons and Charles D. Fennig. Ethnologue: Languages of the world, twenty-first edition., 2018. URL `http://www.ethnologue.com`.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Adriana Stan, Yoshitaka Mamiya, Junichi Yamagishi, Peter Bell, Oliver Watts, Robert AJ Clark, and Simon King. Alisa: An automatic lightly supervised speech segmentation and alignment tool. *Computer Speech & Language*, 35:116–133, 2016.

Stanley Smith Stevens, John Volkmann, and Edwin B Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Maarten Versteegh, Roland Thiolliere, Thomas Schatz, Xuan Nga Cao, Xavier Anguera, Aren Jansen, and Emmanuel Dupoux. The zero resource speech challenge 2015. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

Ngoc Thang Vu and Tanja Schultz. Multilingual multilayer perceptron for rapid language adaptation between and across language families. In *Interspeech*, pages 515–519, 2013.

Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. 2004.

Chengyi Wang, Yu Wu, Shujie Liu, Zhenglu Yang, and Ming Zhou. Bridging the gap between pre-training and fine-tuning for end-to-end speech translation. *arXiv preprint arXiv:1909.07575*, 2019.

Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai. Espnet: End-to-end speech processing toolkit. In *Interspeech*, pages 2207–2211, 2018. doi: 10.21437/Interspeech.2018-1456. URL `http://dx.doi.org/10.21437/Interspeech.2018-1456`.

Shinji (. e. Watanabe and Jen-Tzung Chien. *Bayesian speech and language processing*. Cambridge University Press, Cambridge, 2015. ISBN 1107055571;9781107055575;.

PC Woodland and Daniel Povey. Large scale discriminative training for speech recognition. In *ASR2000-Automatic Speech Recognition: Challenges for the new Millenium ISCA Tutorial and Research Workshop (ITRW)*, 2000.

Ping Xu and Pascale Fung. Cross-lingual language modeling for low-resource speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 21 (6):1134–1144, 2013.

Steve Young. A review of large-vocabulary continuous-speech. *IEEE Signal Processing Magazine*, 13(5):45, 1996. doi: 10.1109/79.536824.

Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. The htk book. *Cambridge university engineering department*, 3:175, 2002.

Dong Yu and Li Deng. *AUTOMATIC SPEECH RECOGNITION.* Springer, 2016.

Dong Yu, Li Deng, and George Dahl. Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition. In *Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

Neil Zeghidour, Gabriel Synnaeve, Maarten Versteegh, and Emmanuel Dupoux. A deep scattering spectrum—deep siamese network pipeline for unsupervised acoustic modeling. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4965–4969. IEEE, 2016.