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, 2021

Abstract

Automatic Speech Recognition (ASR) has made major leaps in its advancement largely due to two different machine learning models: Hidden Markov Models (HMMs) and Deep Neural Networks (DNNs). State-of-the art results have been achieved by combining these two disparate methods to form a hybrid system. This also requires that various components of the speech recognizer be trained independently based on a probabilistic noisy channel model. Although this HMM-DNN hybrid ASR method has been successful in recent studies, the independent development of the individual components used in hybrid HMM-DNN models makes ASR development fragile and expensive in terms of time-to-develop the various components and their associated sub-systems. The resulting trade-off is that ASR systems are difficult to develop and use especially for new applications and languages.

The alternative approach, known as the end-to-end paradigm, makes use of a single deep neural-network architecture used to encapsulate as many as possible sub-components of speech recognition as a single process. In the so-called end-to-end paradigm, latent variables of sub-components are subsumed by the neural network sub-architectures and the associated parameters. The end-to-end paradigm gains of a simplified ASR-development process again are traded for higher internal model complexity and computational resources needed to train the end-to-end models.

This research focuses on taking advantage of the end-to-end model ASR development gains for new and low-resource languages. Using a specialised light weight convolution-like neural network called the deep scattering network (DSN) to replace the input layer of the end-to-end model, our objective was to measure the performance of the end-to-end model using these augmented speech features while checking to see if the light-weight, wavelet-based architecture brought about any improvements for low resource Speech recognition in particular.

The results showed that it is possible to use this compact strategy for speech pattern recognition by deploying deep scattering network features with higher dimensional vectors when compared to traditional speech features. With Word Error Rates of 26.8% and 76.7% for SVCSR and LVCSR respective tasks, the ASR system metrics fell few WER points short of their respective baselines. In addition, training times tended to be longer when compared to their respective baselines and therefore had no significant improvement for low resource speech recognition training.

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, Ibinabo Alamina, 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

Abstract 2
Dedication 2
Acknowledgements 3
Copyright Statement 4
List of Figures 10
List of Tables 13
List of Algorithms 14
Acronyms 15
List of Symbols 19

1 Introduction **19**
1.1 ASR As a Machine Learning problem 20
1.2 Generative-Discriminative Speech Models disambiguation 21
1.3 Low Resource Languages 23
1.4 The Wakirike Language 24
1.5 Research aim and objectives 24
1.5.1 Research Question 26
1.6 Main Contribution to knowledge 26
1.7 Scope of the study 27
1.8 Thesis outline 28
1.9 Chapter Summary 28

2 Literature Review **30**
2.1 Speech Recognition Overview 30
2.1.1 HMM-based Generative speech model 31
2.1.2 Challenges of Speech Recognition 32

2.1.3 Challenges of low speech recognition . . . 33
2.2 Low Resource Speech Recognition . . . 34
2.2.1 Low Resource language modelling . . . 35
2.2.2 Low Resource Acoustic and speech modelling . . . 38
2.3 Groundwork for low resource end-to-end speech modelling . . . 39
2.3.1 Deep speech . . . 39
2.3.2 Speech Recognition on a low budget . . . 40
2.3.3 Adding a Scattering layer . . . 41
2.4 Chapter Summary . . . 42

3 Methods, Models and Systems 44
3.1 Assumptions . . . 45
3.2 Speech Processing software and tools . . . 45
3.2.1 CMUSphinx . . . 47
3.2.2 Kaldi . . . 51
3.2.3 Mozilla DeepSpeech . . . 54
3.2.4 Matlab and ScatNet toolbox . . . 57
3.2.5 TensorFlow . . . 64
3.2.6 Choregraphe . . . 70
3.2.7 Alisa . . . 70
3.3 Pilot Studies . . . 73
3.3.1 Auto-correlation Experiments . . . 73
3.3.2 Experiments with Nao robot . . . 77
3.3.3 Digit Speech Recognition and Alignment Experiments . . . 77
3.4 Sequence-to-sequence Model Experiments . . . 78
3.4.1 Procedure for designing sequence-to-sequence RNN models . . 80
3.4.2 Sequence-to-sequence character-to-diacritically-labelled-character model . . . 81
3.4.3 Sequence-to-sequence Grapheme-to-Phoneme (G2P) model . . 82
3.4.4 GRU language model for Wakirike language based on TensorFlow . . . 84
3.4.5 Bi-Directional LSTM-based end-to-end speech model . . . 84
3.4.6 ESP-Net Experiments . . . 85

3.5 Method of evaluation 85
3.6 Chapter Summary 86

4 Background 1: Recurrent Neural Networks in Speech Recognition 88
4.1 Neural network architecture 88
4.1.1 Multi-layer Perceptron (MLP) 89
4.1.2 Sigmoid and soft-max Activation Function 90
4.1.3 Back propagation algorithm (backprop) 91
4.1.4 Gradient Descent 92
4.2 RNN, LSTM and GRU Networks 93
4.2.1 Deep Neural Networks (DNNs) 93
4.2.2 Recurrent Neural Networks 95
4.2.3 Back propagation through time (BPTT) algorithm 96
4.2.4 LSTMs and GRUs 100
4.3 Deep speech architecture 102
4.3.1 Connectionist Temporal Classification (CTC) 104
4.3.2 Forward-backward algorithm 106
4.3.3 CTC Loss function 109
4.4 Attention Mechanism 110
4.5 Chapter Summary 112

5 Background 2: Deep Scattering network 114
5.1 Fourier transform 115
5.2 Wavelet transform 116
5.3 Discrete and Fast wavelet transform 117
5.4 Mel filter banks 119
5.5 Deep scattering spectrum 122
5.6 Chapter Summary 123

6 Empirical Analysis 1: Wakirike Language Model 125
6.1 General Considerations for Sequence-to-sequence modelling 125
6.1.1 Selection of Sequence Model 126
6.1.2 Selection of RNN-architectures for sequence modelling 127
6.1.3 Neural Network geometry 128

6.1.4 Network Saturation Parameters . . . 128
6.1.5 Regularisation measure . . . 130
6.2 Data Preparation . . . 130
6.3 GRU RNN Architecture . . . 131
6.4 Language Model Training Experiments . . . 133
6.5 Output Language Model and Language Generation . . . 133
6.6 Discussion . . . 135
6.7 Chapter Summary . . . 137

7 Empirical Analysis 2: Deep Recurrent Speech Recognition models 138
7.1 Deep Scattering Features . . . 139
7.2 CTC-BiRNN Architecture . . . 140
7.2.1 CTC Decoding . . . 142
7.2.2 Model Hyper parameters . . . 144
7.3 Summary of Bi-RNN Experiment Design . . . 145
7.4 BiRNN with Attention Transducer end-to-end Architecture . . . 146
7.5 Summary of birnn with Attention Transducer Experiment Design . . . 147
7.6 Speech Model Baselines . . . 148
7.7 Speech Model Simulations . . . 148
7.7.1 Bi-RNN-only end-to-end model Experiments . . . 148
7.7.2 Bi-RNN with Attention Transducer Experiments . . . 149
7.8 Model Results Interpretation . . . 151
7.8.1 Bi-RNN-only experiment discussion . . . 151
7.8.2 Bi-RNN with Transducer and attention mechanism experiment discussion . . . 152
7.9 Chapter Summary . . . 153

8 Conclusion and Future Work 156
8.1 Discussion of Research Output models . . . 157
8.1.1 Main contribution to knowledge . . . 158
8.1.2 Summary of goals achieved in this work . . . 158
8.2 Limitations of the study . . . 159
8.3 Directions for future study . . . 160

8.3.1 Generative adversarial networks (GAN) 160
8.3.2 Attention-based Models 161
8.3.3 Model Pre-training 161
8.4 Conclusion 163

Appendix I - Haar wavelet **163**

Appendix II - Gabor and Morlet wavelet filters **164**

Appendix III - Scatter Transform implementation **168**

Appendix IV - Sample TensorFlow Client code **171**

Appendix V - Wakirike Phonetic dictionary **173**

List of Figures

2.1 HMM Generative Model showing six states (1 to 6); state transition probabilities, a_{12} to a_{56}; and, emission probabilities, b_2 to b_5 for observations $\mathbf{o}_1$ to $\mathbf{o}_6$ (Young et al., 2002) 32

2.2 Automatic Speech Recognition Pipeline showing main components of Feature Extraction, Acoustic Model, Language Model and Lexicon Model (Besacier, Barnard, Karpov, & Schultz, 2014a). 34

3.1 CMU Sphinx4 recogniser system. The core modules: `FrontEnd`, `Decoder` and `Linguist` coloured in grey. 47

3.2 The Kaldi Architecture(Povey, Ghoshal, et al., 2011) has four layers. At the deepest layer are the external libraries, followed by the C++ back-end library. At the top: are the C++ executables called by shell scripts 52

3.3 Scatter transform wavelet filter plots of various dilations of J 60

3.4 Unnormalised scattergram 62

3.5 Log normalised scattergram 62

3.6 Sample TensorFlow computation graphs(Goldsborough, 2016) showing a simple addition operation on the left and a more involved sequence on the right comprising a dot operation followed by an addition and then a sigmoid operation 65

3.7 Tensorflow graph with backprop nodes (Goldsborough, 2016). The forward propagation is on the left (a) and; the Forward propagation with back propagation (b) is on the right 67

3.8 Alisa iterative architecture (Stan et al., 2016) involving acoustic model, skip-network recognition and confidence level determination 72

3.9 Original waveform input from 3-second utterance for auto-correlation 75
3.10 (a) Positive values of original waveform (b) Filtered values (c) Peak counter (d) Trough counter 75
3.11 (a) Positive values of original waveform (b) Isolated Segment plot (c) Correlation plot of (a) against (b) plot. counter 76
3.12 Relationship Types and their neural network interpretation. (Karpathy, 2015) 79
3.13 Diacritic symbol generator training data sequences 82
3.14 Diacritic symbol generator model training loss 83

4.1 Neuron cell (Landahl, McCulloch, & Pitts, 1943) where x_i are inputs and w_i are input weights 89
4.2 Perceptron algorithm having multiple neuron cells 89
4.3 An LSTM Cell showing Input, output and forget gates (Graves, Jaitly, & Mohamed, 2013) 101
4.4 Beam Search Lattice Structure showing forward (left) and backward (right) paths (Graves, Fernández, Gomez, & Schmidhuber, 2006) . . . 108
4.5 Attention mechanism is centred around a time-distributed dense operation that determines similarity measure between current decoding sequence hidden input and all input sequence outputs. 111

5.1 Fourier Equation 115
5.2 Sample Spectrogram from an arbitrary input signal showing frequency-power content of the signal 116
5.3 Time frequency tiling for (a) Fourier Transform (b) Short-time Fourier Transform (STFT) (c) Wavelet transform 117
5.4 Mel filter plot showing overlapping frequency bins (Lyons, 2012) . . . 121
5.5 Scattering network - 2 layers deep 124

6.1 Wakirike Language model training Loss curves for (a) 3-Layer GRU and (b) Single-Layer RNN 134

7.1 Deep scattering Speech Model architecture reveals the 5-hidden layer Bi-RNN $h_t^{(1)}$ to $h_t^{(5)}$ being trained by DSN features. 142

7.2 Prefix beam search algorithm 154
7.3 Bi-RNN-only Experiments Error curve, where $w < x < y < z$ are taken arbitrarily across the total number of epochs 155
7.4 birnn with Attention Transducer Training Loss: (a) and (d) with 250-dimension scatter transform features; (b) and (c) with 83-dimension Log-Mel features 155

1 Haar wavelet 165
2 Multi resolution analysis of Haar wavelets 165
3 Fourier transform of adjacent scale Gabor wavelet. τ has been set to 0.8 167
4 Fourier transform of adjacent scale Gabor wavelet. τ has been set to 0.8 169

List of Tables

6.1 Language Models comparison 133
6.2 Language Models sample generation 136

7.1 Bi-RNN-only Experiments 149
7.2 Bi-RNN-only Experiments Summary 149
7.3 Bi-RNN with attention and transducer Experiments 150
7.4 Bi-RNN with attention and transducer Experiments Summary 150

List of Algorithms

1 DNN training algorithm 95
2 RNN training algorithm 99

List of Abbreviations

AM	Acoustic Model
ASR	Automatic Speech Recognition
Bi-LSTM	Bi-directional Long Short-Term Memory cell RNN
Bi-RNN	Bi-directional Recurrent Neural Network
BLEU	BiLingual Evaluation Understudy
BLSTM	Bi-directional LSTM
CFG	Context Free Grammar
CMU	Carnegie Mellon University
CMVN	acrlongcmvn
CMN	Cepstral Mean Normalisation
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
CUDA	Compute Unified Device Architecture
CV	Common Voice speech corpus
DBN	Deep Belief Network
DCT	Discrete Cosine Transform
DNN	Deep Neural Network
DNNs	Deep Neural Networks
DSN	Deep Scattering Network
DTW	Dynamic Time Warping
FSG	Finite-State Grammar
ESPNet	End-to-End Speech Neural Network Toolkit
FSGs	Finite-State Grammars
FST	Finite-State Transducer
G2P	Grapheme-to-Phoneme
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HLDA	Heteroscedastic Linear Discriminant Analysis
HMM	Hidden Markov Model
IDFT	Inverse Discrete Cosine Transform
LDA	Linear Discriminant Analysis
LLC	Language Learning Companion
LM	Language Model
LPC	Linear Predictive Coding
LSTM	Long Short-Term Memory
MFCC	Mel Frequency Cepstral Coefficients

Acronyms contd.

MIMO	Multiple Input Multiple Output
MISO	Multiple Input Single Output
ML	Machine Learning
MLLT	Maximum Likelihood Linear Transformation
MLP	Multi Layer Perceptron
MLPs	Multi Layer Perceptrons
OOV	Out-Of-Vocabulary
PLP	Perceptual Linear Prediction
RASTA	RelAtive SpecTrAl
RBM	Restricted Boltzmann Machine
RELU	Rectified Linear Unit
RNN	Recurrent Neural Network
RNNs	Recurrent Neural Networks
RNN-LM	Recurrent Neural Network Language Model
SGMM	Sub-space Gaussian Mixture Model
SGD	Stochastic Gradient Descent
SIMO	Single Input Multiple Output
SISO	Single Input Single Output
STC	Semi-Tied Co-variance matrix
TTS	Text-to-Speech
VAD	Voice Activity Detection
VTLN	Vocal Tract Length Normalisation
WER	Word Error Rate
WFST	Weighted Finite State Transducer

Symbols

Symbol	Meaning
$\mathbf{O}$	vector sequence of natural observations
$\mathbf{o}_i$	observation vector at time=i
$\mathbf{w}$	sequence of words recognised
w_i	word recognised at time=i
a_{ij}	probability between state i and j
b_j	probability of output observation
M	Hidden Markov Model
H	Entropy of a language
PP	Perplexity of a language
$\mathbf{x}$	observation vector in terms of input features
C	output machine learning class prediction
$\mathbf{W}$	matrix of neural network weight parameters
E	Error margin
$\nabla_{\mathbf{w}}$	derivative with respect to weights
σ	neural network non-linear function
$\mathbf{h}$	matrix of hidden state weights
$\mathbf{l}$	sequence of output class labels
$\mathbf{y}$	sequence of output classes
$y_{t,p}$	CTC output probabilities
π	CTC output character sequence
α	forward probability per time-step
β	backward probability per time-step
$\mathcal{L}$	Cross entropy loss for a sequence of input data
z	sequence of output characters
a_k	Fourier series coefficient
x	continuous signal input
C	Continuous wavelet transform
a	continuous wavelet scaling factor
b	continuous wavelet shifting factor
ϕ	orthonormal bases (scaling) function
ψ	shifting (translation) function
$\hat{\psi}$	mother wavelet
h	Haar coefficients
M	Mel scale function
δ_m	frequency range
d_t	delta-delta MFCCs
ψ_j	Scatter transform wavelet
S_n	n-th order Scatter transform

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 & 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, Barnard, Karpov, & Schultz, 2014b; Rosenberg, Audhkhasi, Sethy, Ramabhadran, & Picheny, 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 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 (C. Becchetti & 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 & Li, 2013; S. . e. Watanabe & Chien, 2015) that work being done in Machine Learning has enhanced the research of automatic speech recognition. Similarly any progress made in ASR usually constitutes 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, Bishop, & Minka, 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 & 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) (M. J. F. Gales, Watanabe, & Fosler-Lussier, 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, Yu, 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 (A. Hannun et al., 2014) with modest results without a language model. However, applying a language model was observed to serve as a correction factor further improving recognition results (A. Hannun et al., 2014).

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, Kuo, Rennie, and Picheny (2015) recommends around 2000 hours of transcribed speech data for a 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, Italian and English languages were substituted and used as control variables to study low resource effects of a language when exposed to speech models developed in this work. Therefore, English and Italian languages simulated low resource constraint by purposely limiting the number of hours of recorded speech data during ASR system training.

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 & 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 & 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. An 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 668,522 words.

1.5 Research aim and objectives

In this work, we develop speech processing and language models based on deep and recurrent neural network implementations. These models use input features which

are of interest to new and low resource languages. 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 and Italian languages. The aim of this research is therefore to build competitive ASR systems in a resource conservative manner, encompassing both system resources as well as training data.

The research objectives were as follows:

- Discover fundamental tasks relating to Language learning. In particular, speech recognition;
- Discover building blocks for creating ASR systems generally, and then, limitations for new languages;
- Build robust ASR systems using methods that also address resource concerns; and
- Build and evaluate resource-friendly, end-to-end 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 and/or ensure training completes within few hours to few days;
ii. Optimisation of sub-tasks and training architecture within the ASR pipeline;
iii. Observe and recommend models which perform better or train faster than others;
iv. Make efficient use of training parallelism;
v. Obtain better or close to 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 sub-word, 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 sub-word, word and syntax level speech model based on Character-Temporal-Classification CTC.

1.5.1 Research Question

Considering the recent development of end-to-end systems facilitated by deep-learning and sequence modelling, is it possible to combine the recent pattern recognition strides in deep scattering transform with an end-to-end sequential model that results in a robust speech recognition system that new and low resource languages can leverage? In addition, can supporting Speech recognition sub-systems be replaced or enhanced or simplified by sequence-oriented end-to-end deep learning models?

1.6 Main Contribution to knowledge

This work uses a character-based neural language model for the low resourced language of Wakirike. In addition, 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 language model had a better perplexity score than a 5-gram language model baseline, the DSN-CTC end-to-end sequence model performed competitively but not better than the baselines with a Word Error Rate of 12.9% and 76.8%; for SVCSR and LVCSR tasks respectively.

The main contributions to knowledge of this research include:

- Rather than developing separate systems including Acoustic Model, Language Model, phonetic dictionary, aligned text and speaker related data transformation, the systems developed in this research use a single end-to-end frame-work. This framework, on the other hand, does not require separate sub-system train-

ing, but rather, uses only input audio and output text sequences for training. This is quite beneficial for low resource settings.

- Detailed alignment of text and speech is not required only rough alignment comprising of segmented utterances and equivalent text
- The designed ASR system is enhanced with a robust character-based Recurrent Neural Network Language Model which is trained integrally within the super end-to-end model.
- Contribution to the Zero Resource challenge (Versteegh et al., 2015) in terms of sub-word modelling of speech features using lightweight Deep Scattering Network (DSN) and modelling of syntax-level speech with an end-to-end speech model. Since the speech model is an RNN-sequence speech model, the output speech text is modelled at the syntax-level as opposed to the word-level.
- This research also implemented hybrid model subsystems based on alternative sequential models specifically for the wakirike language including
  i. Wakirike Diacritic text converter from plain Wakirike text
  ii. Phonetic Dictionary for Wakirike language
- This research, at the early stages, investigated the design of an unsupervised syllable-phone recogniser using auto-correlation and Gaussian Mixture Model (GMM)

1.7 Scope of the study

This study acknowledged from the onset that it may not be able to gather the data required to build ASR systems for the Wakirike language as this is an initial problem this work hopes to get a step closer to solving. As a result, the English and Italian languages were substituted and simulated low resource constraints by reducing the amount of hours of recorded speech data for training or reducing the vocabulary size. This provides a rough estimate of how much of the target Wakirike language speech data will be required to provide equivalent results based on this multilingual approach.

1.8 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.

1.9 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 struggles 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 Machine Learning problem that models speech production, transmission and perception and speech models are trained from language-specific data. While these ML 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 contribution and the outline of this thesis.

Chapter 2

Literature Review

This chapter describes the transition from generative speech models to discriminative end-to-end recurrent neural network models. Low speech recognition strategies are also discussed and their 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, a vector space 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.

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 (M. Gales, Young, 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

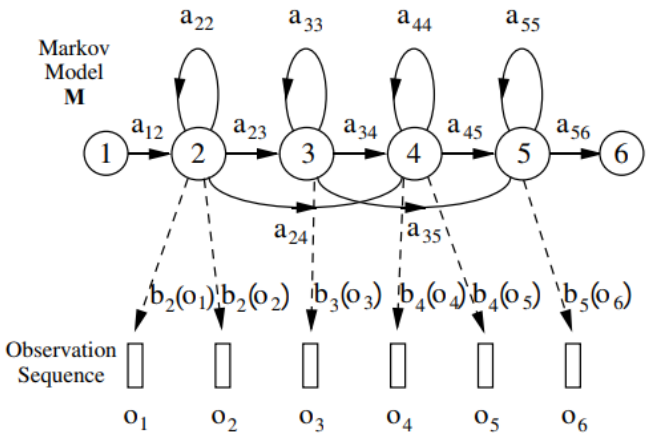


Figure 2.1: HMM Generative Model showing six states (1 to 6); state transition probabilities, a_{12} to a_{56}; and, emission probabilities, b_2 to b_5 for observations $\mathbf{o}_1$ to $\mathbf{o}_6$ (Young et al., 2002)

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, 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

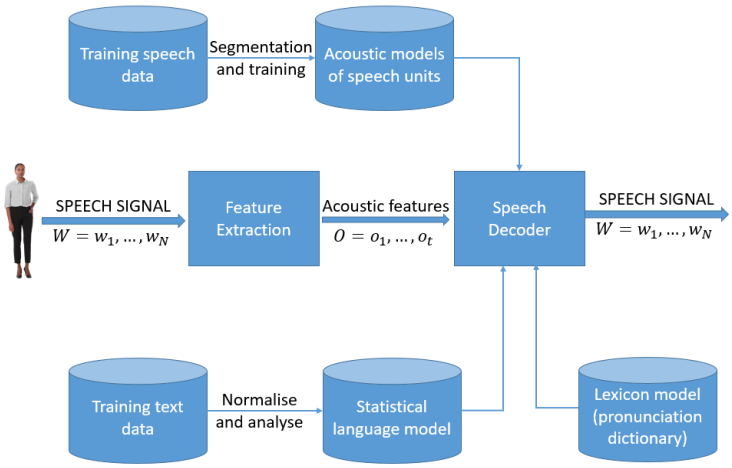


Figure 2.2: Automatic Speech Recognition Pipeline showing main components of Feature Extraction, Acoustic Model, Language Model and Lexicon Model (Besacier et al., 2014a).

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 applies this knowledge in a similar domain having different set of system parameters(Ramachandran, Liu, & Le, 2016). Early work of this nature for speech recognition is demonstrated in (Vu & 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 schemes 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, Desouza, Mercer, Pietra, & Lai, 1992; Kuhn & Mori, 1990) 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, Ducharme, Vincent, and Jauvin (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 T. Luong, Socher, and Manning (2013); Mikolov, Deoras, Kombrink, Burget, and Černockỳ (2011); Sutskever, Vinyals, and Le (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, Socher, & Manning, 2014).

For the neural network implementations so far seen, a large amount of data is required due to the nature of the vocabularies to be large, even for medium-scale speech recognition applications. Y. Kim, Jernite, Sontag, and Rush (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 Y. Kim et al. (2016), however based on Gated Recurrent Unit (GRU) RNNs (Cho et al., 2014), for the Wakirike 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, Burget, et al. (2011) and Ghoshal, Swietojanski, and Renals (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 hat-swap 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, Burget, et al. (2011) 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, Lamblin, Popovici, & Larochelle, 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 & 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 intensiveA. Hannun et al. (2014). 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 & Povey, 2000) and the use of neural network precursors to initialise the HMM parameters (Mohamed, Dahl, Hinton, 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 (A. Y. Hannun, Maas, Jurafsky, & Ng, 2014) 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, Puhrsch, and Synnaeve (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 S. Mallat (2016). 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 networks. 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. The proposed speech model therefore, 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 & 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 & 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 & 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 (S. 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

Methods, Models and Systems

This chapter describes the system building methodology (Nunamaker Jr, Chen, & Purdin, 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. 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.

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) algorithms.

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. This knowledge includes among others: How well does the DSN features train using an end-to-end approach? What range of features can be discovered using the DSN approach? How fast can we train models using DSN features in end-to-end ASR modelling? Are there benefits to be gained by applying sequence models to low-resource ASR systems?

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
- ESPNet

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
- End-to-End Speech Network Toolkit (ESPNet) 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.

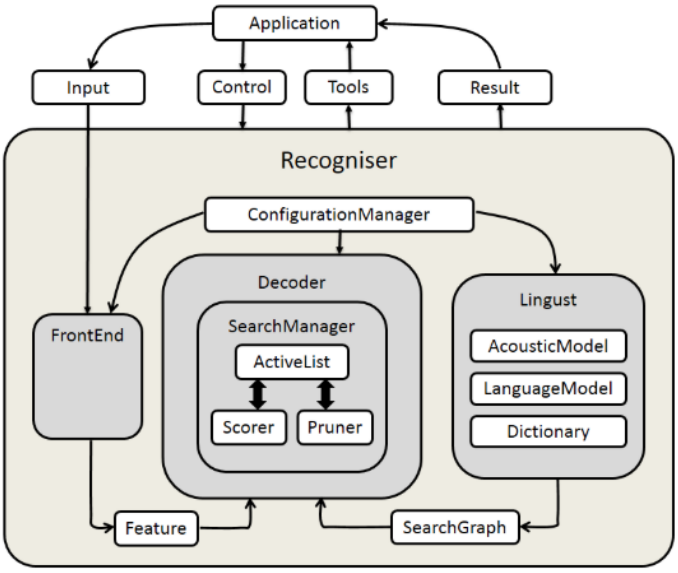


Figure 3.1: CMU Sphinx4 recogniser system. The core modules: `FrontEnd`, `Decoder` and `Linguist` coloured in grey.

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. 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 program 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, Ghoshal, et al. (2011) 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).

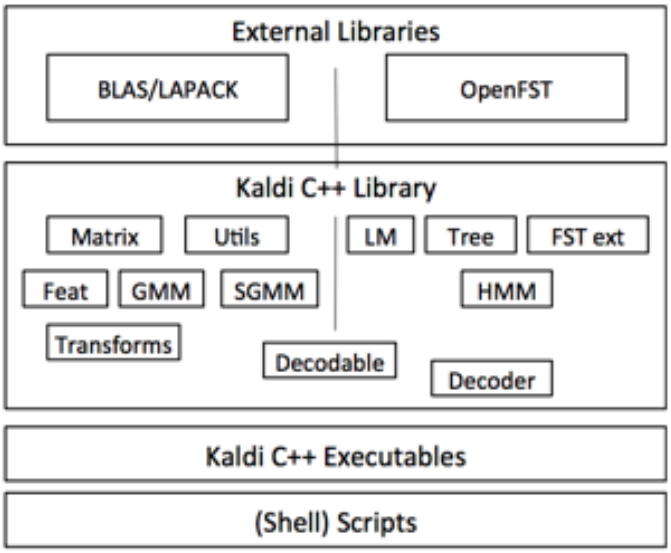


Figure 3.2: The Kaldi Architecture(Povey, Ghoshal, et al., 2011) has four layers. At the deepest layer are the external libraries, followed by the C++ back-end library. At the top: are the C++ executables called by shell scripts

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 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 (A. Hannun et al., 2014). 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, Pouzyrevsky, Clark, & Koehn, 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(Ardila et al., 2019) and the Wall Street Journal (WSJ)(Paul & 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 & Mallat, 2011; Sifre & 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

$$S_1x(t, j_1) = |x \star \psi_{j_1}| \star \phi(t) \tag{3.3}$$

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

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
```

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

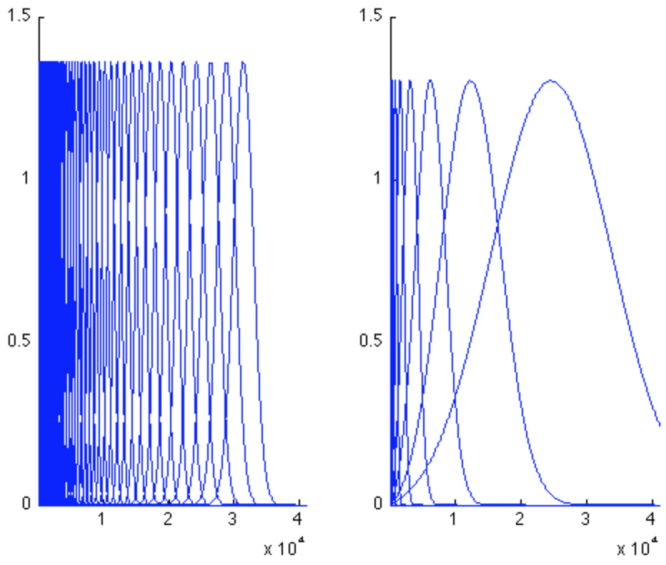


Figure 3.3: Scatter transform wavelet filter plots of various dilations of J

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);
```

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);
```

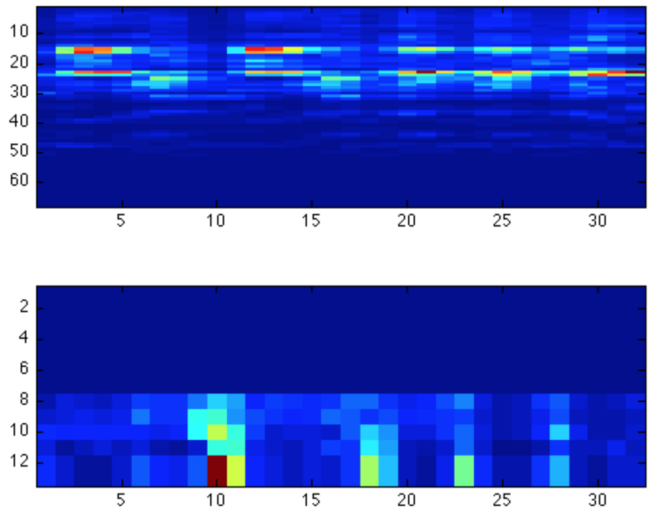


Figure 3.4: Unnormalised scattergram

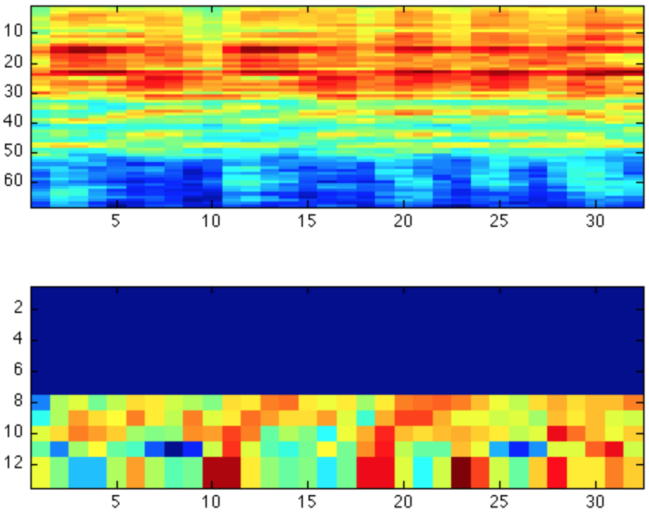


Figure 3.5: Log normalised scattergram

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, Isard, and Murray (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" 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

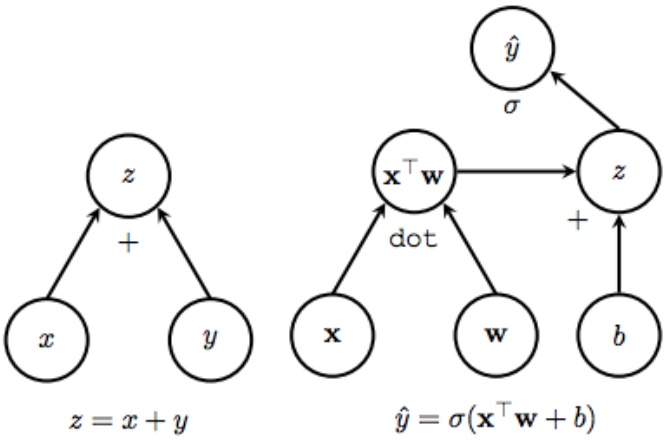


Figure 3.6: Sample TensorFlow computation graphs(Goldsborough, 2016) showing a simple addition operation on the left and a more involved sequence on the right comprising a dot operation followed by an addition and then a sigmoid operation

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 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 I. Goodfellow, Bengio, and Courville (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.

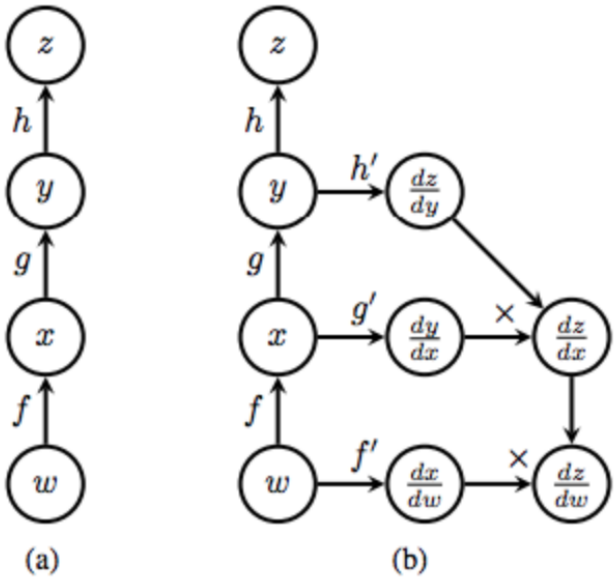


Figure 3.7: Tensorflow graph with backprop nodes (Goldsborough, 2016). The forward propagation is on the left (a) and; the Forward propagation with back propagation (b) is on the right

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, Sutskever, & Hinton, 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) (Pot, Monceaux, Gelin, & Maisonnier, 2009). 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. Together 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 & Ney, 2001) and methods described in (Novotney & Schwartz, 2009). It was observed that Alisa provided good alignment but was not

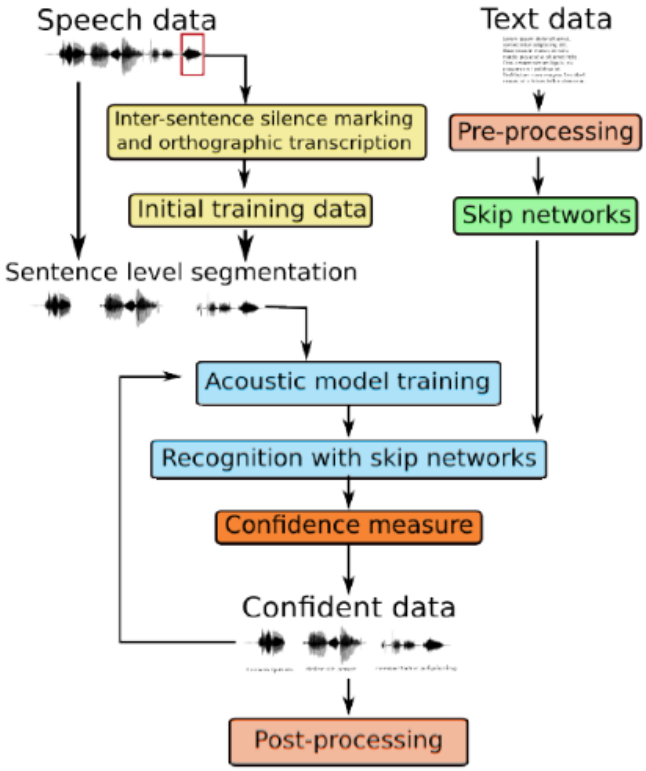


Figure 3.8: Alisa iterative architecture (Stan et al., 2016) involving acoustic model, skip-network recognition and confidence level determination

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 Pilot Studies

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 the 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 estimates. This presents an unnormalised posterior distribution measurement of phoneme segments over the entire signal. Note that this was a pilot study, and as such, the data used as a single 3-second audio recording made by the researcher, as a demonstration of an alternative method to estimate phoneme distribution. This experiment was designed for the purpose of exposure to Matlab audio processing toolbox. Furthermore, apart from the fact that this research eventually utilises more advanced correlation techniques using wavelets, this method did not fit the thesis objectives. The end-to-end method this work derives discriminates

between classes at the character level rather than the phonetic level. Therefore, only Segmentation and auto-correlation steps were performed. Gaussian Mixture Model (GMM) density estimation step was truncated as this aspect is a well understood procedure and did not add to the underlying objective of exposure to Matlab audio processing toolbox. The first two steps of segmentation and auto-correlation which was done for the single audio file is specified in the paragraphs below.

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 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 the segmented phonemes are calculated. Although the procedure is subject to degrade due to signal channel distortion associated speech production, this exercise helped to further emphasise the need for robust signal distortion invariant speech features and pre-processing highlighted in the section 2.3.3.

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

Generally, speech recognition preprocessing uses a fixed, overlapping window for segmentation. The segmentation algorithm designed for this experiment rather attempts to simulate and segment phoneme transitions using a natural phonetic transition process between vowel (periodic signals) and consonants (non-periodic-noisy signals). Figure 3.10 describes the various steps of the segmentation phase while

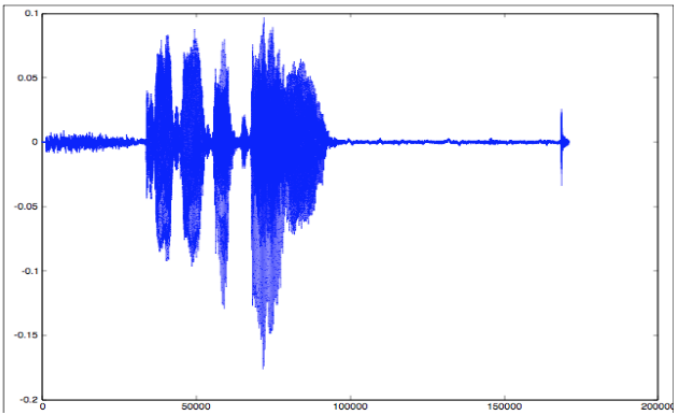


Figure 3.9: Original waveform input from 3-second utterance for auto-correlation

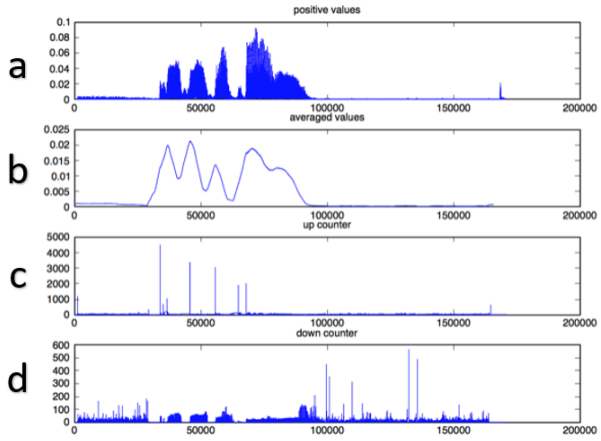


Figure 3.10: (a) Positive values of original waveform (b) Filtered values (c) Peak counter (d) Trough counter

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 about the zero value on the x-axis (Figure 3.10a). The pre-filtering removed all the negative signal values and retained only all the positive values. At the next step, a smoothing kernel is selected based on experimentation to perform both smoothing as well as determining the peaks and trough (3.10b). The simple moving-average filter had a range of 5000 points and was used to extend the vowel regions so to make them more pronounced as well as smooth out the signal profile. Then a threshold is applied to segment the waveform based on discovered inflection points (Figure 3.10c and d). Segmentation can take place as the signal transitions between peaks and troughs in Figure 3.10.

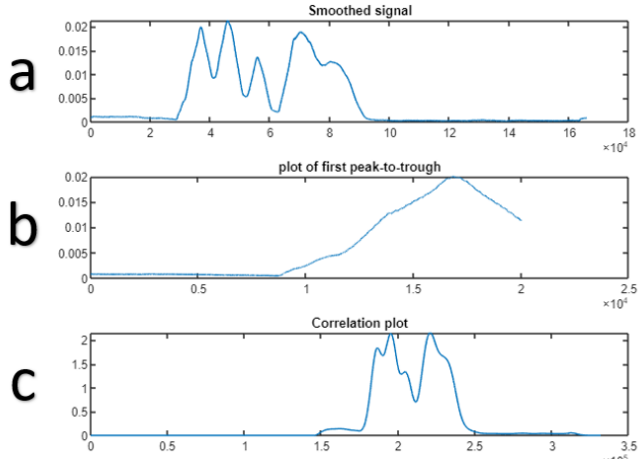


Figure 3.11: (a) Positive values of original waveform (b) Isolated Segment plot (c) Correlation plot of (a) against (b) plot. counter

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. This was achieved by the filtering done at the segmentation stage. Figure 3.11 below shows sample auto-correlation result for the first peak-to-trough segment.

Figure 3.11(a) is the smoothed signal from the segmentation procedure. Figure 3.11(b) is the first peak to trough indicating the first phoneme in the utterance, and Figure 3.11(c) shows the auto correlation plot. From Figure 3.11(c), it can be seen from the twin peaks in the cross-correlation plot indicate that the phoneme-pair in the first peak-to-trough segment was discovered twice within the signal. This can form the basis of the phoneme's posterior distribution. Determining if there was a way of improving the accuracy of phoneme detection was not within the scope of this research.

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 sec-

ond phase top-bottom auto-correlation. The major weakness of this auto-correlation method is the lack of context between the phonemes. For example consonants, which comprise non periodic noise signals, will be difficult to detect without the context of surrounding preceding and succeeding phonemes. The longer the proximity context, the higher the level of distinction between phoneme pairs. These contextual relationships, therefore, can not be resolved in shallow systems such as this. The Bayesian method of segmentation (Kamper, Jansen, & Goldwater, 2016), provides an alternative method which seeks to improve on these weaknesses using ASR feature preprocessing. In this scheme, a combination of acoustic embedding and Dynamic Time Warping (DTW) for clustering is employed to replace auto-correlation. In essence, it is more efficient to use clustering from extracted features with less intrinsic noise than using a naive smoothed audio data pre-processing method.

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 Sequence-to-sequence Model Experiments

A significant issue arises when using HMM-based toolkits such as Kaldi in low resource ASR applications. 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 (M. J. Gales, Knill, Ragni, & Rath, 2014; Ragni & Gales, 2018; Ragni, Knill, Rath, & Gales, 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 introduces RNN sequence-to-sequence modelling and some of the pilot studies done using these models and in Chapters 7 and 8, how these methods deal with the problem of automatic speech alignment in a fashion which is 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

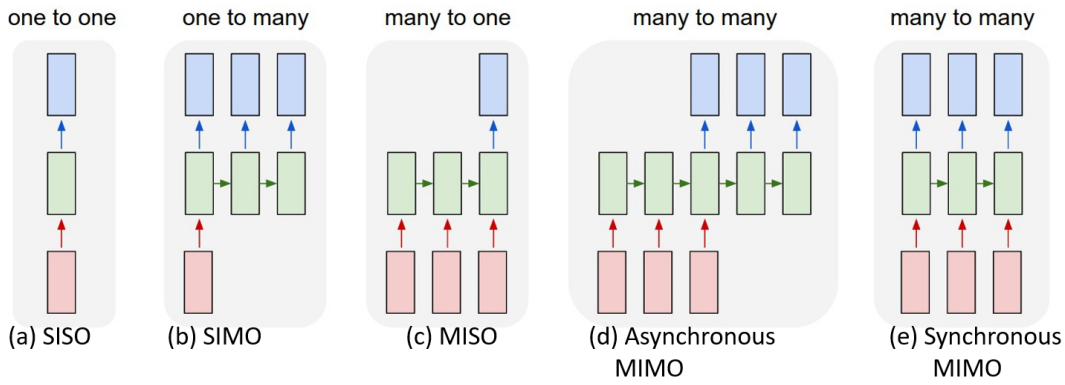


Figure 3.12: Relationship Types and their neural network interpretation. (Karpathy, 2015)

for successful training.

In the wild, three types of naturally occurring sequence relationships exist: the one-to-many relationship also referred to here as Single Input Multiple Output (SIMO); the many-to-one relationship also referred to here as Multiple Input Single Output (MISO), and; the many-to-many also referred to here as Multiple Input Multiple Output (MIMO). The one-to-one relationship also referred to as Single Input Single Output (SISO) is excluded here. Although the SISO relationship is a naturally existing relationship, this relationship is not a sequence-type relationship and will not be modelled using a Recurrent Neural Network, but rather, can be modelled using regular Deep Neural Networks. In addition, the MIMO relationship can be further subdivided into synchronous and asynchronous MIMO. In synchronous MIMO, the inputs and the outputs have equal lengths but in the asynchronous MIMO, the input and the output lengths are not necessarily equal.

Figure 3.12 illustrates how the five relationship types are translated into neural networks where (3.12a) refers to a regular DNN structure and (3.12b to d) are different RNN structures. Examples of each type of relationship structure are given in Karpathy (2015). In Figure 3.12, the red blocks at the bottom are input blocks, the green blocks are neural network units and the blue blocks are output blocks. For SIMO relationship (Figure 3.12b), A single input will generate a sequence as output. An example of a SIMO is an image captioning task where a single image input will generate a sentence or sequence of words. For the case of the MISO (Figure 3.12c), a sequence of inputs will generate a single output. As an example, consider a fault detection task where a sequence of historical data can classify whether an

instrument is faulty or not. In this research, we only implement asynchronous MIMO (Figure 3.12d) and synchronous MIMO (Figure 3.12e) which satisfied the design requirement of the models implemented. In the asynchronous case (3.12d), we see here that due to the misaligned or jagged alignment of inputs to outputs, there is a degree of freedom between the inputs and outputs such that they do not have to have equal number of inputs and outputs. Within the RNN model, this is implemented using two different Recurrent Neural Networks. One RNN for the inputs known as the encoder network and one RNN for the output known as the decoder network. Together these encoder and decoder networks become a RNN-Transducer network. For the synchronous MIMO in Figure (3.12e), however, only one RNN is implemented constraining the number of inputs to be equal the number of outputs. Note that both MISO and SIMO are also implemented with a single RNN structure, the difference between MISO, SIMO and synchronous MIMO is that for SIMO and MISO the corresponding inputs or outputs are ignored. For the SIMO, we only input the first RNN sequence and ignore the rest. For the MISO sequence, we only harvest the last output and ignore the preceding sequence of outputs.

3.4.1 Procedure for designing sequence-to-sequence RNN models

In the next two sections (3.4.2 and 3.4.3), two pilot study experiment designs are discussed. These experiments had two objectives; the first objective was to implement regular speech-recognition tasks using sequence-to-sequence methods and second objective was the exploration of sequence network designs described in the previous Section (3.4). Since these experiments were mere pilot studies and not required in the final outputs, the results were not improved upon and were reserved for further investigation in future publications and the emphasis of these experiments was centred on sequence modelling design and implementation. A detailed discussion of the final sequence models designed in this thesis is reserved for Chapters 6 and 7. Generally for the sequence-to-sequence model experiments designed in this thesis we follow the following steps.

- Step 1: Select the sequence RNN model that satisfies the requirement for the sequence relationship being modelled.

- Step 2: Select an appropriate RNN (see Chapter 4 for discussion of RNN types) for each RNN component.
- Step 3: Design neural network components including
    i. Number of hidden layers
    ii. Number of neurons in hidden layers
    iii. Network saturation parameters
        – Weight initialisation
        – Non-linear function
        – Number of epochs
        – Learning rate
        – Cost function
        – Optimiser
- Step 4: Design Regularisation measures. These are measures to ensure your network, which can be viewed as a high dimension function fitter does, not over-generalise or under-generalise.

Note that in this research, parameters in steps 3 and 4 are either experimentally determined or selected based on similar research which has yielded the desired outcomes.

3.4.2 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 this section and the next section represent precursor experiments centred around sequence modelling for speech recognition tasks, sections 3.4.4 and 3.4.5 introduce the major experiments in this work later discussed in Chapters 6 and 7.

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

0	nyọ koruapụ ma igbiki mị ọkị se ini ḍuko ini p...	nyo koruapu ma igbiki mi oki se ini duko ini p...
1	ọbụḍuko oyighọrịapụ fịnjị ma galili chịngị mi ...	obuduko oyighoriapu finji ma galili chingi mi ...
2	ini ori ọrị sịkị ini kpẹrẹkị ọ tẹkẹmẹ inia bie...	ini ori ori siki ini kpereki o tekeme inia bie...
3	mịn jizọs kraịst bẹ pakabọ furo mị simeọkụ-ẹ d...	min jizos kraist be pakabo furo mi simeoku-e d...
4	juda bẹ pẹrẹz bẹ na zẹra bẹ na yimẹ (inia nyen...	juda be perez be na zera be na yime (inia nyen...
5	ram bẹ amịnadab bẹ yimẹ amịnadab bẹ nasọn bẹ y...	ram be aminadab be yime aminadab be nason be y...
6	salmọn bẹ boaz bẹ yimẹ (rehab ma nyọngọrọ-ẹ) b...	salmon be boaz be yime (rehab ma nyongoro-e) b...
7	jẹsi bẹ amanyana devid bẹ bẹ yimẹ devid bẹ sol...	jesi be amanyana devid be be yime devid be sol...

Figure 3.13: Diacritic symbol generator training data sequences

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 as post-processing technique for an end-to-end model output text. This experiment follows the design procedure outlined in Section 3.4.1. Figure 3.13 gives example input and output sequences for this experiment.

Although after training for 75 epochs (Figure 3.14) the accuracy of the Wakirike diacritics sequence generator model was still unacceptable (6%), the model may have been improved by changing the sequence-relationship model from an asynchronous MIMO to a synchronous MIMO sequence model. Improvements from a synchronous design stems from the insertion and deletion errors observed from the output transcription. Ensuring the outputs and inputs had equal length would therefore constrain the model for better results. The main aim of the pilot experiment, however, was to implement asynchronous transducer RNN design and therefore further experiments for the diacritics sequence generator were not done.

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

This follow up experiment to the previous experiment in section 3.4.2, 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 performed

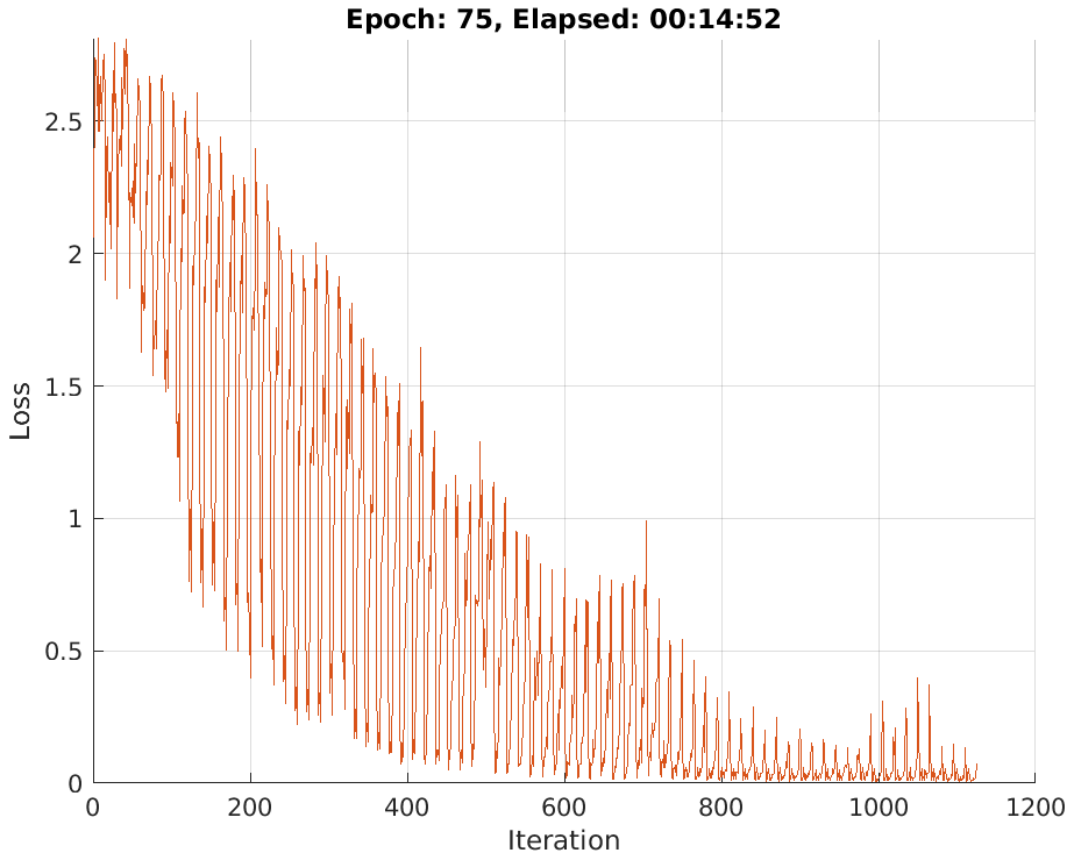


Figure 3.14: Diacritic symbol generator model training loss

used the latter non-marked graphemes as input. For this experiment, a pre-trained model having a 2-layer transducer RNN having 256 hidden units was fine-tuned on the training data. Appendix V shows the output Wakirike phonetic dictionary generated from this experiment.

What follows in the next three sections are sequence-to-sequence experiments actively developed in this research and are detailed in chapters (6 and 7). A brief summary of the experiments are highlighted in the following sections (3.4.4, 3.4.5 and 3.4.6). Note that these models all utilise TensorFlow deep learning library including the Bi-directional speech model (section 3.4.5) which is built on top of Mozilla DeepSpeech with the exception of section 3.4.6 which is based on PyTorch; a python library having deep learning features identical to that of TensorFlow.

3.4.4 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.

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

A similar LSTM sequence-to-sequence network based on Baidu Research's original research design (A. Hannun et al., 2014) 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).

3.4.6 ESP-Net Experiments

The ESP-Net (End-to-end Speech Network) toolkit (S. 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.

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, Roukos, Ward, & Zhu, 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

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

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, Mat-

lab 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, metrics for the evaluation of the models built in this research is discussed.

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 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

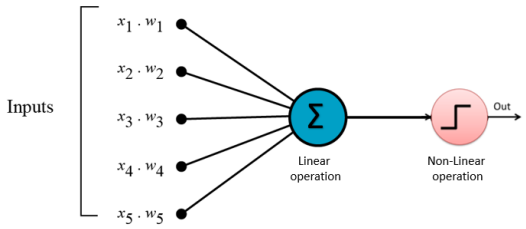


Figure 4.1: Neuron cell (Landahl et al., 1943) where x_i are inputs and w_i are input weights

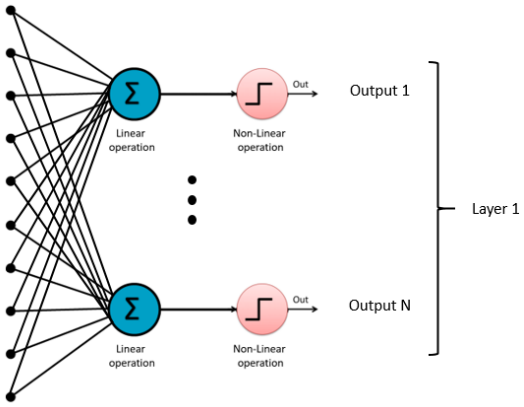


Figure 4.2: Perceptron algorithm having multiple neuron cells

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. The nonlinear portion of this equation is given by

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

The other linear half of the equation is 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:

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

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

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 or saturation (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.

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

where l is the label value for the output value y.

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 RNNs from a specialised MLP or the 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 & Deng, 2016). This will therefore include multi-layer Perceptrons (MLPs) having more than one hidden layer (Dahl, Yu, Deng, & Acero, 2012) as well as deep belief networks (DBNs)(Mohamed, Dahl, & Hinton, 2009; Yu, Deng, & Dahl, 2010) having a similar structure. There-

fore, 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 & 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 (A. Y. Hannun et al., 2014; Yu & 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

1 //$\mathbf{x}_t$ is the input feature sequence //$\mathbf{I}_t$ is the label sequence;
2 initialise weights randomly;
3 **for** *error is significant or epochs less than maximum* **do**
4 **for** $t \leftarrow 1; t <= T; t \leftarrow t+1$ **do**
5 //forward computation ;
6 $\mathbf{u}_t \leftarrow \mathbf{W}_{xh} + \mathbf{W}_{hh}\mathbf{h}_{t-1}$;
7 $\mathbf{h}_t \leftarrow f(\mathbf{u}_t)$;
8 $\mathbf{v}_t \leftarrow \mathbf{W}_{hy}\mathbf{h}_t$;
9 $\mathbf{y}_t \leftarrow g(\mathbf{v}_t)$
10 **end**
11 **begin**
12 //backprop through time ;
13 $\delta_T^y = (\mathbf{l}_T - \mathbf{y}_T) \bullet g'(\mathbf{v}_T)$;
14 $\delta_T^h = \mathbf{W}_{hh}^\top \delta_{t+1}^h + \mathbf{W}_{hy}^\top \delta_T^y \bullet f'(\mathbf{u}_T)$;
15 **for** $t \leftarrow T-t; t >= 1; t \leftarrow t-1$ **do**
16 $\delta_t^y = (\mathbf{l}_t - \mathbf{y}_t) \bullet g'(\mathbf{v}_t)$;
17 $\delta_t^h = \left[\mathbf{W}_{hh}^\top \delta_{t+1}^h + \mathbf{W}_{hy}^\top \delta_t^y\right] \bullet f'(\mathbf{u}_t)$;
18 **end**
19 **end**
20 update weights and biases according to gradient descent;
21 **begin**
22 $\mathbf{W}_{hy}^{new} = \mathbf{W}_{hy} + \gamma \sum_{t=1}^{T} \delta_y^t \mathbf{h}_t^\top$;
23 $\mathbf{W}_{hh}^{new} = \mathbf{W}_{hh} + \gamma \sum_{t=1}^{T} \delta_h^t \mathbf{h}_{t-1}^\top$;
24 **end**
25 **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, Senior, and Beaufays (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:

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

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

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

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

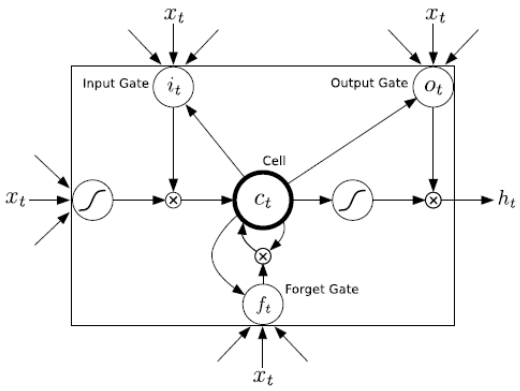


Figure 4.3: An LSTM Cell showing Input, output and forget gates (Graves et al., 2013)

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

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 A. Y. Hannun et al. (2014) 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.

A. Y. Hannun et al. (2014) 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 $\mathcal{B} : A^{`T} \rightarrow A^{\leq T}$. 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'^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 la-

bels. 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

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

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

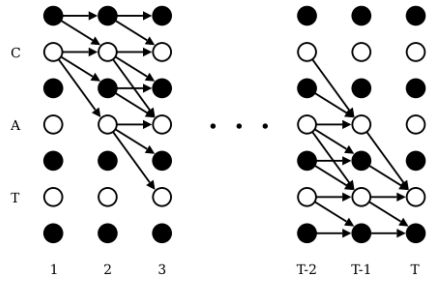


Figure 4.4: Beam Search Lattice Structure showing forward (left) and backward (right) paths (Graves et al., 2006)

$\alpha(2,3)$, 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, $\alpha(t,u)=0$.

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 soft max 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 4.40, 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

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

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 Attention Mechanism

This can be likened to how neural networks simulate how the human brain functions in achieving tasks. Attention is a cognitive process of concentrating on one or a few things and at the same time ignoring others. Therefore attention mechanism focuses more on the informative data segment of a model that contributes more to the final output. It creates an underlying relationship between the input and output of a model producing improved transduction from input to output sequence.

Attention mechanism is an extension to the sequence-to-sequence asynchronous MIMO RNN architecture discussed in Section 3.4. 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. Some other strategies which have also attempted to achieve this includes a combination of convolutional and recurrent schemes (Gehring, Auli, Grangier, Yarats, &

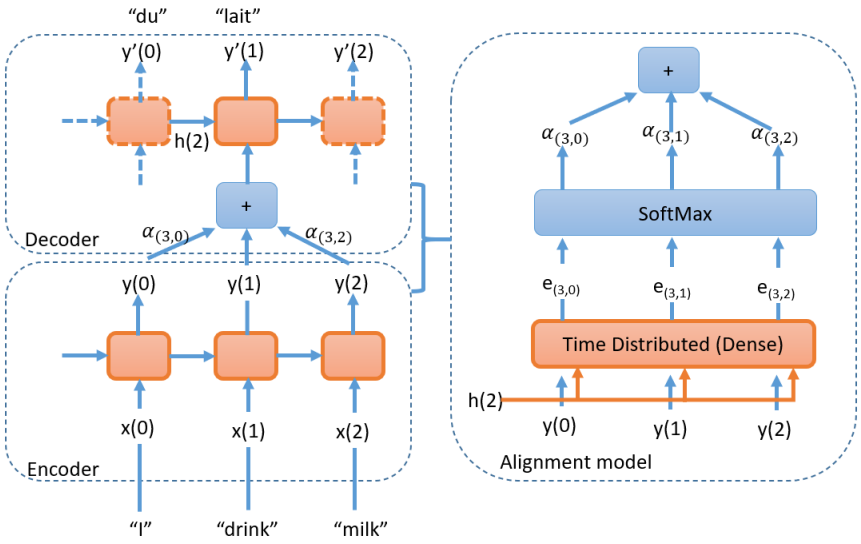


Figure 4.5: Attention mechanism is centred around a time-distributed dense operation that determines similarity measure between current decoding sequence hidden input and all input sequence outputs.

Dauphin, 2017; Kaiser & Bengio, 2016; Kalchbrenner et al., 2016).

Attention is achieved by joint training of an attention neuron that takes outputs of the hidden layer of each input in the encoder along with the entire RNN as shown in figure 4.5. In the original attention mechanism proposed by Bahdanau, Cho, and Bengio (2014), the weighted sum of all encoder outputs will determine what segment of the sequence receives more attention. The output of this attention neuron at each time-step is then concatenated with the output of the RNN and fed to the decoder.

Since dot product is a fairly decent measure of similarity, and faster to compute,M.-T. Luong, Pham, and Manning (2015) proposed a dot product between encoder's output and decoder's previous hidden state as a basis for attention mechanism. For this to work however, both of these vectors must be of the same dimensions. This type of attention mechanism is known as multiplicative attention. Just like in Concatenative attention, the dot product gives a score and all the scores (at a given decoder timestep) go through a softmax layer to give the final weights.

In equation 4.62 below, $\tilde{\mathbf{h}}_{(t)}$, is the output of the attention neurons and $\alpha_{(t,i)}$ represents the weight of each output that is jointly learned where $e_{(t,i)}$ represents variants of how the weights can be obtained. Here three variants are shown either dot, general or concatenation. The dot and general are variants proposed by M.-T. Luong et al. (2015). The general proposition represents $e_{(t,i)}$ as the dot product

of decoder hidden weights with the linear transposition of the encoder outputs (i.e. time-distributed Dense layer with no bias term). Another modification proposed was to use the decoder's hidden state at the current time step rather than at the previous time step (i.e., $\mathbf{h}_{(t)}$ rather than, $\mathbf{h}_{(t-1)}$), then use the output from the attention mechanism ($\tilde{\mathbf{h}}_{(t)}$) directly to compute the decoder's predictions (rather than using it to compute the decoder's current hidden state). They also discovered that by adding a re scaling parameter $\mathbf{v}$, to the concatenative attention mechanism these three dot product variants perform better than the simple concatenative attention.

$$\begin{aligned} \tilde{\mathbf{h}}_{(t)} &= \sum_i \alpha_{(t,i)} \mathbf{y}_{(i)} \\ \text{with } \alpha_{(t,i)} &= \frac{exp(e_{(t,i)})}{\sum_{i'} exp(e_{(t,i')})} \\ \text{and } &= \begin{cases} \mathbf{h}_{(t)}^{\top} \mathbf{y}_{(i)} & dot \\ \mathbf{h}_{(t)}^{\top} \mathbf{W} \mathbf{y}_{(i)} & general \\ \mathbf{v}^{\top} \tanh\left(\mathbf{W}[\mathbf{h}_{(t)}; \mathbf{y}_{(i)}]\right) & concatenate \end{cases} \end{aligned} \tag{4.62}$$

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. Although this study makes use of RNN transducers with attention, integration of the self-attention architecture was left for a further study.

4.5 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 Perceptron (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 S. 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 & Mallat, 2011; Peddinti et al., 2014; Sainath et al., 2014; Zeghidour, Synnaeve, Versteegh, & Dupoux, 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.

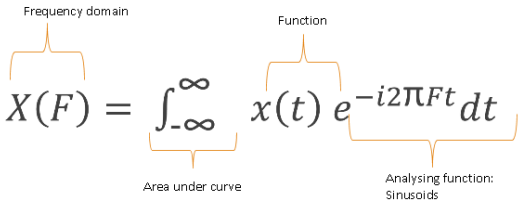


Figure 5.1: Fourier Equation

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

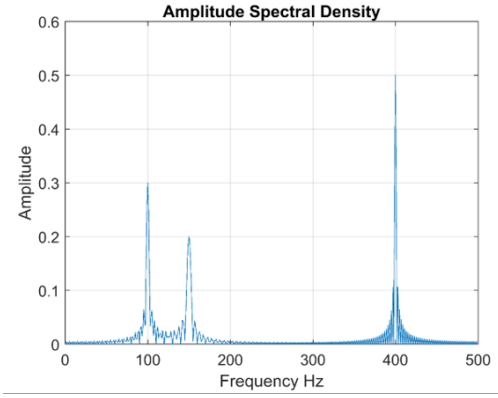


Figure 5.2: Sample Spectrogram from an arbitrary input signal showing frequency-power content of the signal
Continuous Wavelet Transform (2015)

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

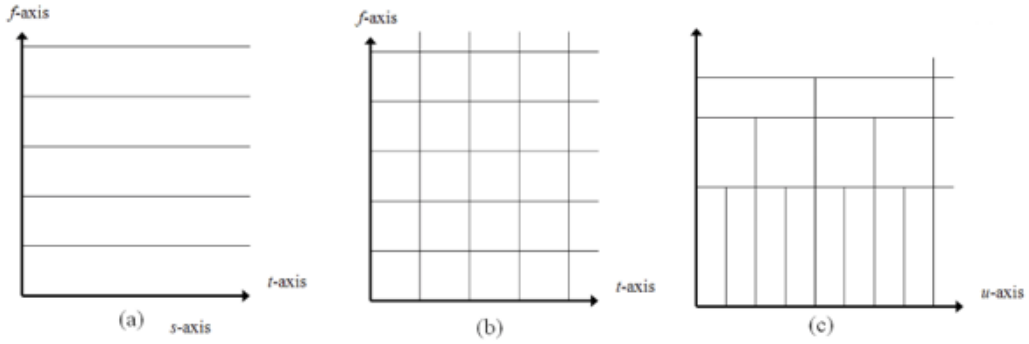


Figure 5.3: Time frequency tiling for (a) Fourier Transform (b) Short-time Fourier Transform (STFT) (c) Wavelet transform

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 (S. G. 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 & 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, Yamagishi, & King, 2010; McLoughlin, 2009). The following paragraphs describe how the Mel filters are derived.

The Mel scale as described by Stevens, Volkmann, and Newman (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. (L. 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)(L. 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 Cosine Transform

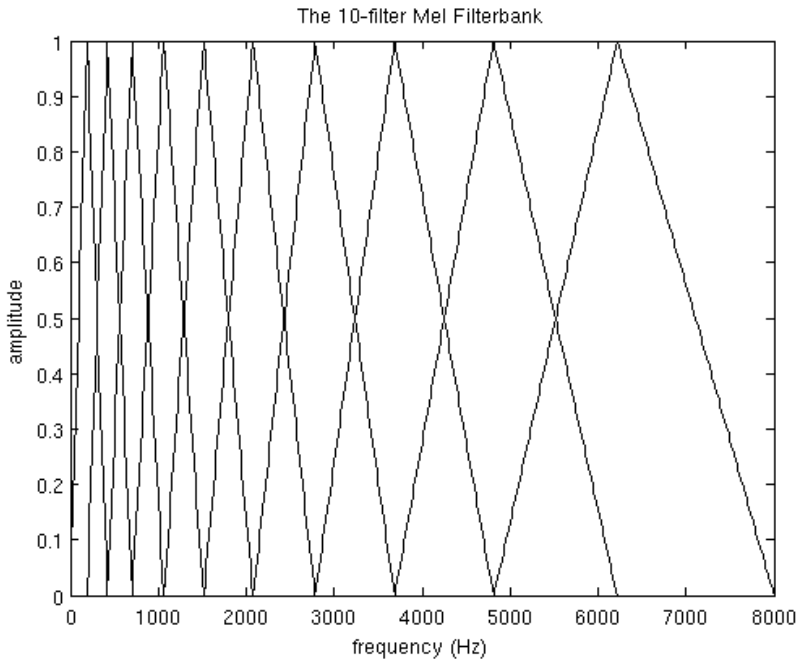


Figure 5.4: Mel filter plot showing overlapping frequency bins (Lyons, 2012)

(IDFT) of the Mel filters. The coefficients are also concatenated with their delta 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(M. Gales et al., 2008).

As an attempt for MFCCs to incorporate dynamic frequency changes of the sig-

nal, 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 (M. Gales et al., 2008).

5.5 Deep scattering spectrum

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

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 MFSC filters are constructed by dilating the wavelet by 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. The resulting coefficients are 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 and the Deep Scattering Network (DSN) is that the 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 $|x \star \psi_{j_1}|$

and the process of time averaging to obtain invariant contracted envelopes reapplied. The resulting first order scattering coefficient were obtained from

$$S_1x(t, \psi_{j_1}) = |x \star \psi_{j_1}| \star \phi(t)) \tag{5.21}$$

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_1x 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.

$$S_2x(t, j_1, j_2) = ||x \star \psi_{j_1}| \star \psi_{j_2}| \star \phi(t) \tag{5.23}$$

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

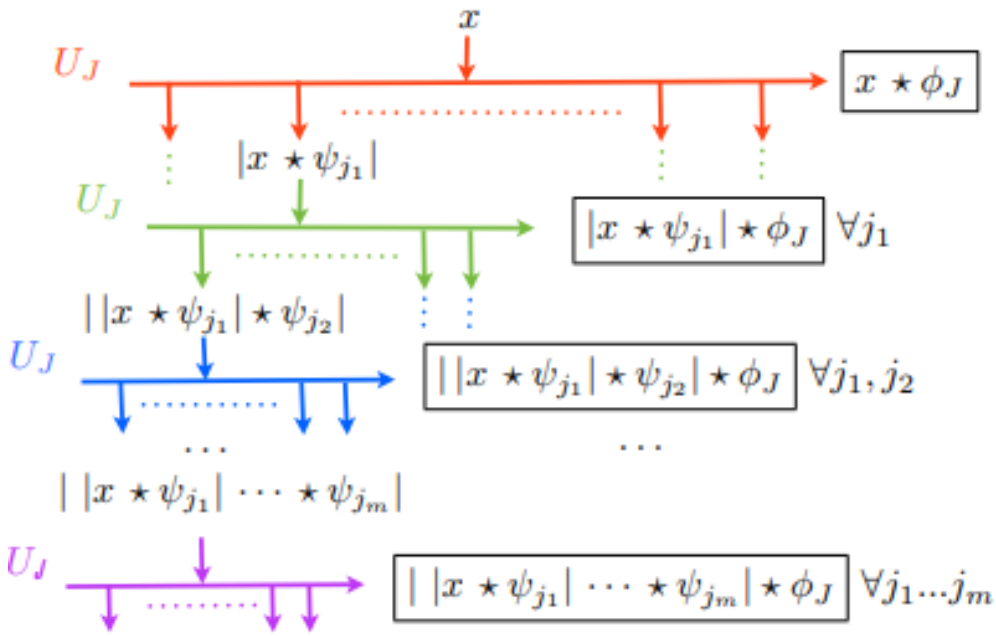


Figure 5.5: Scattering network - 2 layers deep
Andén and Mallat (2011)

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

The details of the language models developed for the Wakirike language is discussed in this chapter. The language models developed draw upon the premise that the grammar of a language is expressed in the character sequence pattern ultimately rendered in word sequences. The two models developed in this chapter follow RNN implementations discussed in chapter 4.

6.1 General Considerations for Sequence-to-sequence modelling

The systems built in this work is centred around Deep Neural Networks (DNNs). A DNN, is a function approximator of some function $f*$. The sequence models developed in this work are based on DNNs. The models can therefore be viewed simply as the mapping of $f*$ from its inputs to its outputs. While the inputs and outputs may vary from one system to another along with that of the design of the DNNs implemented, these models share common features that aim towards the common goal of function approximation.

In Chapter 1, it is mentioned that the systems developed in this work fall under the class of problems known as pattern recognition tasks, whether it be recognition of speech patterns or that of language patterns for language models. They are therefore also grouped into the branch of machine learning algorithms known as

classifiers. I. Goodfellow et al. (2016) formalises a classier as the function $y = f(x)$, which based on a set of learnable parameters, θ, maps an input x to a category y. Therefore defines

$$\mathbf{y} = f(\mathbf{x}; \theta) \tag{6.1}$$

The General criteria considered in this section is a continuation of all the design considerations for sequence models defined in Section 3.4 and Section 3.4.1. These defined criteria represent the features of the network that vary from one sequence design to another and ultimately have influence on the learnable parameters, θ of our DNN-based RNN-sequence model. The following paragraphs therefore discuss these hyper parameters and their selections for the three sequence models involved in empirical analysis in this study.

6.1.1 Selection of Sequence Model

In Section 3.4, we establish five different sequence models types (see Figure 3.12), and the rationale for only using MIMO for systems. Due to the fact that the inputs and the outputs for speech and language modeled in this work, are time series data and so therefore, the natural speech and language observations can be modelled as MIMO sequences. Following this, the next thing to be decided would only be whether the natural occurrence be modelled as synchronous or asynchronous MIMO.

For the design of the character-based language sequence model, the idea is that since the language rules are expressed in the character sequence, then, for each input in the training data the most accurate next character based on all the previous characters leading to the current would be the next found in the training sequence. Therefore, there is only one corresponding output and vice versa, hence, we use a synchronous MIMO relationship to model the language model.

In this work, there are two speech models developed for low resource end-to-end ASR. The first is based on the synchronous MIMO design and the second is based on the asynchronous details of these designs are given in Sections 7.2 to 7.6. For both of these regardless, the CTC algorithm ensures the asynchronous MIMO relationships between inputs and outputs are restored.

6.1.2 Selection of RNN-architectures for sequence modelling

In the previous section (6.1.1) and in Section 3.4, five RNN/DNN sequence-to-sequence models were defined as

i. Single Input Single Output (SISO)

ii. Multiple Input Single Output (MISO)

iii. Single Input Multiple Output (SIMO)

iv. Synchronous Multiple Input Multiple Output (MIMO)

v. Asynchronous Multiple Input Multiple Output (MIMO)

Note that number 1 above can only apply to a regular DNN, while numbers 2 to 5 apply to RNNs. Apart from the super-structures, the second sequence-to-sequence design criteria considers the sub-structures that comprise these super structures at the cell level. There are five of RNN sub architecture considered in this work

i. Regular DNNs

ii. Long Short-Term Memory (LSTM)

iii. Gated Recurrent Unit (GRU)

iv. Bi-directional Recurrent Neural Network

v. Bi-directional LSTM

Chapter 4 presented an in-depth look at the above RNN sub-structures. For the RNN-LM developed in this Chapter, the sub-structure selected was the GRU. Similarly, the Bi-LSTM is used for first speech model experiments. Details of the designs and selection criteria are given in this Chapter and Chapter 7. Note however, that as the name implies, the Bi-directional LSTM (BLSTM) is a Bi-RNN where the regular RNNs are replaced by LSTMs. Furthermore, the results in this Chapter show that there was no need to use the heavier duty LSTMs or Bi-RNNs for the language model development as the deep-GRUs proved to be quite sufficient. Speech models however, required a more robust substructure where BLSTM have been shown to perform better (S. Kim, Hori, & Watanabe, 2017).

6.1.3 Neural Network geometry

Neural network geometry refers to the number of layers and the number of neurons per layer. This will give the total degrees of freedom of the neural network. Increasing the number of neurons per layer causes the neural network to extend the dimensions for discrimination. However, increasing the number of layers enables better generalisation of the dimensional data. These parameters have been selected empirically based on pilot experiments and experiments on neural networks for similar research. Generally for neural networks involving RNNs good practice should have the number of layers start from 2-layers and the number of neurons per layer from 64. Experiments for this research typically had 3-5 layers with the number of neurons being between 128 and 2048.

6.1.4 Network Saturation Parameters

The succeeding paragraphs discuss parameters that are selected to either ensure that the network trains in a stable manner that saturates or that tries to assist the network to train faster. Most of these parameters have been selected based on similar research or based on particular types of neural network architectures.

Weight initialisation

These are the values the neuron weights have at the start of training. If these are widely ranging values, they tend to make the network unstable. However, having values that are uniformly distributed around a stable value such as zero ensures that the back propagation adjustments are also changing at a stable rate that favours the gradient descent movement in the data space towards the global optimum. Weight initialisation for models developed in this thesis followed this Gaussian initialisation and that of common stable weight initialisation models such as Xavier initialisation (Kumar, 2017) or Glorot initialisation (Glorot & Bengio, 2010) .

Non-Linear function selection

In Chapter 4, the following nonlinear functions sigmoid, tanh and RELU were introduced. These functions enable neural networks to navigate nonlinear space as function approximators. Out of these three nonlinear functions, only RELUs (He, Zhang, Ren, & Sun, 2015) are immune to the "vanishing gradient problem". As identified in Glorot and Bengio (2010), the vanishing gradient problem seen in very deep neural networks where gradients get smaller while back propagating through the network and quickly become zero and stopping the network from saturating at that point. Due to the hidden layers of the RNN, they constitute very deep networks and therefore susceptible to the vanishing gradient problem. The three models main models developed in this work use clipped RELUs for activation.

Number of epochs

An epoch is an event that occurs when the neural network has processed all the training data available. Neural networks iteratively get trained until the network saturates or has reached an optimal state where the performance cannot improve further. Usually, it takes several epochs for the network to get to a global optimum assuming all other parameters are configured optimally. In the models developed in this thesis, the number of epochs were selected empirically from pilot experiments. In some of the experiments performed, epochs were selected based on the Research objectives not to train for more than a few hours for the language models or a few days for speech models.

Learning rate

Learning rate has been introduced in Chapter 4. The rule of thumb for DNNs is the larger the network the lower the learning rate should be. Learning rates applied to sequence models developed in this work based on geometries used ranged from 0.0001 to 0.0005.

Cost function selection

The root mean square error, cross entropy and the CTC loss cost functions have been introduced in Chapter 4. This work made use of the cross entropy for the language model and the CTC for the speech models. Comparing CTC to AutoSegCriterion proposed by (Collobert et al., 2016) the differences for AutoSegCriterion are (1) no blank labels are required (2)nodes and possibly edges have unnormalised scores and (3) global normalisation is utilised instead of per-frame normalisation.

Optimiser

Gradient descent algorithm has been introduced in Chapter 4. Different algorithms that improve upon the Gradient descent algorithm, such as ensuring a global minima is determined, are explored in work. The adadelta and adam optimizer are examples of such used in this thesis.

6.1.5 Regularisation measure

Rather than adjusting neural network geometry in order to find an optimal geometry that neither over fits or under fits the data, a common way to avoid over fitting the data is by a method known as dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). Dropout was the regularisation strategy employed by this research. Dropout will therefore determine which neurons will refrain from emitting its output at each layer. Using similar research, dropout values were 20% for the language model and 10% for Bi-RNN with attention transducer and 0% for Bi-RNN only experiments.

6.2 Data Preparation

A published version of the Wakirike New Testament Bible was obtained and used as the data source for RNN training of the language model. There was no readily available soft or online copy of the Wakirike new testament bible. As such, the Wakirike New Testament Bible text corpus text was entered into the ASR system

from the physical copy using a text editor to form a text corpus. The complete corpus had a word count of 668,522 words and a character count of 6,539,176 characters. Following the k-fold cross validation process (Géron, 2019), 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. As the validation set is not seen during training it can be used to keep track of how well the training is going and that it is not over-fitting the data by simply memorising it.

Preprocessing of the text corpus involved selecting a set of characters as the input feature set and removing all other characters not found in the input feature set. The Unicode representations of the character set consisted of letters and punctuation marks. These are one-hot encoded and batched for sequential input. Neural network parameters which are not automatically determined through back propagation are called hyper-parameters. These are usually experimentally determined and manually set while configuring the network. A hyper-parameter for the language model RNN is the input sequence length. For the language model built a 30 characters-long sequence length is chosen. This length is an average phrase sequence. In these phrases, long-term character dependencies of words can be captured. At the same time, keeping the sequence length at this value, and not longer, will pose less of a burden on the computer system resources during parameter computations.

Another hyper-parameter for training used was the batch size. The batch size parameter determined how many 30-character sequences will be trained in parallel in order to speed up the training process. Increasing the batch size also meant an increase in the size of the matrix multiplications being performed and therefore, the computing power system resource being demanded by the language model. By experimenting with various batch sizes it was determined a batch size of 200 was suitable for training the language model with respect to the other training parameters.

6.3 GRU RNN Architecture

The modified LSTM RNN known as the Gated Recurrent Unit (GRU) discussed in Chapter 4 is employed for the neural network model built in this Chapter. 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 architecture of the GRU RNN used to train the Wakirike text corpus had an internal network size of 512 nodes for each layer and was 3 layers deep. In a study by (I. J. Goodfellow, Bulatov, Ibarz, Arnoud, & Shet, 2013), it was shown that increasing the number of nodes in a neural network will lead to over-fitting; however, simultaneously increasing the network depth mitigates this effect. In other words, in order to expand the degrees of freedom of a neural network and at the same time constrain the network to generalise well on unseen data, it is necessary to increase the number of neurons in both length and depth. Experiments carried out in this chapter follow this recommendation. Initial experiments had an internal node size of 128 and a single layer deep. While this showed promise of converging, the error rate was still high, therefore the network was expanded to the final model above. Externally, the network model is further sequenced 30 times, representing the input sequence length hyper-parameter and the number of recurrent connections where each connection represents a sequenced time step.

Another hyper-parameter sensitive to network size is the learning rate. The learning rate is selected in such a manner that an increase in the network size makes the learning rate more prone to overshooting. Therefore, increased degrees of freedom in a neural network will require the learning rate to be made smaller so that it does not overshoot the network saturation point. Small learning rates of between 0.001 and 0.005 were used. Furthermore, the language model neural network was designed to overcome over-fitting using the dropout method (Srivastava et al., 2014) which has been shown to be effective for regularising deep neural networks. The hyper-parameter for dropout 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. Intuitively, dropout works by forcing the remaining active neurons to infer what is missing in the activations that have been dropped and ultimately leads to better generalisations as activations are based on inference than on memory.

Table 6.1: Language Models comparison

Language Model	Train time	Perplexity	Epochs
GRU RNN 3-layer model (CPU training)	2 days	30.920	75
5-gram with Keysner Soothing and interpolation	5 minutes	238.720	N/A
GRU RNN single-layer model (GPU training)	5 hours	641491.52	120
Plain RNN single-layer model (GPU training)	9 hours	27087.893	180
GRU RNN 3-layer model (cloud GPU training)	2 hours	30.920	75

6.4 Language Model Training Experiments

Two sets of character-based neural network RNN-based experiments are developed in this chapter. A third word based statistically modelled language model is also developed based on Heafield et al. (2013) estimates as a baseline model. Character-based perplexity measurements were used to compare the character-based models and a conversion factor based on (Hwang & Sung, 2017) is used to compare character-based models on the word-based counterparts. Experiments for the RNN language models were majorly performed using tensorflow-MKL, which is a highly parallelised (44-threads for one of the experiments) cpu-based experiments. The experiment with the largest number of neurons was also performed on cloud-based GPUs (Nvida Tesla T4). Details of the experiments carried out and resulting perplexity are shown in Table 6.1.

6.5 Output Language Model and Language Generation

The 3-layer network experiments were trained on both CPU and GPU configurations. Both were trained for 75 epochs, where an epoch indicates that the model has processed all of the training data. Recall that the model is trained until it is saturated. In other words, the model trains until the prediction accuracy is no longer improving. This usually will take several epochs.

The loss plots for the three-layer and single layer GRU RNN are shown in Figure 6.1. The three-layer GRU-RNN achieved a prediction accuracy of 65% on held-out data. When the created 3-layer GRU character-based RNN language model is

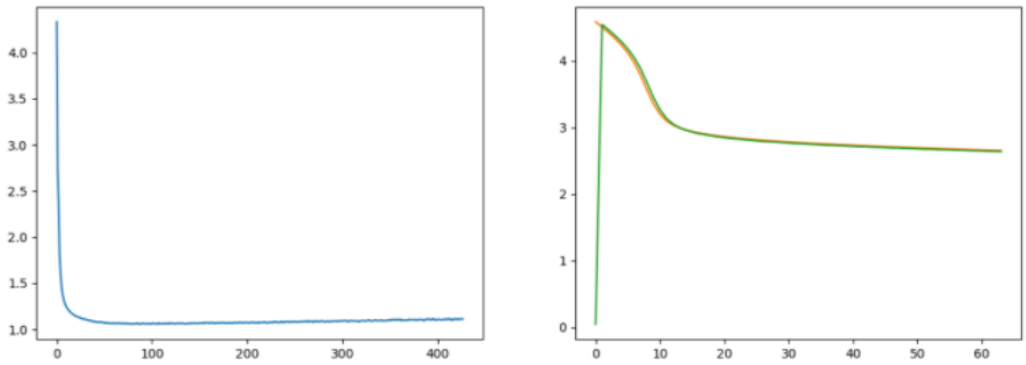


Figure 6.1: Wakirike Language model training Loss curves for (a) 3-Layer GRU and (b) Single-Layer RNN

seeded with an input character, one can force the network to 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 constructing its own sentences. The generated language output from the GRU language model was found to be a reflection of the overall context of 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:

$$PP(W) = P(w_1, w_2 \dots w_N)^{\frac{1}{N}} \tag{6.2}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{6.3}$$

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.

Intuitively, the perplexity metric measures the ability of a language model to predict held-out data. A character based perplexity metric is possible using the negative log likelihood of the character sequence. A character based perplexity

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

$$PP(X) = exp\left\{\frac{\sum_{t=1}^{T} \log P(x_t|x_{1:t1})}{T}\right\} \tag{6.4}$$

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 re-scored by re-estimating with a 5-gram word-based language model

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.

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 (Chen & Goodman, 1996) for various lengths of the held-out data.

6.6 Discussion

The result of the training of the 3-layer-deep GRU-Cell Recurrent Neural Network on low-resourced Wakirike Language had decent results with an accuracy of 65%. Even with this accuracy, the model had shown to have learned the vocabulary and was able to construct phrases consistent with the Wakirike language structure. The single-layer GRU cell however after 120 epochs did not learn the vocabulary and was not able to learn any words. It can also be seen from the loss curves of the language models that the models reach saturation fairly quickly after about 20 epochs for the 3-layer GRU RNN model and after about 40 epochs for the single-layer RNN.

The results also showed that the 3-layer GRU language model developed a better language model than the 5-gram model in terms of the perplexity metric because the perplexity of the 3-layer GRU RNN model was lower than that of the 5 gram model. The single layer GRU model being a shallow model with a single-layer did

Table 6.2: Language Models sample generation

a) Original Wakirike Text
mine-o anikanika boro sobie korobo enjelapu so, we duko o piri sa ibiok-wein mi sikima be jinye dukobia bo, ya tamuno worinime sime inibo piri wa tatari duko borosam, a piki mioku bari ani dukoabe na nemikase tomonibo
b) GRU RNN 3-layer model (75 epochs)
ani se mi be chinmgbolu mi ani se chua yee anisiki ini tamuno be bu s-arame, se nwo beme, a kokomaye duko o piriabe, o bi se mi mieari ye mi ori oria koki a kuro mi nyana yee. o bi bara mi o nwose o diebia ani
c) Plain RNN single-layer model (180 epochs)
min on o o bo oeuemin on o oniaia a bire nami bieee mani o onuo o be bo oe berimini okuma ani mani o o onuaminiana bireme,eanaminianiania b-i bo ono bo onia anaa beremanaa bi nao sike,einama nieiei mi niei ia
d) GRU RNN single-layer model (120 epochs)
ia iiiii o i ii i i iiii iii i oi oi o oiai oi ii ii ii iiiii i iiii iiiiii iii iii iii ii iii o oi i i o ii iiii iiii iiiii i ii i iii i o ii oi oia oi iiiiii o i o o o i oi o oi iiiii i iiiii ii

not however learn anything having a very high perplexity heading towards infinity. Table 6.2 below demonstrates output generations from the single layer and 3-layer GRU RNN language models based on the sampling procedure described in Section 6.5

Table 6.2(a) to (d) demonstrates how the language generated by various language models resembles the original training data. The original Wakirike language is given in part (a) then, the other language model generations (following the procedure described in Section 6.5) are shown in order of decreasing similarity. It can be, therefore, observed that the 3-layer GRU language model had the highest similarity to the original Wakirike language, which has also been evidenced by its low perplexity score. The other single-layer models having higher perplexity scores showed lesser degrees of similarity to the original Wakirike language.

Finally, it would be in the interest of this research to further consider increasing the number of layers to achieve higher levels of accuracy or consider optimising other parameters which may help the training results, such as the sequence length of the model. However, this was not done as the time constraint of one day for training the language model was already stretched and we are certain that these

hyper-parameters have a direct influence on the size of the model. This in turn will affect the computing resources required and hence a trade-off of the training time required to saturate the models.

6.7 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

Throughout the development of this thesis, the establishment of deep learning as a strategy where computers learn through representation of patterns at varying degrees of complexity has been an underlying theme. It was also emphasised how this is achieved by internal layer-wise encapsulations. Structures discussed in Chapter 2, such as layer-wise stacking of neural network type architectures such as the Restricted Boltzmann Machine (RBM) and Deep Belief Network (DBN) were used to implement such representations.

In this chapter, the end-to-end Bi-directional Recurrent Neural Network model is described. Bi-RNN for speech recognition tasks is employed here as opposed to regular RNNs or DBNs mentioned above in the preceding paragraph. Bi-RNNs are used because of the contextual nature of speech. In Chapter 6 it was demonstrated how deep stacking of GRUs outperform single-layer RNNs for extended sequences. That is to say, words in a sentence or paragraph are contextual to the sentence/paragraph over particularly long sequences and, these word contexts are better captured by the GRU architecture. More importantly, Bi-RNN's have a forward and backward RNN and these give the neural network the ability to analyse (look-up) the words from the backward RNN not currently seen by the forward RNN in the sentence succinctly giving the BiRNN parameters a contextual feature (Graves et al.,

2006).

In addition to the procedure for designing sequence-to-sequence RNNs outlined in Section 3.4.1, this Chapter describes the training data, data preprocessing, derivation of feature vectors and output decoding. First, speech features developed by making use of the deep scattering convolution networks DSN is discussed. The DSNs are used as inputs to the end-to-end model. Two end-to-end networks are then described. The core Bi-RNN network and a second Bi-RNN network augmented with an RNN-transducer and an attention mechanism. A formal presentation of the speech neural network model parameters and architecture is given and the decoding algorithm is also detailed in sections contained within this chapter. Finally, the results are presented and the findings from the model results discussed.

7.1 Deep Scattering Features

In Chapter 4, we derived a fast wavelet transform from a low pass filter and a high pass filter. The speech features used for the BiRNN is obtained from successive wavelet-modulus operations of a deep scattering network 2 layers deep. This 2-layer DSN comprises a first-order scatter transform. The wavelet modulus operator is derived from the combination of 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 zeroth scattering layer the Q-band parameter was $Q = 8$ and the first 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.

For the second end-to-end architecture involving a transducer with attention mechanism, a period of is used to capture a window of 4 seconds for audio signals sampled at 16000Hz. The same Q-band parameters having Q=8 for the zeroth layer and Q=1 for successive layers are used, In addition, the total number of layers deep was M=3 giving rise to a 2nd-order Scatter Network. This produced a feature-vector having 250 dimensions.

7.2 CTC-BiRNN Architecture

The CTC-Bi-RNN sequence model design follows the synchronous MIMO design described in Section 3.4. As a result of the CTC-decoder implementation in Section 7.2.1, however, the decoder converts Bi-RNN model from a synchronous MIMO to an asynchronous one.

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, $x_{t,p}^{(i)}$ 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 (*Mozilla Deepspeech*, 2019) CTC bi-directional RNN implementation along with the accompanying Mozilla Common voice data set (Ardila et al., 2019). The Common Voice Data set project consists of voice samples in short recordings approximately 4 seconds each. The complete data set 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 A. Hannun et al. (2014). 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 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.

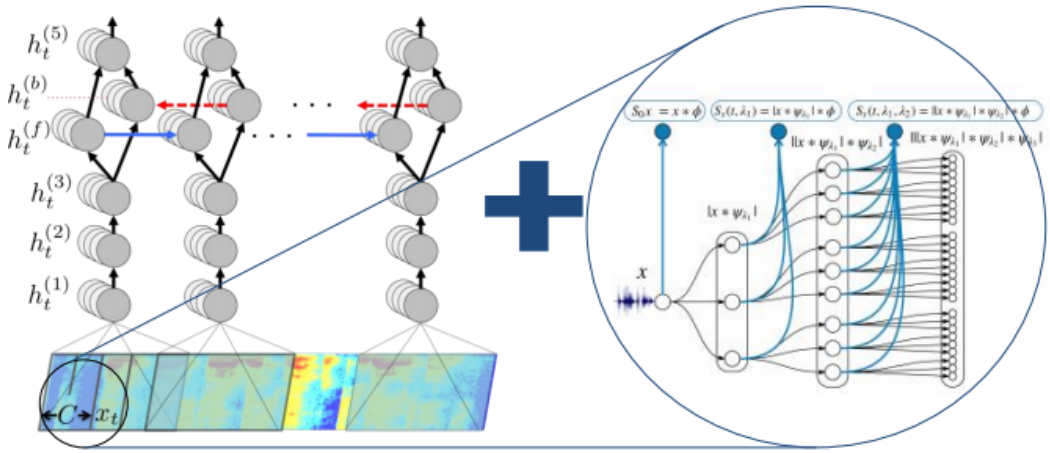


Figure 7.1: Deep scattering Speech Model architecture reveals the 5-hidden layer Bi-RNN $h_t^{(1)}$ to $h_t^{(5)}$ being trained by DSN features.

7.2.1 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 A. Y. Hannun et al. (2014) 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 $W(\ell)$.

7.2.2 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 & 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 7.3 shows word error rates for various GPU configurations and audio data-set sizes.

7.3 Summary of Bi-RNN Experiment Design

The details of the Bi-RNN model has been outlined in Sections 7.2 and 7.2.1. This section now summarises the process of the Bi-RNN experiment from data collection to output text transcriptions. Recall once again, as this is an end-to-end experiment the input-end will comprise raw audio speech utterances and at the output-end will be the character sequences which are resolved into words using a language model. The design of this experiment therefore utilises the Bi-RNN to ingest raw audio utterances as preprocessed scatter-transforms and outputs text transcription which can be compared against the original audio transcriptions.

The audio clips and the corresponding transcriptions are downloaded with a script and the subsequent locations are stored into a configuration file. Being an end to end process, no further data pre-processing is required except conversion of the audio file from a binary format to a numeric-text format. From this numeric text format, the scatter-transforms are computed when loaded from the configuration file.

The CTC-algorithm discussed in Sections 4.3.3 and 7.2.1 is responsible for correcting fuzzy alignments between audio input and output text. This relationship according to Section 3.4.1 is an asynchronous MIMO, but the Bi-RNN represents a synchronous MIMO. Hence a synchronous MIMO is combined with the CTC-decoder such that while Bi-RNN takes care of the Multiple Input Multiple Output relationship the CTC decoder takes care of collapsing output characters and, therefore, restoring the asynchronous relationship between outputs and inputs. For steps 1 and 2 of Section 3.4.1 we use a Bi-RNN. Details of the internal structure of the Bi-RNN from Sections 7.2 and 7.2.1 are used for step 3. This includes 3 hidden regular DNN layers and two bi-directional LSTMs hidden-layers and a softmax output layer. Neural network saturation parameters are selected based on desired training time, similar research practice, and accuracy expectations in line with the research

objectives. They include weight initialization having a mean of 0 and standard deviation of 0.046875; clipped RELU non-linear function (see Chapter 5); learning rate of 0.001 with 0.95 momentum. Finally, the adam optimiser and CTC-loss function (Section 4.3.3) are incorporated at the training output stages.

The CTC decoder described in Section 7.2.1 is then used to determine the characters from the softmax output. The CTC decoder had a look ahead beam search parameter of 200 characters.

7.4 BiRNN with Attention Transducer end-to-end Architecture

The core of this model is a CTC-Transformer+Attention Transducer model. Together these two architectures achieve joint speech training and decoding. The CTC-Transformer model is based on a Bi-LSTM similar to what is obtainable in the DeepSpeech model. There are up to 11 variants of Attention networks implemented in ESPNet, however, the results of the experiments done this work experiment was determined from the attention model described in Chorowski, Bahdanau, Serdyuk, Cho, and Bengio (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 20-200 epochs depending on the design goals and accuracy required.

This model uses the asynchronous MIMO model. Although this model would require more neural network layers and about 2 times more RNN units than the synchronous MIMO-RNN model, ultimately, addition of Attention-based models ensures faster convergence and ultimately faster time to train. This can only be achieved using asynchronous MIMO models which are the only models that support attention-mechanism.

Using a weighting function, α, 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.

$$\mathcal{L} = \alpha\mathcal{L}^{ctc} + (1-\alpha)L^{att} \tag{7.10}$$

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

$$\log p(y_n|y_{1:n1}, \mathbf{h}_{1:T^{'}}) = \log p^{hyp}(y_n|y_{1:n1}, \mathbf{h}_{1:T^{'}}) + \beta \log p^{lm}(y_n|y_{1:n1}) \tag{7.11}$$

Where joint decoding, $\log p^{hyp}(y_n|y_{1:n1}, \mathbf{h}_{1:T^{'}})$ is given by

$$\log p^{hyp}(y_n|y_{1:n1}, \mathbf{h}_{1:T^{'}}) = \alpha \log p^{ctc}(y_n|y_{1:n-1}, \mathbf{h}_{1:T'}) + (1-\alpha) \log p^{att}(y_n|y_{1:n-1}, \mathbf{h}_{1:T'}) \tag{7.12}$$

7.5 Summary of birnn with Attention Transducer Experiment Design

According to the stepwise procedure of deriving RNN sequence-to-sequence models , the Bi-RNN with Attention Transducer model incorporates an asynchronous MIMO design for Step 1 (Section 3.4.1). For steps 2 and 3 (Section 3.4.1), 6 layers similar to the Bi-RNN-only design consisting 3 regular DNN layers, 2 BLSTM and the output softmax layer. Neural network components The number of neurons for the hidden layers were 2048 neurons. Network saturation parameters were similar to the Bi-RNN-only experiment setup having the number of epochs between 20-200 epochs and in line with research objectives. As the training was significantly faster in this setup, more epochs could be incorporated into the experiments. The initial experiment had only 20 epochs using baseline experiments and to determine how fast the training would be. Subsequent experiments were between 100 and 200 epochs according to desired accuracy and research objectives. Weights were initialized between 1 and -1 according to a normal distribution. Learning rate was 0.001; Non linear function was a clipped RELU. Cost function and optimizer was ctc-loss and ada-grad variant. Decoding was done according to the joint decoding function described in Section 7.4. Finally, a drop out of 10% was used to avoid over fitting of the model.

7.6 Speech Model Baselines

The model baselines were trained alongside their scatter transform counterparts. In addition, we adopted the model produced by the Mozilla DeepSpeech team. This model had a similar architecture with 5 hidden units and 2048 hidden units. This baseline was trained on Librespeech corpus and the common voice data corpora (Ardila et al., 2019; Panayotov, Chen, Povey, & Khudanpur, 2015). Study by A. Y. Hannun et al. (2014) 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 their study had 5 hidden layers having either 1,824 or 2,048 hidden units in each hidden layer.

Word Error Rates obtained by this additional 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 corpora.

7.7 Speech Model Simulations

Speech model training experiments were carried out on the two different end-to-end models as well as on different GPU configurations. The first set of experiments was performed for the Bi-RNN-only model. The first GPU configuration for the Bi-RNN-only model consisted of 2 GPUs having a total of 10 gigabytes of memory. The second GPU configuration comprised 5 GPUs having a total of 15 gigabytes of memory. Experiments for the BiRNN end-to-end model with transducer and attention were also performed using a GPU configuration having 4 gigabytes of memory.

7.7.1 Bi-RNN-only end-to-end model Experiments

For the Bi-RNN-only end-to-end experiments, GPU configuration experiments were carried out on varying-size subsets of the common voice corpus. The various GPU

Table 7.1: Bi-RNN-only Experiments

Experiment	Hours of speech	Total training time	Training status
1. 2xGPU 10GB RAM	1	7 days	Complete
2. 2xGPU 10GB RAM	10	40 days	Not complete
3. 5xGPU 15GB RAM	2	2 days	Complete
4. 5xGPU 15GB RAM	40	40 days	Not complete
5. 1xCPU 16GB RAM	20	20 days	Not complete
6. 1xGPU 2GB RAM	20	20 days	Not complete

Table 7.2: Bi-RNN-only Experiments Summary

Experiment	Hours of speech	Corpus	epochs	Metric	Score
1. 2xGPU 10GB RAM	1	CV LVCSR	40	WER(%)	100+
2. 2xGPU 10GB RAM	10	CV LVCSR	25	WER(%)	100+
3. 5xGPU 15GB RAM	2	CV LVCSR	40	WER(%)	100
4. 5xGPU 15GB RAM	40	CV LVCSR	40	WER(%)	87

configurations along with the training times are shown in Table 7.1.

It can be seen in Table 7.1, only two experiments had reached completion. The others had to be stopped for exceeding reasonable training times of 20 and 40 days. Out of the four experiments that did not complete, all the GPU-based experiments had trained for up to 20 epochs and quantitative metrics were taken for these experiments. Table 7.2 shows the details for the Word Error Rate (WER) accuracy metrics for a total of four experiments. The number of hours of speech, corpus type and total number of epochs are also shown. Accuracy curves are shown in figure 7.3.

7.7.2 Bi-RNN with Attention Transducer Experiments

The End-to-End Speech Neural Network Toolkit (ESPNet) (S. Watanabe et al., 2018) provides building blocks for BLSTM transducer with attention mechanism described in Section 7.4. Two experiments involving a much smaller audio corpus guaranteed to converge quickly at training and a larger Italian language speech corpus (*Voxforge*, 2019) 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 speech corpora utterances are 16-bit linearly sampled at 16kHz, each record-

Table 7.3: Bi-RNN with attention and transducer Experiments

Experiment	Hours of speech	Training time	Epochs	Training status
1. 1xGPU 4GB (log mel.)	1	15 minutes	20	Complete
2. 1xCPU 16GB (scatter feat)	1	3 days	100	Complete
3. 1xGPU 4GB (log mel)	10	11 hours	200	Complete
4. 1xGPU 4GB (scatter feat)	10	38 hours	200	Complete

Table 7.4: Bi-RNN with attention and transducer Experiments Summary

Experiment	Hours of speech	Corpus	Metric	Score
1. 1xGPU 4GB (log mel)	1	AN4 SVCSR	WER(%)	12.9
2. 1xGPU 4GB (scatter feat.)	1	AN4 SVCSR	WER(%)	26.8
3. 1xGPU 4GB (scatter feat)	10	Voxforge-italian (LVSCR)	WER(%)	76.7
4. 1xGPU 4GB (log mel)	10	Voxforge-italian (LVSCR)	WER(%)	72.4

ing 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.

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 BLSTM and is similar to what is obtainable in the DeepSpeech model. 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.

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.

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 Model Results Interpretation

Interpretation of Model Results Experiments carried out to train the end-to-end ASR were performed on the following system configurations

i. GPU (GTX1050) with 2GB RAM

ii. GPU (GTX1050) with 4GB RAM

iii. GPU (GTX1060) with 6GB RAM

iv. GPU (GTX1070) with 3GB RAM

v. CPU with 16GB RAM

7.8.1 Bi-RNN-only experiment discussion

Configurations with CPU were used as control experiments to compare the GPU efficiency with the CPU being compensated with more memory. The higher memory allowed the CPU configurations to remain accessible during training unlike the GPU systems that used up most of the system resources bringing the computer system close to a grinding halt and making the GPU systems difficult to access while training was in progress. In as much as a number of the experiments done exceeded 10 days, our goal was not to exceed more than 10 days training for speech models. What the GPU lacked in memory resources was compensated for in computational speed gained due to their capacity for parallel. By changing the batch size, memory resource requirement and computational parallelism was simultaneously managed for all experiments. Therefore for CPU training computational speedup was attempted by increasing the batch size and for GPU training batch sizes were reduced so as not to quickly deplete the small memories. For Bi-RNN-only experiments, regardless of the batchsize allocations, only 2GPU configurations (1 and 3 in Table 7.1) completed training for the given amounts of epochs and training data.

Table 7.2 shows four GPU-only configurations. These GPU training configuration experiments had completed at least 20 epochs. Training metrics for these configurations are plotted in Figure 7.3. A reduction in training loss is observed once the data was increased to two hours of training. This gives an indication of

the model learning on the amount of data given. Even though the speech models were trained on English language only. We can simulate low resource settings in the English language by limiting the amount of data available during training. Moreover, word error rates (WER) only showed improvement on the 40 hours data set. This also indicates 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

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. Initial experiments were performed on single GPU Units. Batch size settings for these experiments were very small to fit into the limited RAM sizes on the GPUs. At a later stage, multi-GPU units were utilised as a strategy to speed up training by increasing the batch sizes to run on the combined memory. This however did not result in the anticipated speed up. It is suspected that this outcome may have been as a result of latency copying model parameters between GPU units and CPU multiple times during training.

7.8.2 Bi-RNN with Transducer and attention mechanism experiment discussion

Results from Bi-RNN with Transducer and attention experiments had shown greater promise in terms of completion of training within the time constraints set. Results shown in table 7.3 show that both CPU and GPU training completed training for less than 20 hours of training and total number of epochs.

We used a Small Vocaluary Continusous Speech Recognition (SVCSR) corpus of English language and a Large Vocaluary Continusous Speech Recognition (LVCSR) of Italian and achieved a decent score of 26.8% for the SVCSR corpus and a high error score of 76.7% for the LVCSR corpus. The high error score for the LVCSR from the observed results was attributed to the fact that the amount of data given 10 hours is not sufficient for meaningful convergence. This is also evidenced by the baseline result having a similar high error rate of 72.4%. A similar effect was also observed in the Bi-RNN-only experiments such that after training of 40 hours of data for 40 epochs the error was at 87%.

From the training curves (Figures 7.4 and 7.3 however, we can see that at 40 epochs the Bi-RNN with transducer and attention mechanism experiments had a faster rate of convergence and this led to experiments being completed within the time limits set.

7.9 Chapter Summary

In this chapter the details of the combination of an end-to-end deep bi-RNN architecture and deep scattering features were elaborated on. The architecture described 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. A 163-dimension 1st-order feature vector of deep-scattering encoding derived from a sampled raw audio file is fed into this network.

A second set of experiments comprising a similar architecture containing a Bi-LSTM this time with encoder and decoder architectures and a transducer is also developed and tested.

The result showed the second set of experiments having the transducer architecture with attention mechanism are able to train faster but out of all the results, although some models came close to their respective baselines none actually performed better than the baseline models. In Chapter 8 we address ways that the results may be improved.

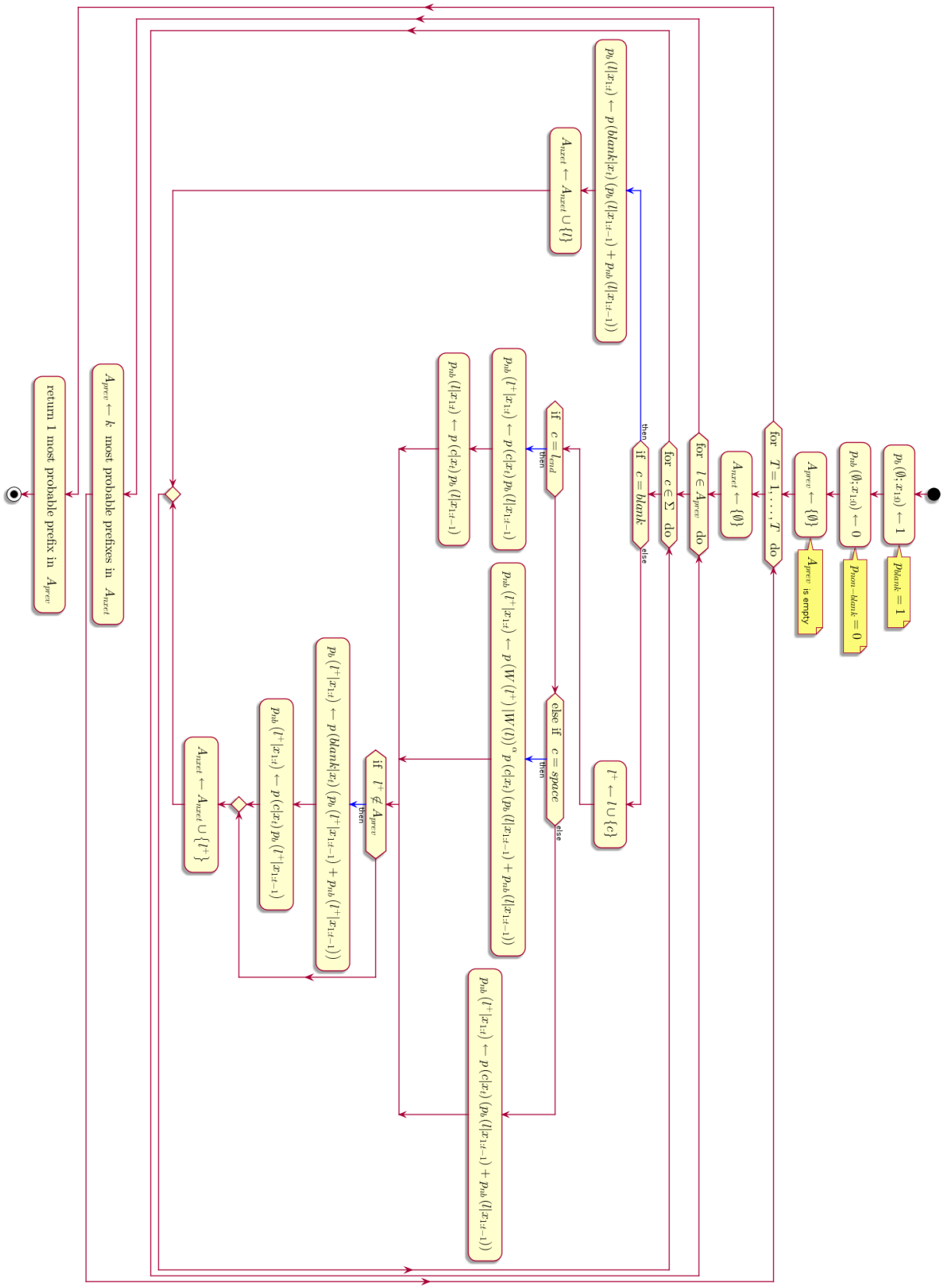


Figure 7.2: Prefix beam search algorithm

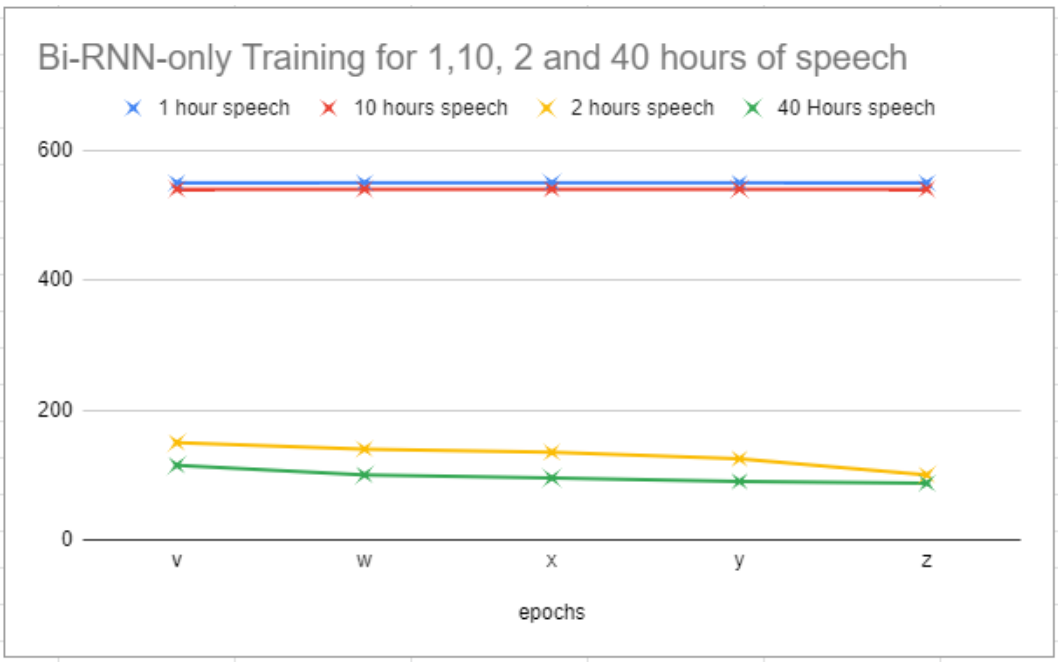


Figure 7.3: Bi-RNN-only Experiments Error curve, where $w < x < y < z$ are taken arbitrarily across the total number of epochs

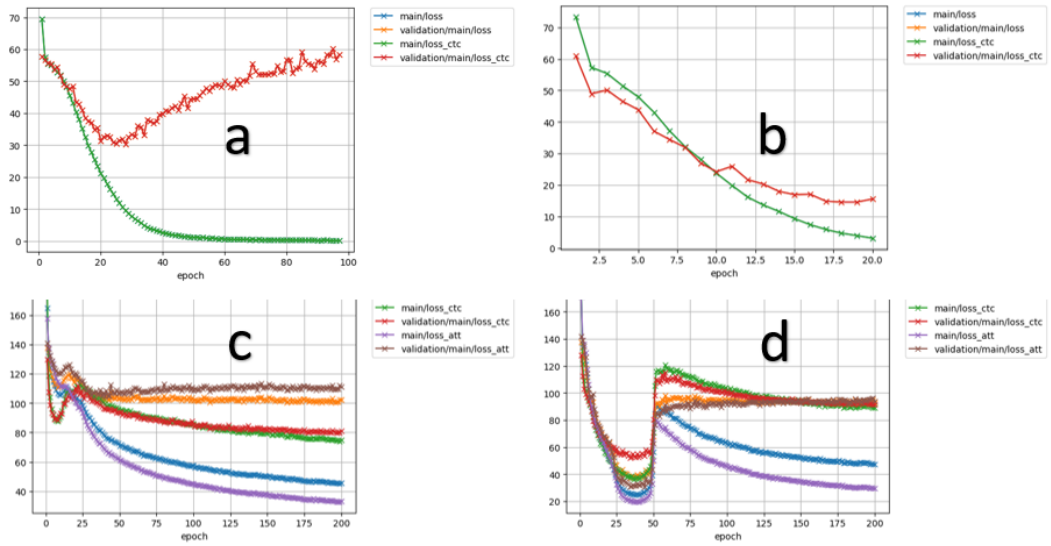


Figure 7.4: birnn with Attention Transducer Training Loss: (a) and (d) with 250-dimension scatter transform features; (b) and (c) with 83-dimension Log-Mel features

Chapter 8

Conclusion and Future Work

This research has been inspired by the notion of a Language Learning Companion (LLC). One of the artificial intelligence tasks for LLC is Automatic Speech Recognition (ASR). This research, therefore, explored the current advancements that have been made in the field of ASR and discovered that one of the major challenges in the field of ASR is that under the most robust and current methods, new and low resourced languages are unable to integrate ASR systems without a great deal of effort and rigorous techniques. In particular, considering ASR as a machine learning pipeline, this research favours a discriminative approach to ASR over generative approaches. Although hybrid HMM-DNN, the hmm component which is considered a generative model, does deliver robust results, the ASR pipeline involves training of individual components of the pipeline in cumbersome and separate processes. The alternative end-to-end approach, however, attempts to simplify this by offering a solution that involves training of a single discriminative "end-to-end" model.

Since ASR is a machine learning pipeline, the advancement of Machine Learning has had 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. This research examined the effect of using a different set of input features from the state-of-the-art log-mel features. Feeding these scatter transform features into end-to-end ASR DNN with the objective of a comparable fast and efficient speech recognition for new and low resource languages. The major advantage of our system is that in addition to robust end-to-end models, the end-to-end system is trained with unique front-end scatter transform features. Moreover, the intrinsic integration of a character-based language model help to train low-resource languages for ASR in a resource efficient manner.

This Chapter discusses the outcome of the end-to-end and deep sequential models engineered towards low-resource speech recognition developed by this research. In addition, additional models being developed in the research community which are closely related models are mentioned as areas of further research interest. These include Generative Adversarial Networks (GANs) and Self-Attention-based models.

8.1 Discussion of Research Output models

Two well established key aspects of ASR from the ASR formulation in Equation 2.3 include an Acoustic Model (AM) and a Language Model (LM). The objective of this research was to understand these key aspects of ASR systems and develop ASR systems that can be accessible by new and or low resource languages.

The research objectives were met by developing speech models based on a neural network end-to-end approach. End-to-end discriminative neural network speech models have now become a well established method in Automatic Speech Recognition. The Bi-directional Recurrent Neural Network (Bi-RNN) end-to-end system developed in this work,is augmented by features derived from a deep scattering network as opposed to the standard Mel Frequency Cepstral Coefficients (MFCC) features used frequently in current acoustic models. These specialised deep scattering features consumed by the Bi-RNN model represent a light-weight convolution model.

On their own, the end-to-end models do achieve decent results. The results however, can be improved by the inclusion of a language model. This work implemented a 5-gram n-gram language model and a character-based recurrent neural network

language model (RNNLM) for the Wakirike and English languages and the latter was incorporated into some of the speech models developed in this research.

8.1.1 Main contribution to knowledge

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 therefore demonstrates that Deep Scattering features derived from wavelet filter operations on audio data can produce viable candidates for end-to-end training of Automatic speech recognition models. Preliminary experiments showed word error rates were not too far from the baselines having achieved 26.8% WER (14% points from the baseline) for SVCSR and 76.7% for LVCSR which was 4% points from the baseline.

8.1.2 Summary of goals achieved in this work

The aspects of the Zero-resource speech recognition challenge (Versteegh et al., 2015) that mirror the Language Learning Companion motivation for this work seek ASR systems that model sub-word language and word/super-word language constructs. This research developed models that met these goals.

Initial pilot studies for this work involved feature engineering of speech signals using correlation (Section 3.3.1). These could be viewed as a precursor to the scattering transform since wavelet operations are related to correlation. Other pilot experiments developed various sequence-to-sequence models for sub-word ASR system modeling, including a grapheme-to-phoneme model trained in Section 3.4.3. From this model, a phonetic dictionary for the Wakirike language was obtained. An ASR system post-processor was also developed based on a sub-word sequence-to-sequence RNN model (Section 3.4.2). This post-processor was to insert diacritic, tonal symbols into an ASR system output text. Although the sub-word systems built in this work brought insight to sequence modelling using Recurrent Neural Network (RNN)s, in practice, only the diacritic post-processing tool theoretically had direct application in the final research output. The Grapheme-to-Phoneme (G2P) tool did not have a direct application to the research output because the research output

end-to-end system removes the need of using G2P systems. The G2P tool would however be valuable for hybrid GMM-HMM/DNN ASR systems.

Some aspects of ASR system development covered in this work not directly related to Zero-speech challenge included that of speech segmentation and speech-to-text alignment (Section 3.3.3). These are pre-input-feature processing tasks which occur at the data preparation stage. Once again, although speech segmentation is not directly integrated into our end-to-end system, one of the huge benefits of end-to-end CTC-based systems implemented in our work for new/low-resource languages is the time saved from the need for accurately aligned speech during training. CTC-based end-to-end systems is able to achieve alignment based on roughly aligned inputs. Roughly aligned here means chunks of segmented raw speech audio and their text equivalent. Hence, alignment is only at the segmentation or utterance stage and no need to align at the word or sub-word level.

Finally, in addition to the speech model built using DSN features, in Chapter 6, a word-level language model was developed using a character-based, GRU-RNN language model. This model had an improved perplexity to the baseline language model which was based on a statistical 5-gram language model.

Unlike some of the sub-word systems which did not have direct impact on the final research outcome, the GRU-RNN language character based model developed by this research was integrated into some of the final speech models built in this work.

8.2 Limitations of the study

This research was able to develop ground-breaking end-to-end speech models for speech recognition. However, only preliminary results have been obtained for deep scattering features. A handful of ASR sequence models were developed for the Wakirike language. The Wakirike language, however, which motivated this research, did not have any end-to-end speech model built for it. It is the goal of this research, therefore, in a future study, to develop a speech corpora for the Wakirike language.

A more immediate limitation, was the high system capacity requirements for training Large Vocaluary Continusous Speech Recognition. The training of LVCSR

such as the full common voice data set requires high-end capacity systems and GPU configurations that were not accessible for training. Notwithstanding, considering the current configurations that were available, a number of improvements to the present models are possible; one of which asks for further investigation of first and second order scatter network features. Although first order and second order features were explored in this research, the number of experiments carried out were not sufficient to predict the significance of one over the other.

8.3 Directions for future study

Improvements in this work can be divided into immediate improvements and the longer term extensions for this study. Immediate improvements for the DSN speech features include the addition of Vocal Tract Length Normalisation (VTLN) and noise-robust speech enhancement. These feature engineering could immediately improve robustness of the ASR systems developed in this work. Other longer term extensions of this work include data augmentation and model optimisation methods. The following sections mention one feature enhancement technique and two model optimisation techniques that are possible extensions for this work.

8.3.1 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.

8.3.2 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 & 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.

While we have used attention-based encoder-decoder networks for training of features, considerable resource-saving may be gained by employing self-attention-based networks. This has been explored by Salazar, Kirchhoff, and Huang (2019) with satisfactory results.

8.3.3 Model Pre-training

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 20 days and should not generally exceed 10 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 (I. 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, Lee, and Mazeika (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 are 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, Wu, Liu, Yang, and Zhou (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.

8.4 Conclusion

The outcome of this research is an ASR system which facilitates fast and efficient speech recognition using the end-to-end speech recognition and deep scattering speech features. Another advantage of our ASR system was the intrinsic integration of a character-based language model. This enabled the developed ASR system satisfy both criteria of the low-resource challenge. The first being the top level word and sentence modelling seen in the character-based language model, and the second being the sub-word and acoustic modelling of input features seen in the deep scattering network features.

The word error rates obtained by our model with different data sizes and speech corpora was not as good as the features using log-mel features. However, the results were competitive. One limitation in this study was that data augmentation using Vocal Tract Length Normalisation was not done. This could be added as a precursor to the scattering transform. Furthermore, despite the fact that end-to-end models have been, until recently, dominated by RNN-sequence-models, the self-attention (Vaswani et al., 2017) architecture has shown a lot of promise over the machine learning landscape and a lucrative model for low-resource end-to-end speech recognition.

This research demonstrates sequence-to-sequence and end-to-end models as flexible and still relatively untapped effective tools in many aspects of ASR. By the development of sequence models at the phonetic and syntactic levels of comprehension and by the development of an end-to-end sequence model speech system, we provide examples for low-resource Wakirike and simulate low-resource languages by constraining rich-resource English and Italian language. Finally, this research recommends the minimum requirements for low resource languages desiring to utilise end-to-end sequential training in terms of computing resources and training data. For training, data should consist of 5-10 second utterances with transcribed speech and between 40-80 hours of speech data for bare minimum LVCSR languages and applications. For training system requirements, this work recommends CUDA[1] GPU training having at least 8GB RAM.

[1]Compute Unified Device Architecture (CUDA) is a Parallel processing library and trademark of Nvidia incorporated

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.

$$E = \int_{-\infty}^{\infty} |x|^2 dt = ||x(t)||^2 = 0 \tag{1}$$

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:

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

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:

$$\psi(t) = \phi(2t) + \phi(2t - 1) \tag{3}$$

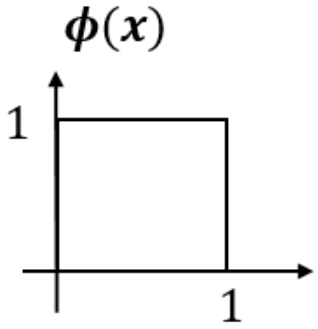
Observe here that $c_0 + c_1 = 2$, i.e. dyadic. The solution to this recurrence equation and the resulting plot is given in Figure 1.

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

$$\phi_{j,k}[n] = 2^{j/2} \phi[2^j n - k] \tag{4}$$

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

$$\phi(x) = \begin{cases} 1 & 0 \le x \le 1 \\ 0 & otherwise \end{cases}$$



$$\psi(x) = \begin{cases} 1 & 0 \le x < \frac{1}{2} \\ -1 & \frac{1}{2} \le x < 1 \\ 0 & otherwise \end{cases}$$

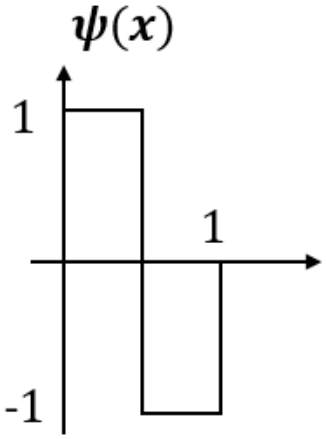


Figure 1: Haar wavelet

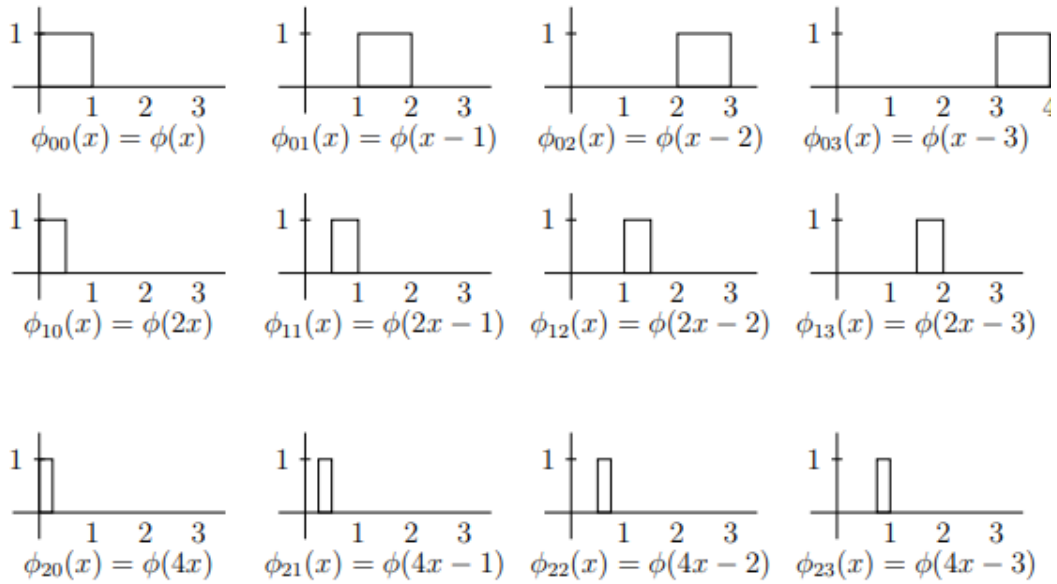


Figure 2: Multi resolution analysis of Haar wavelets

Appendix II - Gabor and Morlet Wavelets

Gabor Wavelet filter

The 1D-Gabor wavelet is defined spatially by

$$\psi(x) = exp\left(-\frac{x^2}{2\sigma^2} + i\xi x\right) \tag{5}$$

The Fourier-transform is

$$\hat{\psi}(\omega) = exp\left(-\frac{\sigma^2(\omega-\xi)^2}{2}\right) \tag{6}$$

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

$$\xi\sigma = \sqrt{-2\log(\hat{\psi}(0))} \tag{7}$$

The wavelets are therefore computed as

$$\psi_j = \frac{1}{a^j\sigma}\left(\frac{x}{a^j}\right) \tag{8}$$

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.

$$\frac{\xi}{a} + \frac{g^{-1}(\tau)}{a\sigma} = \xi - \frac{g^{-1}(\tau)}{\sigma} \tag{9}$$

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

$$\xi\sigma = \sqrt{-2\log(\tau)}\frac{a+1}{a-1} \tag{10}$$

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

$$\xi = 3\pi/4 \tag{11}$$

Equations (7 and 10) 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

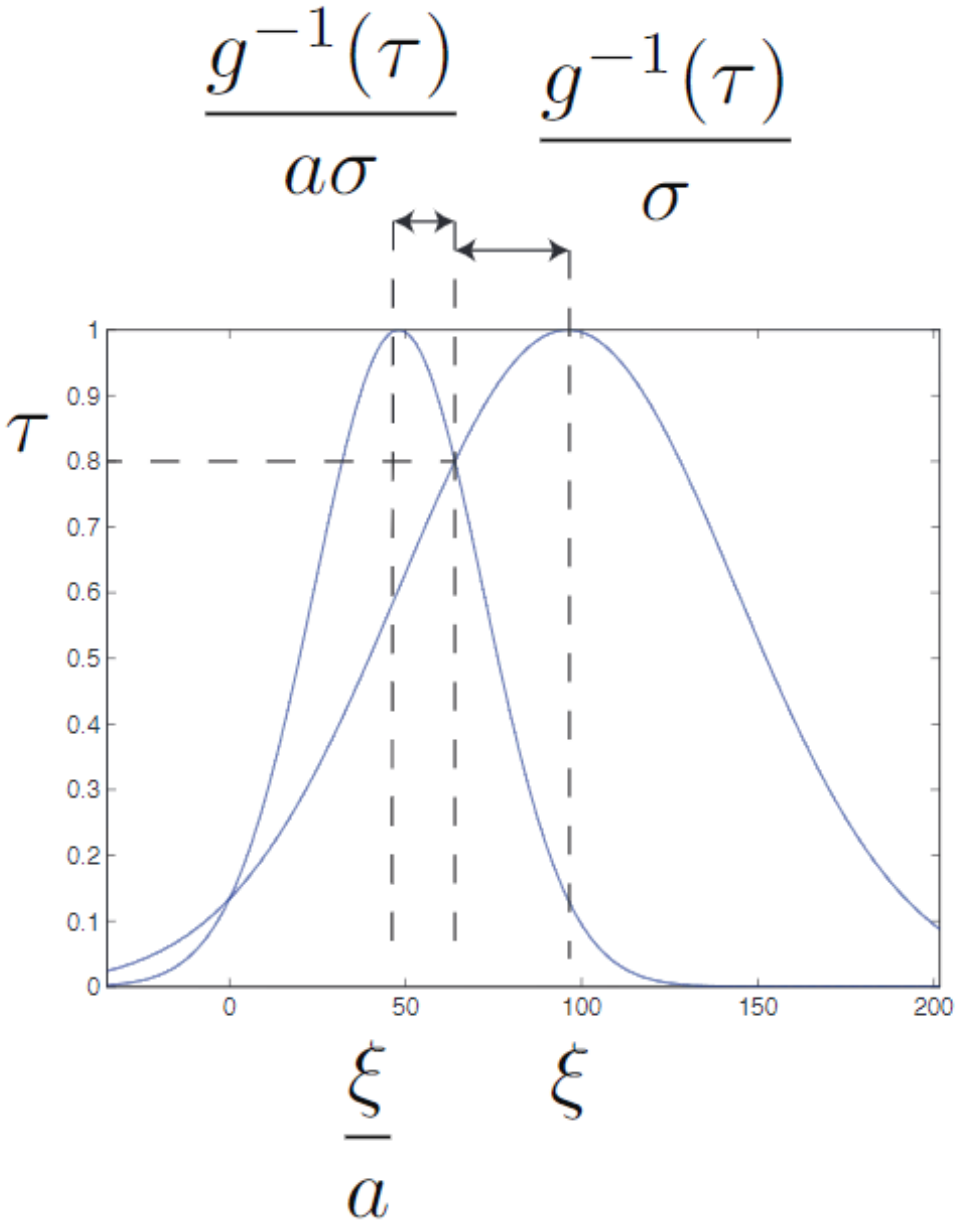


Figure 3: Fourier transform of adjacent scale Gabor wavelet. τ has been set to 0.8

With these requirements met we see that

$$\hat{\psi}(0) = \tau^{\frac{(a+1)^2}{(a-1)^2}} \tag{12}$$

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

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

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:

$$\phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)(\exp(i\xi x) - exp(-\sigma^{22}/2)) \tag{14}$$

It's Fourier transform is

$$\hat{\psi}(\omega) = \omega \exp\left(-\frac{\sigma^2(\omega-\xi)^2}{2}\right) - \exp\left(-\frac{\sigma^2(\omega+\xi)^2}{2}\right)\Big) \tag{15}$$

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

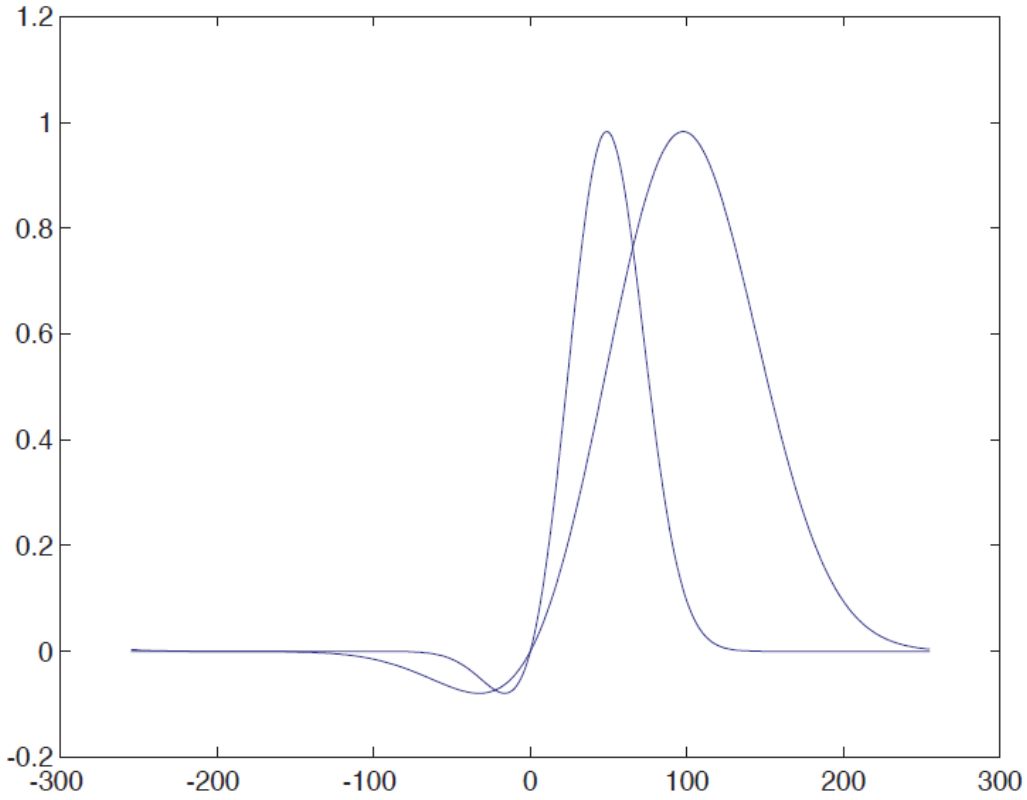


Figure 4: Fourier transform of adjacent scale Gabor wavelet. τ has been set to 0.8

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..
        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)
```

Appendix V - Wakirike Phoneme dictionary

dieme DD I E M EH
a A2
abaija A B A IH DZ A
abiud A B I U D
abu A2 BB UH
agbamieapu A GB A M IH EH A P UH
aizaya A IH Z A Y A
akilos A K IH L OH S
akim A K IH M
akpaka A KP A K A
aku A K UH
ama A M A
amanyana A M A N Y A N A
amanyanabo A M A N Y A N A BB OH
amanyanakiri A M A N Y A N A K I R I
amaogbo A M A O GB O
aminadab A M IH N A D A B
amon A M OH N
anga A N G A
ani A0 N IH
anianga A N IH A N G A
aniatibi A N IH A T IH B IH
anierechi A N IH EH R EH TS IH
anieso A N IH E S O
anisiki A N IH S IH K IH
apala A P A L A
ari A0 R I
aria A R IH A
asa A S A
ateli A T E L I
awo A W OH
awomeni A W OH M EH N IH
azo A Z OH
ba BB A
baasam BB A A S A M
babia BB A BB IH A
babilon B A B I L OH N
bakama B A K A M A

balafa BB A L A F A
balafame B A L A F A M EH
bara BB A R A
barasin BB A R A S I N
barasinsam BB A R A S I N S A M
bari BB A R IH
be B EH
bebe BB EH BB EH
bebia BB EH BB IH A
bebo BB EH BB OH
bee BB E EH
beinmabia BB E I N M A BB IH A
beleme BB EH L EH M EH
beme BB EH M EH
bere BB EH R EH
berenime BB E R E N I M EH
berijineme BB E R I DZ I N E M EH
betlihem B EH T L I H EH M
bia BB IH A
biari BB I A R IH
bie BB I EH
biebele BB I E BB EH L EH
biejudaboe BB I E DZ U DD A BB OH EH
bike BB I K EH
bin B I N
bipi BB IH P IH
birikunme BB I R I K U N M EH
birikunye BB I R I K U N Y E
bo BB OH
boaz B O A Z
boe BB O EH
boima BB OH IH M A
bokateinke BB O K A T EH IH N K EH
boko BB OH K OH
boloka BB OH L OH K A
bome BB O M EH
boro BB OH R OH
boroma BB OH R OH M A
bu BB UH
bubalabala BB UH BB A L A BB A L A
buchuaye BB UH TS UH A Y E
bugbeinke BB UH GB EH IH N K EH
bugererema BB UH G E R E R E M A
bukuroma BB UH K UH R OH M A
buo BB UH OH
buosome BB U O S OH M EH
buru BB U R U
chie TS IH EH

chin TS IH N
chinbia TS IH N BB IH A
chingbolu TS IH N GB OH L UH
chingi TS IH N G IH
chinme TS IH N M EH
chinmgbolu TS IH N M GB OH L UH
chuabia TS UH A BB IH A
chuaka TS UH A K A
da D A
dabe D A BB EH
dachieme D A TS IH EH M EH
dadiki DD A DD I K I
dadikibia DD A DD I K I BB IH A
damabo D A M A BB O
damaboari D A M A BB O A R IH
damamumma D A M A M U N M A
datekeremabia D A T E K E R E M A BB IH A
dawo DD A W OH
dawoju DD A W OH DZ U
dawonemiapu DD A W OH N E M I A P UH
dawu DD A W U
de D E
deki D E K I
devid D E V I D
die DD I E
dieapu DD I E A P UH
diepakuma DD I E P A K UH M A
diki DD I K I
din DD I N
dina DD I N A
dinma D I N M A
doki DD OH K IH
dokiapu DD OH K IH A P UH
dokibia DD OH K IH BB IH A
dokibo DD OH K IH BB O
dokibosa DD OH K IH BB O S A
dokimabia D OH K IH M A BB IH A
dokime DD OH K IH M EH
duaboromabia DD UH A BB OH R OH M A BB IH A
duko DD U K O
dukoari DD U K O A R IH
dukome DD U K O M EH
dukuke DD UH K UH K EH
dupari DD UH P A R IH
ebraham E B R A H A M
ehaz E H A Z
eka EH K A
ele E L E

```
eliakim E L IH A K IH M
elieza E L I E Z A
eliud E L I U D
ene E N E
enekakaa E N E K A K A A
enjelbo E N DZ EH L BB OH
epele E P E L E
erechi EH R EH TS IH
eremenitoku E R E M E N I T OH K UH
farisiapu F A R I S I A P UH
fi F IH
fiafia F IH A F IH A
fiefariye F I E A R IH Y E
fiema F I E M A
fime F IH M EH
fini F IH N IH
finji F IH N DZ IH
finjie F I N DZ I EH
firimame F IH R IH M A M EH
fisam F IH S A M
fiye F IH Y E
fura F U R A
furo F U R O
galili G A L I L I
gbeinyee GB EH IH N Y E EH
gbelame GB E L A M EH
gbin GB I N
gbinbia GB I N BB IH A
gbolomake GB O L O M A K EH
gbori GB OH R IH
gboribo GB OH R IH BB OH
giein G IH EH IH N
gonogono G O N O G O N O
gose G O S E
goyegoye G O Y E G O Y E
herod H EH R OH D
hezikaya H EH Z IH K A Y A
hezron H EH Z R OH N
i IH0
ibi I BB I
ibu IH0 BB UH
ibubeleme IH BB UH BB EH L EH M EH
ideri I D E R I
iderimaigoniapu I D E R I M A I G O N I A P UH
igani IH G A N IH
igbiki I GB I K I
igoniapu I G O N I A P UH
ijipt I DZ I P T
```

ikeasun I K E A S UH N
ikiankoroapu IH K IH A N K O R O A P UH
ikiankorobo IH K IH A N K O R O BB OH
imanuel I M A N UH EH L
ineda I N E D A
ineina I N E I N A
ini I N I
inia I N I A
inimgba I N I M GB A
iona I OH N A
ipiamgba IH P IH A M GB A
iri IH2 R IH
iriayee IH R IH A Y E EH
irua IH R UH A
iruapakapaka IH R UH A P A K A P A K A
iwo I W O
iya IH0 Y A
izrel I Z R EH L
izrelapu I Z R EH L A P UH
jehosiafat DZ E H O S IH A F A T
jekob DZ E K OH B
jekonaya DZ E K OH N A Y A
jerusalem DZ E R U S A L EH M
jesi DZ EH S I
jin DZ I N
jizos DZ I Z OH S
jodan DZ OH D A N
jokuma DZ O K U M A
jon DZ OH N
joram DZ O R A M
josaya DZ O S A Y A
josef DZ O S EH F
jotam DZ O T A M
juapu DZ U A P UH
juda DZ U D A
judia DZ U D I A
karakarama K A R A K A R A M A
karakaramasam K A R A K A R A M A S A M
karakaraye K A R A K A R A Y E
kienma K IH EH N M A
kiri K I R I
kirikiri K I R I K I R I
kobirima K O BB I R I M A
koko K O K O
kokoma K OH K OH M A
kokomaye K OH K OH M A Y E
kon K OH N
konme K OH N M EH

konsam K OH N S A M
koroma K O R O M A
koromari K O R O M A R IH
koruapu K O R U A P UH
kpeki KP E K I
kpekisam KP E K I S A M
kpereki KP EH R EH K IH
kraist K R A IH S T
kubie K U BB I E
kubiekuroma K U BB I E K UH R OH M A
kun K U N
kunoma K U N O M A
kura K UH R A
kuro K UH R OH
la L A
lame L A M EH
lasa L A S A
lolia L O L I A
ma M A
maa M A A
mae M A EH
magbolu M A GB OH L UH
makubie M A K U BB I E
mama M A M A
mamgba M A M GB A
manasi M A N A S IH
mangi M A N G IH
mangiso M A N G IH S O
mangisobia M A N G IH S O BB IH A
matan M A T A N
mengi M E N G I
mengisara M E N G I S A R A
mengisarabo M E N G I S A R A BB OH
meri M E R I
mesaya M EH S A Y A
mesi M E S I
mgba M GB A
mi M I
mie M IH EH
mieme M IH EH M EH
miemieboe M IH EH M IH EH BB OH EH
miese M IH EH S E
mieyee M IH EH Y E EH
mimgba M IH M GB A
min M IH N
mine M I N E
minea M I N E A
mioku M IH OH K UH

mono M OH N OH
muari M U A R IH
mume M U M EH
mun M U N
munyee M U N Y E EH
na N A
naa N A A
naame N A A M EH
nama N A M A
namaichuka N A M A I TS U K A
nason N A S OH N
nazaret N A Z A R EH T
nazaretbo N A Z A R EH T BB OH
nde N D E
nemi N E M I
nemime N E M I M EH
nengime N EH N G IH M EH
ngisi N G IH S IH
nwo NW OH
nwose NW OH S E
nyanabo N Y A N A BB OH
nyaname N Y A N A M EH
nyengi N Y E N G I
nyo N Y OH
nyongoro N Y OH N G OH R OH
nyongoroe N Y OH N G OH R OH EH
o O0
obed O B E D
obu OH0 BB UH
obuduko OH BB UH DD U K O
obudukoapu OH BB UH DD U K O A P UH
obuju OH BB UH DZ UH
ogbo O GB O
ogboku O GB O K U
ogono OH G OH N OH
ogu O G U
ojuapura O DZ U A P UH R A
oki OH K IH
okibo OH K IH BB O
okimun O K IH M U N
oko OH K OH
oku OH K UH
oku?ku O K U2 K U2
okue OH K UH EH
okuma OH K UH M A
okwein OH K W EH IH N
oloko O L O K O
olokuabe O L O K U A BB EH

olome OH L OH M EH
omie O M I EH
omine O M I N E
ominea O M I N E A
onwu O NW U
onwuele O NW U E L E
opu O P U
opuama O P U A M A
opuma O P U M A
opuso O P U S OH
ori O R I
oria O R I A
orime OH R IH M EH
oru O R U
oruwu O R U W U
osi O S I
osirosiroba O S I R O S I R O BB A
owu O W U
owuabe O W U A BB EH
owuapuawo O W U A P UH A W OH
owuapuminapu O W U A P UH M IH N A P UH
owubonemika O W U BB OH N E M I K A
owutibi O W U T IH BB IH
owutoku O W U T OH K UH
oyi O Y I
oyighoriapu O Y I G H OH R IH A P UH
oyiineina O Y I I N E I N A
ozaya O Z A Y A
pa P A
paka P A K A
pakabo P A K A BB O
pakabobia P A K A BB O BB IH A
pakabome P A K A BB O M EH
pakapaka P A K A P A K A
pakasome P A K A S O M EH
pakumabome P A K UH M A BB O M EH
pakumame P A K UH M A M EH
pekere P EH K EH R EH
pekereme P EH K EH R EH M EH
pele P EH L EH
perez P EH R EH Z
piki P IH K IH
piri P IH R IH
piriabe P IH R IH A BB EH
piribein P IH R IH BB E I N
piribia P IH R IH BB IH A
pirie P IH R IH EH
pirime P IH R IH M EH

```
pirisa P IH R IH S A
puko P UH K OH
ra R A
ram R A M
re R E
rechel R E TS EH L
rehab R E H A B
rehobuam R EH H O B U A M
rema R E M A
rufu R U F U
sadusiapu S A D U S I A P UH
saki S A K IH
salmon S A L M OH N
sara S A R A
sarabia S A R A BB IH A
sarame S A R A M EH
se S EH
selebia S EH L EH BB IH A
seni S E N I
shealtiel S H E A L T I EH L
si S I
sibupikisam S I BB UH P IH K IH S A M
siki S IH K IH
sima S I M A
sime S I M E
simebia S I M E BB IH A
simekaogbo S I M E K A O GB O
simeme S I M E M EH
simeogbo S I M E O GB O
simeokue S I M E OH K UH EH
sise S IH S EH
so S O
sobia S OH BB IH A
sobie S O BB I E
solomon S O L O M OH N
soni S OH N IH
ta T A
tamuno T A M UH N OH
tamunobere T A M UH N OH BB EH R EH
tari T A R IH
tarime T A R IH0 M EH
taro T A R OH
tekeme T EH K EH M EH
tema T E M A
teme T EH M EH
tibini T IH BB IH N IH
tisam T I S A M
toku T OH K UH
```

tomoni T O M O N I
ton T OH N
tonapu T OH N A P UH
tonari T OH N A R IH
tonme T OH N M EH
toroko T OH R OH K OH
torokodabo T OH R OH K OH D A BB OH
torokoisun T OH R OH K OH IH S UH N
torukweinma T OH R UH K W E I N M A
tubo T UH BB OH
vinpiki V I N P IH K IH
wa W A
wasama W A S A M A
weri W EH R IH
ye Y E
yee Y E EH
yela Y E L A
yesele Y E S EH L EH
yi Y I
yibia Y I BB IH A
yime Y I M EH
yuraya Y U R A Y A
zadok Z A D OH K
zera Z EH R A
zerubabel Z E R U B A B EH L

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... others (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Abadi, M., Isard, M., & Murray, D. G. (2017). A computational model for tensorflow: an introduction. In *Proceedings of the 1st acm sigplan international workshop on machine learning and programming languages* (pp. 1–7).

Acero, A. (1990). Acoustical and environmental robustness in automatic speech recognition. In *Proc. of icassp.*

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., ... others (2016). Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*.

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., ... others (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning* (pp. 173–182).

Andén, J., & Mallat, S. (2011). Multiscale scattering for audio classification. In *Ismir* (pp. 657–662).

Andén, J., & Mallat, S. (2014). Deep scattering spectrum. *IEEE Transactions on Signal Processing*, *62*(16), 4114–4128.

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

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., ... Weber, G. (2019). Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Becchetti, C., & Ricotti, L. P. (1998). *Speech recognition: theory and c++ implementation*. New York: Wiley.

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

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137–1155.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems* (pp. 153–160).

Besacier, L., Barnard, E., Karpov, A., & Schultz, T. (2014a). Automatic speech recognition for under-resourced languages: A survey. *Speech Communication*,

56, 85–100.

Besacier, L., Barnard, E., Karpov, A., & Schultz, T. (2014b). Introduction to the special issue on processing under-resourced languages.

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

Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, *18*(4), 467-479. Retrieved from `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.9919&rep=rep1&type=pdf`

Chen, S. F., & Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on association for computational linguistics* (pp. 310–318).

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in neural information processing systems* (pp. 577–585).

Collobert, R., Puhrsch, C., & Synnaeve, G. (2016). Wav2letter: an end-to-end convnet-based speech recognition system. *arXiv preprint arXiv:1609.03193*.

(2015). *Continuous wavelet transform.*

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

Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, *20*(1), 30–42.

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

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

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

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

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

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

Gales, M. J., Knill, K. M., Ragni, A., & Rath, S. P. (2014). Speech recognition and keyword spotting for low-resource languages: Babel project research at cued. In *Spoken language technologies for under-resourced languages.*

Gales, M. J. F., Watanabe, S., & Fosler-Lussier, E. (2012). Structured discriminative models for speech recognition: An overview. *IEEE Signal Processing*

Magazine, *29*(6), 70–81.

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 1243–1252).

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Ghoshal, A., Swietojanski, P., & Renals, S. (2013). Multilingual training of deep neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on* (pp. 7319–7323).

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Goldman, J.-P. (2011). Easyalign: an automatic phonetic alignment tool under praat.

Goldsborough, P. (2016). A tour of tensorflow. *arXiv preprint arXiv:1610.01178*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*.

Graves, A. (2014). *Supervised sequence labelling with recurrent neural networks*. Springer.

Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on machine learning* (pp. 369–376).

Graves, A., & Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning* (pp. 1764–1772).

Graves, A., Jaitly, N., & Mohamed, A.-r. (2013). Hybrid speech recognition with deep bidirectional lstm. In *Automatic speech recognition and understanding (asru), 2013 ieee workshop on* (pp. 273–278).

Grezl, F., & Fousek, P. (2008). Optimizing bottle-neck features for lvcsr. In *Icassp* (Vol. 8, pp. 4729–4732).

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... others (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Hannun, A. Y., Maas, A. L., Jurafsky, D., & Ng, A. Y. (2014). First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *arXiv preprint arXiv:1408.2873*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the ieee international conference on computer vision* (pp. 1026–1034).

Heafield, K., Pouzyrevsky, I., Clark, J. H., & Koehn, P. (2013, August). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st annual meeting of the association for computational linguistics* (pp. 690–696). Sofia, Bulgaria. Retrieved from `https://kheafield.com/papers/`

edinburgh/estimate_paper.pdf

Hendrycks, D., Lee, K., & Mazeika, M. (2019). Using pre-training can improve model robustness and uncertainty. *arXiv preprint arXiv:1901.09960*.

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

Hermansky, H., & Morgan, N. (1994). Rasta processing of speech. *IEEE transactions on speech and audio processing*, *2*(4), 578–589.

Hwang, K., & Sung, W. (2017). Character-level language modeling with hierarchical recurrent neural networks. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5720–5724).

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the" echo state network" approach* (Vol. 5). GMD-Forschungszentrum Informationstechnik Bonn.

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

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

Kaiser, Ł., & Bengio, S. (2016). Can active memory replace attention? In *Advances in neural information processing systems* (pp. 3781–3789).

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., & Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

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

Karpathy, A. (2015). *The unreasonable effectiveness of recurrent neural network.* Retrieved from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Ketkar, N. (2017). Introduction to pytorch. In *Deep learning with python* (pp. 195–208). Springer.

Kim, S., Hori, T., & Watanabe, S. (2017). Joint ctc-attention based end-to-end speech recognition using multi-task learning. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 4835–4839).

Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016). Character-aware neural language models. In *Aaai* (pp. 2741–2749).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Kuhn, R., & Mori, R. D. (1990). A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(6), 570-583. doi: 10.1109/34.56193

Kumar, S. K. (2017). On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*.

Kunze, J., Kirsch, L., Kurenkov, I., Krug, A., Johannsmeier, J., & Stober, S.

(2017). Transfer learning for speech recognition on a budget. *arXiv preprint arXiv:1706.00290*.

Lamere, P., Kwok, P., Gouvêa, E., Raj, B., Singh, R., Walker, W., ... Wolf, P. (2003). *The cmu sphinx-4 speech recognition system.*

Landahl, H., McCulloch, W. S., & Pitts, W. (1943). A statistical consequence of the logical calculus of nervous nets. *The bulletin of mathematical biophysics*, *5*(4), 135–137.

Lasserre, J. A., Bishop, C. M., & Minka, T. P. (2006). Principled hybrids of generative and discriminative models. In *2006 ieee computer society conference on computer vision and pattern recognition (cvpr'06)* (Vol. 1, pp. 87–94).

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Luong, T., Socher, R., & Manning, C. D. (2013). Better word representations with recursive neural networks for morphology. In *Conll* (pp. 104–113).

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

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

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

McLoughlin, I. (2009). *Applied speech and audio processing: with matlab examples.* Cambridge University Press.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., & Černockỳ, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth annual conference of the international speech communication association.*

Mohamed, A.-r., Dahl, G., & Hinton, G. (2009). Deep belief networks for phone recognition. In *Nips workshop on deep learning for speech recognition and related applications* (Vol. 1, p. 39).

Mohamed, A.-r., Dahl, G. E., Hinton, G., et al. (2012). Acoustic modeling using deep belief networks. *IEEE Trans. Audio, Speech & Language Processing*, *20*(1), 14–22.

Mozilla deepspeech. (2019). Retrieved from `https://voice.mozilla.org/en`

Novotney, S., & Schwartz, R. (2009). Analysis of low-resource acoustic model self-training. In *Tenth annual conference of the international speech communication association.*

Nunamaker Jr, J. F., Chen, M., & Purdin, T. D. (1990). Systems development in information systems research. *Journal of management information systems*, *7*(3), 89–106.

Oliphant, T. (2006–). *NumPy: A guide to NumPy.* USA: Trelgol Publishing. Retrieved from `http://www.numpy.org/` ([Online; accessed <today>])

Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). Librispeech: an asr corpus based on public domain audio books. In *2015 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5206–5210).

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual*

meeting on association for computational linguistics (pp. 311–318).

Paul, D. B., & Baker, J. M. (1992). The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on speech and natural language* (pp. 357–362).

Peddinti, V., Sainath, T., Maymon, S., Ramabhadran, B., Nahamoo, D., & Goel, V. (2014). Deep scattering spectrum with deep neural networks. In *Acoustics, speech and signal processing (icassp), 2014 ieee international conference on* (pp. 210–214).

Pennington, J., Socher, R., & Manning, C. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543). Doha, Qatar: Association for Computational Linguistics. doi: 10.3115/v1/D14-1162

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

Pot, E., Monceaux, J., Gelin, R., & Maisonnier, B. (2009). Choregraphe: a graphical tool for humanoid robot programming. In *Ro-man 2009-the 18th ieee international symposium on robot and human interactive communication* (pp. 46–51).

Povey, D., Burget, L., Agarwal, M., Akyazi, P., Kai, F., Ghoshal, A., ... others (2011). The subspace gaussian mixture model—a structured model for speech recognition. *Computer Speech & Language*, *25*(2), 404–439.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., ... others (2011). The kaldi speech recognition toolkit. In *Ieee 2011 workshop on automatic speech recognition and understanding.*

Ragni, A., & Gales, M. J. (2018). Automatic speech recognition system development in the" wild". In *Interspeech* (pp. 2217–2221).

Ragni, A., Knill, K. M., Rath, S. P., & Gales, M. J. (2014). Data augmentation for low resource languages.

Ramachandran, P., Liu, P. J., & Le, Q. V. (2016). Unsupervised pretraining for sequence to sequence learning. *arXiv preprint arXiv:1611.02683*.

Rosenberg, A., Audhkhasi, K., Sethy, A., Ramabhadran, B., & Picheny, M. (2017, March). End-to-end speech recognition and keyword search on low-resource languages. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)* (p. 5280-5284). doi: 10.1109/ICASSP.2017.7953164

S., C. O. D. (2008). *Okrika: A kingdom of the niger delta* (1st ed.). Port Harcourt, Rivers State, Nigeria: Onyoma Research Publications.

Sainath, T. N., Peddinti, V., Kingsbury, B., Fousek, P., Ramabhadran, B., & Nahamoo, D. (2014). Deep scattering spectra with deep neural networks for lvcsr tasks. In *Fifteenth annual conference of the international speech communication association.*

Sak, H., Senior, A. W., & Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, *abs/1402.1128*. Retrieved from `http://arxiv.org/abs/1402.1128`

Salazar, J., Kirchhoff, K., & Huang, Z. (2019). Self-attention networks for connectionist temporal classification in speech recognition. In *Icassp 2019-2019 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 7115–7119).

Saon, G., Kuo, H.-K. J., Rennie, S., & Picheny, M. (2015). The ibm 2015

english conversational telephone speech recognition system. *arXiv preprint arXiv:1505.05899*.

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

Sifre, L., & Mallat, S. (2013). Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1233–1240).

Sifre, L., & Mallat, S. (2014). Rigid-motion scattering for image classification. *Ph. D. dissertation*.

Simons, G. F., & Fennig, C. D. (2018). *Ethnologue: Languages of the world, twenty-first edition.* (Vol. 2018) (No. 11/11/). Retrieved from `http://www.ethnologue.com`.

Smolensky, P. (1986). *Information processing in dynamical systems: Foundations of harmony theory* (Tech. Rep.). COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Stan, A., Mamiya, Y., Yamagishi, J., Bell, P., Watts, O., Clark, R. A., & King, S. (2016). Alisa: An automatic lightly supervised speech segmentation and alignment tool. *Computer Speech & Language*, *35*, 116–133.

Stevens, S. S., Volkmann, J., & Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, *8*(3), 185–190.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1–9).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Versteegh, M., Thiolliere, R., Schatz, T., Cao, X. N., Anguera, X., Jansen, A., & Dupoux, E. (2015). The zero resource speech challenge 2015. In *Sixteenth annual conference of the international speech communication association.*

Voxforge. (2019). Retrieved from `http://www.voxforge.org`

Vu, N. T., & Schultz, T. (2013). Multilingual multilayer perceptron for rapid language adaptation between and across language families. In *Interspeech* (pp. 515–519).

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., ... Woelfel, J. (2004). Sphinx-4: A flexible open source framework for speech recognition.

Wang, C., Wu, Y., Liu, S., Yang, Z., & Zhou, M. (2019). Bridging the gap between pre-training and fine-tuning for end-to-end speech translation. *arXiv preprint arXiv:1909.07575*.

Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., ... Ochiai, T. (2018). Espnet: End-to-end speech processing toolkit. In *Interspeech* (pp. 2207–2211). Retrieved from `http://dx.doi.org/10.21437/`

`Interspeech.2018-1456` doi: 10.21437/Interspeech.2018-1456

Watanabe, S. . e., & Chien, J.-T. (2015). *Bayesian speech and language processing.* Cambridge: Cambridge University Press.

Woodland, P., & Povey, D. (2000). Large scale discriminative training for speech recognition. In *Asr2000-automatic speech recognition: Challenges for the new millenium isca tutorial and research workshop (itrw).*

Xu, P., & Fung, P. (2013). Cross-lingual language modeling for low-resource speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, *21*(6), 1134–1144.

Young, S. (1996). A review of large-vocabulary continuous-speech. *IEEE Signal Processing Magazine*, *13*(5), 45. doi: 10.1109/79.536824

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., ... others (2002). The htk book. *Cambridge university engineering department*, *3*, 175.

Yu, D., & Deng, L. (2016). *Automatic speech recognition.* Springer.

Yu, D., Deng, L., & Dahl, G. (2010). 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.*

Zeghidour, N., Synnaeve, G., Versteegh, M., & Dupoux, E. (2016). A deep scattering spectrum—deep siamese network pipeline for unsupervised acoustic modeling. In *Acoustics, speech and signal processing (icassp), 2016 ieee international conference on* (pp. 4965–4969).